Решения и примеры для разработчиков баз данных MySQL

Охватывает MySQL 4.0

MySQL

Сборник рецептов

Символ®

O'REILLY®

Поль Дюбуа

MySQL Cookbook

Paul DuBois

O'REILLY®

MySQL
Сборник рецептов

Поль Дюбуа

Санкт-Петербург — Москва
2007

Поль Дюбуа

MySQL. Сборник рецептов

Перевод П. Шера

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Научный редактор	*М. Деркачев*
Редактор	*В. Кузнецов*
Корректор	*И. Чернова*
Верстка	*Н. Гриценко*

Дюбуа П.
MySQL. Сборник рецептов. – Пер. с англ. – СПб: Символ-Плюс, 2006. – 1056 с., ил.
ISBN 5-93286-070-7

«MySQL. Сборник рецептов» Поля Дюбуа – это всеобъемлющий сборник задач, ежедневно возникающих у программистов, их решений и практических примеров. Сборник будет полезен всем пользователям MySQL независимо от уровня их подготовки. Каждой задаче, обсуждаемой в книге, соответствует проработанное решение или рецепт с небольшим фрагментом кода, который можно вставлять прямо в приложение. Работа каждого фрагмента подробно поясняется, что позволяет разобраться, как и почему все это работает, и применить готовые приемы к схожим ситуациям. Материал книги пригодится и опытным разработчикам MySQL – им не придется писать весь код с нуля.

Издание содержит сотни примеров – от простых решений, которые послужат напоминанием, до обработки множества SQL-операторов, которые должны выполняться вместе как единое целое. На веб-сайте книги находятся все сценарии, написанные для API таких языков, как Perl, Python, Java и PHP. В книге обсуждаются: использование сценариев для чтения запросов из файла; формирование запросов; создание сценариев MySQL для Web; взаимодействие с сервером; изменение структуры таблиц за счет добавления, удаления или изменения столбцов; импорт и экспорт данных; выявление, подсчет и удаление дубликатов, а также предотвращение их появления; вычисление различных статистических характеристик.

ISBN 5-93286-070-7
ISBN 0-596-00145-2 (англ)

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, edit@symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.

Подписано в печать 06.09.2006. Формат 70×100 1/16 . Печать офсетная.
Объем 66 печ. л. Доп. тираж 1000 экз. Заказ №
Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

Оглавление

Предисловие

В последние годы система управления базами данных MySQL стала весьма популярной. Наибольшее распространение она получила среди программистов, работающих с Linux и другими продуктами с открытым кодом, но и в коммерческом секторе присутствие MySQL все более заметно. Причин тому несколько: MySQL – быстрая, простая в настройке, использовании и администрировании СУБД. Сама MySQL работает во множестве версий UNIX и Windows, а программы, работающие с MySQL, могут быть написаны на множестве языков. MySQL особенно интенсивно используется совместно с вебсервером для создания веб-сайтов на основе баз данных с динамически формируемым содержимым.

С ростом популярности MySQL растет и количество пользователей, задающих вопросы о том, как поступить в той или иной конкретной ситуации, и на эти вопросы необходимо ответить. Именно такую цель ставила перед собой книга «MySQL. Сборник рецептов». Она создавалась как удобное средство, к которому можно было бы прибегать каждый раз, когда требуется быстрое решение или методика решения конкретной задачи, возникающей при работе с MySQL. Поскольку это «поваренная книга», она содержит рецепты: простые инструкции, которым вы можете следовать вместо того, чтобы писать собственный код с нуля. Книга написана в формате «задача-решение», чтобы быть максимально практичной, удобочитаемой и простой для восприятия. Она включает в себя множество коротких разделов, каждый из которых объясняет, как написать запрос, применить прием или создать сценарий для решения небольшой конкретной задачи. Книга посвящена не разработке законченных приложений – ее смысл в том, чтобы помочь вам справиться с возникающими проблемами.

Например, обычный вопрос: «Как поступать с кавычками и специальными символами в значениях данных при создании запросов?». Это несложно, но информация о том, как выполнить такую операцию, бесполезна, если вы не знаете, с чего начать. Эта книга рассказывает, что надо делать, показывает, с чего следует начать и как действовать дальше. Полученная информация пригодится еще не раз – разобравшись, в чем тут дело, вы сможете применять такой подход к любым типам данных: тексту, изображениям, звуковым или видеоклипам, статьям новостей, архивным файлам, PDF-файлам и документам текстовых процессоров. Другой распространенный вопрос: «Можно ли одновременно обращаться к таблицам двух баз данных?». Ответ: «Да», и сделать это несложно, если знать соответствующий синтаксис SQL. Но пока вы его не знаете, подобная операция будет сложной.

Помимо ответов на приведенные вопросы книга содержит сведения о том:

- Как использовать SQL для выборки, сортировки и суммирования записей.
- Как находить совпадающие или отличающиеся записи в двух таблицах.
- Как выполнять транзакции.
- Как определять интервалы дат и времени, в том числе выполнять вычисления с годами.
- Как удалять повторяющиеся записи.
- Как хранить изображения в MySQL и извлекать их для показа на веб-страницах.
- Как преобразовывать разрешенные значения столбца `ENUM` в переключатели (radio buttons) на веб-странице или значения столбца `SET` – во флажки (checkboxes).
- Как написать предложение (statement) `LOAD DATA` для корректного чтения файлов с данными или поиска некорректных значений в файле.
- Как использовать методики поиска по шаблону так, чтобы справиться с несоответствием между форматом даты MySQL *`CCYY-MM-DD`* и датами в пользовательских файлах.
- Как копировать таблицу или базу данных на другой сервер.
- Как переинициализировать столбец последовательности и почему на самом деле не стоит это делать.

Для того чтобы работать с MySQL, необходимо знать, как общаться с сервером, то есть как использовать SQL – язык, на котором формулируются запросы. Поэтому особое внимание в книге уделено использованию SQL для написания запросов, дающих ответы на характерные вопросы. Для изучения и работы с SQL полезна клиентская программа *mysql*, включенная в дистрибутивы MySQL. Интерактивно используя эту программу, вы можете отправлять SQL-предложения на сервер и получать результаты. Прямой интерфейс с SQL делает клиентскую программу *mysql* исключительно полезной, поэтому ей целиком посвящена глава 1.

Но одного умения писать SQL-запросы недостаточно. Информация, извлекаемая из базы данных, часто нуждается в обработке или должна быть представлена в какой-либо специальной форме для дальнейшего использования. Как поступать с запросами, содержащими сложные взаимозависимости, например, когда результаты одного запроса должны быть использованы в качестве основы для других запросов? Сам SQL не имеет достаточной функциональности для выполнения подобных выборок, что создает проблемы с применением логики принятия решений (decision-based logic) для того, чтобы определить, какие запросы следует выполнить. Или, например, необходимо подготовить отчет с очень нестандартными требованиями к форматированию. Сделать это, используя только SQL, очень сложно. Наличие таких трудностей объясняет, почему в книге придается особая важность еще одному вопросу: как писать программы, взаимодействующие с сервером MySQL через API (application programming interface, программный интерфейс при-

ложения). Узнав, как использовать MySQL в контексте языка программирования, вы сможете применить потенциал MySQL следующим образом:

- Запоминать результат запроса и использовать его в дальнейшем.
- Принимать решение на основе того, успешно или нет выполнен запрос, или в зависимости от содержания возвращенных запросом строк. Реализация управляющей логики не вызывает затруднений, если вы используете API, поскольку базовый язык предоставляет средства выражения логики принятия решений: конструкции if-then-else, циклы while, подпрограммы и т. д.
- Как угодно форматировать и отображать результаты запросов. Если вы пишете сценарий для командной строки, можно сгенерировать простой текст. Если это веб-ориентированный сценарий, можно создать HTML-таблицу. Если речь идет о приложении, которое извлекает информацию для передачи в какую-то другую систему, можно записать файл данных, используя XML.

Комбинируя SQL, язык программирования общего назначения и API клиента MySQL, вы получаете чрезвычайно гибкий инструмент для написания запросов и обработки их результатов. Языки программирования усиливают выразительные возможности, обеспечивая дополнительные средства для выполнения сложных операций над базами данных. Но не думайте, что книга слишком сложна. В ней простыми словами рассказывается, как создавать «кирпичики» – маленькие стандартные блоки, используя простые и понятные приемы.

Комбинируйте предложенные приемы в ваших программах и поверьте – вы получите сколь угодно сложные приложения. Как-никак, основой генетического кода являются всего четыре нуклеиновые кислоты, но комбинации этих базовых элементов породили все то впечатляющее разнообразие биологической жизни, которая нас окружает. А из семи нот талантливый музыкант создаст бесчисленное множество мелодий. Так и вы, взяв несколько простых рецептов, добавив немного воображения и применив их к стоящим перед вами задачам программирования баз данных, создадите, если и не произведение искусства, то наверняка весьма практичные приложения, повышающие эффективность работы.

Используемые в книге API MySQL

Программные интерфейсы MySQL доступны для множества языков, в том числе C, C++, Eiffel, Java, Pascal, Perl, PHP, Python, Ruby, Smalltalk и Tcl.[1] Поэтому перед автором этой «поваренной книги» MySQL стояла достаточно сложная задача. Очевидно то, что необходимо привести множество интересных рецептов работы с MySQL, но какой или какие API для этого использо-

[1] Чтобы получить сведения о доступных в настоящий момент API, посетите портал разработки на веб-сайте MySQL по адресу *http://www.mysql.com/portal/development/html/*.

вать? Если приводить каждый рецепт на всех языках, получится или слишком мало рецептов, или слишком толстая книга! Кроме того, появится ненужная избыточность – реализации на разных языках могут быть чрезвычайно похожи друг на друга. С другой стороны, хочется воспользоваться разнообразием доступных языков, так как часто для решения конкретной задачи один язык подходит больше, чем другой.

Чтобы разрешить дилемму, я выбрал из всего многообразия несколько API и использовал их во всех примерах. Рамки сузились, но все же осталась некоторая свобода выбора. Итак, книга охватывает:

Perl

Используем модуль DBI и специальный драйвер MySQL.

PHP

Используем множество встроенных функций поддержки MySQL.

Python

Используем модуль DB-API и специальный драйвер MySQL.

Java™

Используем специальный драйвер MySQL для интерфейса Java Database Connectivity (JDBC).

Почему именно эти языки? С Perl и PHP все просто. Perl – это, вероятно, наиболее распространенный язык для Web, что объясняется такими его достоинствами, как, например, обработка текста. В частности, он весьма популярен при написании программ для MySQL. PHP также достаточно широко распространен и применяется все чаще. Одной из сильных сторон PHP является та простота, с которой осуществляется доступ к базам данных, поэтому выбор PHP для написания сценариев MySQL представляется естественным. Python и Java не так широко используются в программировании для MySQL, как Perl или PHP, но у каждого из этих языков есть много приверженцев. Например, в Java-сообществе, похоже, наблюдается бум MySQL у разработчиков, использующих технологию JavaServer Pages (JSP) для создания веб-приложений, работающих с базами данных. (Небольшое замечание: после того как я написал книгу «MySQL» (New Riders), именно эти два языка чаще других упоминались в отзывах читателей, сожалеющих об отсутствии сведений. Теперь они должны быть довольны!)

Думаю, что эти языки достаточно полно представляют существующую пользовательскую базу MySQL-программистов. Книга будет полезна и тем, кто предпочитает другие языки; нужно лишь обратить внимание на главу 2, чтобы ознакомиться с основными API, используемыми в книге. Зная, как выполнять операции с базами данных посредством указанных API, вы без труда поймете рецепты из следующих глав и сможете перевести их на языки, не охваченные данным изданием.

Для кого написана эта книга

Книга должна быть полезна всем, кто работает с MySQL, начиная с новичков, использующих базу данных в личных целях, и заканчивая профессиональными разработчиками веб-приложений и баз данных. Книга обращена и к тем, кто в настоящий момент не использует MySQL, но хотел бы это делать. Например, она может быть полезна начинающим, которые хотят познакомиться с системами управления базами данных и понимают, что Oracle не очень-то для этого подходит.

Если вы не очень хорошо знакомы с MySQL, то, вероятно, обнаружите массу вещей, о которых и не подозревали. Если у вас есть некоторый опыт, многие из представленных задач уже могут быть вам знакомы, но если вы не занимались непосредственно их решением, книга сэкономит ваше время. Воспользуйтесь предложенными рецептами, примените их в своих программах вместо того, чтобы биться над написанием кода с нуля.

Книга пригодится и тем, кто еще никогда не использовал MySQL. Можно предположить, что раз это сборник рецептов MySQL, а не PostgreSQL или InterBase, то они подходят только для базы данных MySQL. В некотором смысле так и есть, поскольку некоторые SQL-конструкции существуют только в MySQL. Однако многие запросы содержат только стандартный SQL, переносимый на множество других систем баз данных, так что эти запросы вполне можно использовать после небольших изменений или даже совсем без изменений. Кроме того, некоторые из интерфейсов языков программирования обеспечивают доступ, не зависящий от базы данных, так что их можно применять при установлении соединения с любой базой данных.

Материал в книге усложняется постепенно, поэтому если какой-то рецепт покажется вам тривиальным, пропустите его. Если же какой-то рецепт понять трудно, вероятно, стоит оставить его на время, просмотреть предыдущие рецепты и затем вернуться к непонятому.

Более опытные читатели могут быть удивлены тем, что в книге по MySQL периодически встречаются некие базовые сведения, напрямую не относящиеся к MySQL, например установка переменных окружения. Мой личный опыт подсказывает, что это может быть полезно новичкам MySQL. Одной из причин привлекательности СУБД MySQL является простота использования, поэтому она популярна среди людей, не имеющих серьезных предварительных знаний о базах данных. Но именно они зачастую не могут эффективно использовать MySQL, не зная ответ, например, на такой вопрос: «Как избежать ввода полного путевого имени *mysql* при каждом вызове?» Опытные читатели сразу же ответят, что все дело в правильной установке переменной окружения PATH – в ней должен быть указан каталог установки *mysql*. Но не все так хорошо информированы, например пользователи Windows, привыкшие работать только с графическим интерфейсом, а теперь и пользователи Mac OS X, обнаружат, что в знакомом им пользовательском интерфейсе появились мощные, но немного загадочные команды приложения Terminal. Если вы сами находитесь в таком положении, предложенные базовые сведе-

ния помогут одолеть преграды, мешающие простой и эффективной работе MySQL. Продвинутые пользователи могут просто пропускать такие разделы.

Как построена эта книга

Если эта книга перед вами, то весьма вероятно, что вы занимаетесь разработкой приложения, некоторые моменты реализации которого не ясны. В этом случае уже понятно, с какой проблемой вы столкнулись и можно непосредственно искать соответствующий рецепт в оглавлении или алфавитном указателе. В идеале этот рецепт должен оказаться ровно тем, что вам нужно. Если же этого не случится, должен найтись рецепт похожей задачи, который можно адаптировать к вашей. (Я старался пояснять правила, на которых основана каждая методика, чтобы вы смогли изменить любой рецепт в соответствии с требованиями конкретных приложений).

Возможен и другой вариант – чтение книги от корки до корки вне контекста решения какой-то определенной задачи. Это тоже полезно: вы сможете получить широкое представление о возможностях MySQL, так что я бы рекомендовал время от времени пролистывать эту книгу. Если вы будете иметь общее представление о книге и о том, какие типы задач она рассматривает, работа с ней будет более эффективной. Чтобы упростить поиск рецептов, приведу краткое содержание глав.

Глава 1 «Работа с клиентской программой mysql» посвящена стандартной клиентской программе MySQL, запускаемой из командной строки. Часто именно *mysql* является первым интерфейсом MySQL, с которым сталкиваются пользователи, поэтому важно знать, как с ним работать. Эта программа позволяет интерактивно создавать запросы и видеть их результаты, поэтому она очень удобна для быстрых экспериментов. Ее можно использовать в режиме пакетной обработки для выполнения фиксированных сценариев SQL, можно отправлять ее вывод в другие программы. Кроме того, в этой главе рассматриваются другие способы применения программы *mysql*: нумерация строк вывода, улучшение читаемости длинных строк, порождение различных форматов вывода, а также протоколирование сеансов *mysql*.

В главе 2 «Создание программы для MySQL» приведены базовые элементы программирования для MySQL на каждом из языков API: как установить соединение с сервером, как создать запрос, извлечь результаты и обработать ошибки. Обсуждаются обработка в запросах специальных символов и значений `NULL`, формирование библиотечных файлов для инкапсуляции кода часто используемых операций, а также разнообразные способы указания параметров, необходимых для соединения с сервером.

Глава 3 «Выбор записей» охватывает различные аспекты предложения `SELECT` – основного инструмента извлечения данных с сервера MySQL: указание определенных строк и столбцов для извлечения, выполнение операций сравнения, обработка значений `NULL`, выбор одного из разделов результата запроса, использование временных таблиц и копирование результатов в другие таблицы. В последующих главах некоторые из вопросов будут рассмот-

рены подробнее; здесь же представлен обзор концепций, на которых основываются приведенные решения. Эта глава необходима тем, кто не имеет достаточных базовых знаний по SQL, например не знаком с выборкой записей.

В главе 4 «Работа со строками» изучается работа со строковыми данными: сравнение строк, проверка соответствия шаблону, разбиение и соединение строк. Кроме того, рассматриваются вопросы чувствительности к регистру и выполнения полнотекстового поиска.

Глава 5 «Работа с датами и временем» посвящена датам и времени. Она описывает формат дат MySQL и возможности отображения дат в других форматах. Обсуждаются преобразования различных единиц времени, арифметические действия с датами (определение интервалов или одной даты на основе другой, вычисления с годами). Вводится специальный тип столбца MySQL `TIMESTAMP`.

В главе 6 «Сортировка результатов запроса» рассказывается, как расположить строки результата запроса в нужном порядке. Рассматриваются направление сортировки, обработка значений `NULL`, учет чувствительности строк к регистру, сортировка по дате или по части значения поля. Приводятся примеры сортировки особых видов значений, таких как доменные имена, IP-адреса и значения `ENUM`.

В главе 7 «Формирование итогов» рассказывается о способах оценки общих характеристик множества данных, таких как количество элементов или максимальное, минимальное и среднее значения.

В главе 8 «Изменение таблицы с помощью предложения ALTER TABLE» описываются возможности изменения структуры таблиц за счет добавления, удаления или изменения столбцов, а также создание индексов.

В главе 9 «Получение и использование метаданных» обсуждаются способы получения сведений о результате запроса (например количество строк или столбцов результирующего множества, имя и тип каждого его столбца). Кроме того, в ней рассказано о структуре таблиц и столбцов и о том, как получить от MySQL информацию о доступных базах данных и таблицах.

В главе 10 «Импорт и экспорт данных» описывается обмен данными между MySQL и другими программами. Рассматриваются преобразование формата файла, извлечение столбца из файла данных или изменение порядка столбцов, контроль и проверка правильности данных, перезапись значений, например дат, которые часто приходят в разных форматах. Объясняется, как понять, какие значения создают проблемы при загрузке в MySQL посредством предложения `LOAD DATA`.

В главе 11 «Формирование и использование последовательностей» изучаются столбцы `AUTO_INCREMENT` – специальный механизм MySQL для генерации порядковых номеров. Рассказывается о том, как генерировать новые значения последовательности и определять, какое из значений использовалось последним, как пересчитать столбец, как начать последовательность с заданного значения, как построить таблицу так, чтобы она поддерживала сразу несколько последовательностей. Показано, как использовать значения `AUTO_INCREMENT`

для установления между таблицами связей (relationship) типа «главная-подчиненная» («master-detail») и как обойти при этом подводные камни.

В главе 12 «Использование нескольких таблиц» рассказывается о соединениях (join) – операциях комбинирования строк одной таблицы со строками другой таблицы. Описывается сравнение таблиц для нахождения совпадений или расхождений, составление списков «главная-подчиненная» и получение итоговых значений, перечисление связей типа «многие-ко-многим», изменение и удаление записей одной таблицы в зависимости от содержания другой таблицы.

В главе 13 «Статистические методы» исследуются статистические характеристики: количественные показатели распределения, плотность распределения, регрессии и корреляции. Показано, как расположить множество строк в случайном порядке и как организовать выбор произвольной строки из множества.

Глава 14 «Обработка повторяющихся записей» посвящена выявлению, подсчету и удалению записей-дубликатов. Особое внимание уделяется тому, как избежать их появления.

В главе 15 «Выполнение транзакций» показывается, как обрабатывать несколько предложений SQL, которые должны выполняться вместе как единое целое. Рассматривается режим автофиксации транзакций MySQL (autocommit), фиксация и откат транзакций. Для тех, чья версия MySQL не поддерживает работу с транзакциями, приведены возможные обходные пути.

Глава 16 «Знакомство с MySQL для Web» предлагает основы написания вебсценариев MySQL. Веб-программирование позволяет формировать динамические страницы или собирать информацию для хранения в базе данных. Описывается конфигурирование Apache для работы сценариев Perl, PHP и Python, а также конфигурирование Tomcat для запуска Java-сценариев, написанных с применением нотации JSP. Приводится обзор библиотеки стандартных тегов Java (JSTL – Java Standard Tag Library), которая будет широко использоваться на JSP-страницах в последующих главах.

В главе 17 «Внедрение результатов запросов в веб-страницы» рассказывается о том, как использовать результаты запросов для создания различных HTML-структур: абзацев, списков, таблиц, гиперссылок и навигационных индексов. Описывается хранение изображений в MySQL с их последующим извлечением и отображением. Показано, как отправить загружаемое результирующее множество в броузер.

В главе 18 «Обработка ввода через Web с помощью MySQL» представлены способы получения данных, вводимых пользователями через Web, и их использования для создания новых записей в базе данных или для поиска. Речь идет об обработке форм, в том числе о построении элементов формы, таких как переключатели (radio buttons), всплывающие (pop-up) меню или флажки (checkboxes), на основе информации из базы данных.

В главе 19 «Управление веб-сеансами с помощью MySQL» рассказывается, как писать веб-приложения, которые запоминают информацию и хранят ее

на протяжении нескольких запросов при помощи MySQL. Этот способ удобен при поэтапном сборе информации или при необходимости принять решение в зависимости от действия, совершенного пользователем ранее.

В приложении A «Получение программного обеспечения MySQL» указано, где найти исходные тексты примеров данной книги, а также программное обеспечение, необходимое для использования MySQL и создания собственных программ для баз данных.

В приложении B «JSP и Tomcat для начинающих» приведен общий обзор JSP и правила инсталляции веб-сервера Tomcat. Прочтите его, если вам необходимо установить Tomcat, или вы недостаточно хорошо знакомы с ним, или если вы никогда не писали JSP-страницы.

Наконец, в приложении C «Справочная информация» приводятся источники дополнительной информации по вопросам, затронутым в данном издании. Кроме того, в нем перечислены несколько книг, из которых можно почерпнуть основные сведения об использованных языках программирования.

Ближе к концу книги могут встречаться рецепты, предполагающие знакомство с разделами предыдущих глав. Внутри главы более поздние разделы также часто используют приемы, изученные ранее в главе. Если вы (читая книгу не от начала до конца) обнаружите рецепт, содержащий непонятный прием, посмотрите по предметному указателю или оглавлению, где о нем рассказывалось. Окажется, что методика рассматривалась в более ранних главах. Например, если в рецепте результат запроса сортируется при помощи незнакомой для вас инструкции (clause) `ORDER BY`, обратитесь к главе 6, в которой обсуждаются различные способы сортировки и поясняется, как они работают.

Платформы и версии

Разработка кода в этой книге ведется для MySQL 3.23 и 4.0. Поскольку в MySQL регулярно добавляются новые возможности, некоторые примеры могут не работать на старых версиях. Представляя такие функции, я старался отмечать имеющуюся зависимость от версии.

Я использовал следующие API-модули для MySQL: DBI версии 1.20 и выше, *DBD::mysql* версии 2.0901 и выше, MySQLdb версии 0.9 и выше, MM.MySQL версии 2.0.5 и выше и MySQL Connector/J 2.0.14. DBI требует Perl 5.004_05 или выше вплоть до DBI 1.20, после которой требуется уже Perl 5.005_03 или выше. MySQLdb требует Python 1.5.6 или выше. MM.MySQL и MySQL Connector/J требуют Java SDK 1.1 или выше.

Использованы трансляторы: Perl 5.6 и 5.6.1; PHP 3 и 4; Python 1.5.6, 2.2 и 2.3; Java SDK 1.3.1. Большая часть приведенных сценариев PHP будет работать и в PHP 3 и в PHP 4 (хотя я настойчиво рекомендую PHP 4). Сценарии, требующие PHP 4, отмечены особо.

Несмотря на то что сам я предпочитаю UNIX в качестве среды разработки, книга не накладывает подобных ограничений. Большая часть приведенного материала в равной мере относится и к UNIX и к Windows. При разработке

основной части рецептов использовались операционные системы Mac OS X, RedHat Linux 6.2, 7.0 и 7.3 и различные версии Windows (Me, 98, NT и 2000).

Предполагается, что СУБД MySQL уже установлена и готова к использованию. Кроме того, я считаю, что раз вы планируете написать собственную MySQL-программу, то достаточно хорошо знаете выбранный язык. При необходимости установить программное обеспечение загляните в приложение A. Дополнительные материалы об использованных языках программирования представлены в приложении C.

Соглашения, принятые в этой книге

В оформлении книги приняты следующие соглашения:

`Моноширинный шрифт`

Используется для листингов программ, а также для выделения в тексте элементов программ, например имен переменных и функций.

`Моноширинный полужирный шрифт`

Используется для обозначения текста, который вводит пользователь, чтобы выполнять команды.

`Моноширинный курсив`

Применяется в описании синтаксиса для элементов, которые пользователь при вводе должен заменять конкретными значениями.

Курсив

Используется для выделения URL, имен хостов, каталогов и файлов, команд UNIX и их параметров, а также новых терминов.

Команды часто приведены вместе с приглашением на ввод, чтобы показать контекст, в котором они выполняются. Команды, вводимые в командной строке, показаны с приглашением `%`:

```
% chmod 600 my.cnf
```

Такое приглашение обычно используется в UNIX, но это не означает, что команда может выполняться только в этой среде. Если не оговорено обратное, команды, приведенные с приглашением %, будут, как правило, работать и в Windows.

Если команда должна быть выполнена в UNIX от имени пользователя `root`, то в качестве приглашения используется символ `#`:

```
# chkconfig --add tomcat4
```

Для команд, используемых только в Windows, применяется приглашение `C:\>`:

```
C:\> copy C:\mysql\lib\cygwinb19.dll C:\Windows\System
```

Предложения SQL, выполняемые в клиентской программе *mysql*, начинаются с приглашения `mysql>` и завершаются точкой с запятой:

```
mysql> SELECT * FROM my_table;
```

В примерах, показывающих результаты выполнения запросов в том виде, в каком они представлены в *mysql*, иногда используется многоточие (...) с целью обратить внимание на то, что часть результата удалена и в действительности строк больше, чем показано. Далее приводится запрос, возвращающий длинный список строк, средняя часть которого пропущена:

```
mysql> SELECT name, abbrev FROM states ORDER BY name;
+----------------+--------+
| name           | abbrev |
+----------------+--------+
| Alabama        | AL     |
| Alaska         | AK     |
| Arizona        | AZ     |
...
| West Virginia  | WV     |
| Wisconsin      | WI     |
| Wyoming        | WY     |
+----------------+--------+
```

Примеры, иллюстрирующие синтаксис предложений SQL, не содержат приглашения `mysql>`, но включают точку с запятой, необходимую в качестве разделителя предложений. В этом примере приведено одно предложение:

```
CREATE TABLE t1 (i INT)
SELECT * FROM t2;
```

А в этом примере их два:

```
CREATE TABLE t1 (i INT);
SELECT * FROM t2;
```

Знак точка с запятой принят в качестве разделителя предложений в программе *mysql*. Но при этом данный символ не является элементом собственно языка SQL, поэтому при выполнении SQL-предложений в других программах (например, в Perl или Java) его использовать не надо.

Таким знаком отмечены подсказки, советы и прочие примечания.

Веб-сайт книги

На веб-сайте книги «MySQL Cookbook» вы найдете исходные тексты и тестовые данные примеров:

http://www.kitebird.com/mysql-cookbook/

В книге вы найдете много ссылок на основной набор программ с именем `recipes`. Его можно использовать, чтобы сократить количество вводимой информации. Например, если в книге встречается предложение `CREATE TABLE`, описывающее, как выглядит таблица базы данных, то вместо того чтобы на-

бивать определение таблицы, вы можете найти в каталоге *tables* командный файл SQL и использовать его для создания таблицы. Перейдите в каталог *tables* и выполните следующую команду (*имя_файла* – это имя файла, содержащего предложение CREATE TABLE):

```
% mysql cookbook < имя_файла
```

Если требуется указать имя пользователя или пароль, введите их перед именем базы данных.

Дополнительная информация о дистрибутивах имеется в приложении A.

Сайт Kitebird также предлагает некоторые примеры из книги в режиме онлайн, так что вы можете выполнить их прямо из броузера.

Комментарии и вопросы

Направляйте любые замечания и вопросы, касающиеся этой книги, издателю:

O'Reilly & Associates, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (in the United States or Canada)
(707) 829-0515 (international/local)
(707) 829-0104 (fax)

O'Reilly поддерживает специальную веб-страницу:

http://www.oreilly.com/catalog/mysqlckbk/

Чтобы задать технические вопросы или дать комментарии о книге, пишите по адресу:

bookquestions@oreilly.com

Дополнительная информация о книгах, конференциях, Resource Centers и O'Reilly Network представлена на веб-сайте O'Reilly:

http://www.oreilly.com

Дополнительные ресурсы

Любой язык, имеющий приверженцев, стремится извлечь пользу из работ своего сообщества, так как пользователи пишут код, к которому получают доступ и другие. В частности, Perl обслуживается огромной сетью, созданной для обеспечения внешними модулями, которые не поставляются в рамках собственно Perl. Это Comprehensive Perl Archive Network (CPAN), механизм организации и распространения кода и документации Perl. CPAN содержит модули, обеспечивающие доступ к базам данных, веб-программирование и XML-обработку (отмечены только несколько аспектов, имеющих непосредственное отношение к нашим рецептам). Внешняя поддержка существует и для других языков, хотя в настоящий момент ни для одного она не организована на уровне CPAN. Для PHP существует архив PEAR, а для Python – архив модулей под названием Vaults of Parnassus. Для Java хоро-

шей отправной точкой может послужить сайт компании Sun. В таблице перечислены сайты, на которых можно найти дополнительную информацию.

Язык API	Сайт
Perl	*http://cpan.perl.org/*
PHP	*http://pear.php.net/*
Python	*http://www.python.org/*
Java	*http://java.sun.com/*

Благодарности

Я хотел бы поблагодарить технических редакторов: Тима Олвайна (Tim Allwine), Дэвида Лейна (David Lane), Хью Вильямса (Hugh Williams) и Джастина Зобеля (Justin Zobel). Они внесли ряд полезных замечаний и дополнений, касающихся структуры книги и технической точности. Некоторые члены MySQL AB были столь любезны, что высказали свое мнение о книге. В частности, главный разработчик MySQL Монти Вайдениус (Monty Widenius) тщательно просмотрел текст и выявил множество проблем. Аржен Ленц (Arjen Lentz), Джени Толонен (Jani Tolonen), Сергей Голубчик (Sergei Golubchik) и Зак Грент (Zak Greant) также просматривали отдельные разделы рукописи. Я получил комментарии от Энди Дастмена (Andy Dustman), автора модуля Python MySQLdb, и Марка Мэтьюса (Mark Mathews), автора MM.MySQL и MySQL Connector/J. Спасибо всем, кто сделал эту книгу лучше; все ошибки остаются лишь на моей совести.

Лори Петрики (Laurie Petrycki), ведущий проекта, выносила идею этой книги, осуществляла общее редакторское руководство и подгоняла остальных. Ленни Мюллнер (Lenny Muellner) – специалист, помогавший преобразовать рукопись из исходного формата в нечто, пригодное для печати. Дэвид Чу (David Chu) редактировал книгу. Элли Волькхаузен (Ellie Volckhausen) создала обложку, и я рад видеть там рептилию. Линли Долби (Linley Dolby) – технический редактор и корректор издания, а контроль качества обеспечивали Колин Горман (Colleen Gorman), Даррен Келли (Darren Kelly), Джеффри Холкомб (Jeffrey Holcomb), Брайан Сойер (Brian Sawyer) и Клер Клотье (Clair Cloutier).

Спасибо Тодду Гренье (Todd Greanier) и Шону Лахману (Sean Lahman) из архива бейсбола, вложившим много труда в создание базы данных по бейсболу, использованной во множестве примеров.

Некоторые авторы умеют продуктивно работать, сидя за клавиатурой, но мне лучше пишется вдали от компьютера и желательно с чашечкой кофе. Поэтому я выражаю свою признательность кафе Sow's Ear в Вероне за создание приятных условий для многочасового бумагомарания.

Моя жена Карен оказывала мне всяческую поддержку в том, что оказалось гораздо более долгим мероприятием, чем предполагалось. Я очень ценю ее содействие, а ее терпение достойно восхищения.

1

Работа с клиентской программой mysql

1.0. Введение

СУБД MySQL использует архитектуру клиент-сервер, построенную вокруг сервера, *mysqld*. Сервер – это та самая программа, которая управляет базами данных. Клиентские приложения не делают этого напрямую, они сообщают о ваших намерениях серверу, используя запросы SQL (Structured Query Language – язык структурированных запросов). Клиентская программа или программы устанавливаются локально на той машине, с которой будет осуществляться доступ к MySQL, а сервер может быть установлен где угодно, лишь бы клиентские приложения могли установить с ним соединение. MySQL – это по своей сути СУБД с сетевой структурой, поэтому клиентские приложения могут взаимодействовать с сервером, локально работающим на той же машине или же установленным удаленно, возможно, на другом конце планеты. Клиентские приложения могут выполнять различные функции, но в любом случае каждое из них устанавливает соединение с сервером, отправляет ему SQL-запросы для выполнения операций над базой данных и получает от сервера результаты запроса.

Одним из таких клиентских приложений является *mysql* – программа, включенная в дистрибутив MySQL. При интерактивном использовании *mysql* выводит приглашение на ввод запроса, отправляет его серверу MySQL на выполнение и отображает результат. То есть программа *mysql* полезна и просто сама по себе, но кроме этого она может значительно упростить ваше MySQL-программирование. Часто возникает необходимость, например, просмотреть структуру таблицы, к которой вы обращаетесь из сценария, проверить запрос, прежде чем вставлять его в программу, чтобы убедиться, что он выводит соответствующий результат, и т. д. Для всего этого очень удобно использовать *mysql*. Есть возможность применять *mysql* неинтерактивно, например для чтения запросов из файла или из других программ. Поэтому *mysql* может использоваться внутри сценариев, заданий демона *cron*, а также совместно с другими приложениями.

Глава написана с тем, чтобы помочь вам наиболее эффективно использовать возможности программы *mysql*. Очевидно, что для того чтобы вы могли самостоятельно опробовать рецепты и примеры, предложенные в книге, необходима учетная запись MySQL и база данных, на которой будут производиться операции. Два первых раздела описывают, как использовать *mysql* для того, чтобы все это подготовить. Для наглядности примеры построены в предположении, что:

- сервер MySQL работает на локальном компьютере;
- имя пользователя и пароль для работы с MySQL – это `cbuser` и `cbpass`;
- база данных называется `cookbook` (сборник рецептов).

В своих экспериментах вы можете нарушить любое из этих допущений. Необязательно, чтобы сервер работал на локальной машине, и необязательно использовать имя пользователя, пароль и имя базы данных, приведенные в книге. Разумеется, если вы будете использовать MySQL не совсем так, как написано, придется несколько изменить примеры, чтобы они работали со значениями, соответствующими вашей системе. Но даже если вы будете работать с другими именами, я рекомендовал бы, по крайней мере, создать специальную базу данных именно для опробования приведенных рецептов, а не тестировать их на базе данных, уже используемой в каких-то других целях. Иначе имена существующих таблиц могут вступить в конфликт с именами таблиц из примеров, и вам придется вносить в примеры изменения, которые не потребовались бы, если бы работа велась на отдельной базе данных.

1.1. Создание учетной записи пользователя MySQL

Задача

Вам нужно создать учетную запись для соединения с сервером MySQL, установленным на определенном хосте.

Решение

Используйте предложение (statement) `GRANT` для создания пользовательской учетной записи MySQL. Затем установите соединение с сервером, указав имя и пароль из созданной учетной записи.

Обсуждение

Для соединения с сервером MySQL требуются имя пользователя и пароль. Дополнительно можно указать имя хоста, на котором работает сервер. Если параметры соединения не заданы явно, *mysql* использует значения по умолчанию. Например, если не указано имя хоста, *mysql* обычно считает, что сервер работает на локальном компьютере.

Следующий пример показывает, как использовать программу *mysql* для соединения с сервером и выполнения предложения `GRANT`, которое создаст пользовательскую учетную запись с правами на доступ к базе данных `cookbook`.

Аргументами *mysql* являются: *-h localhost* (указывает на соединение с сервером MySQL, работающим на локальном компьютере), *-p* (сообщает программе *mysql* о необходимости запрашивать пароль) и *-u root* (соединение от имени MySQL-пользователя `root`). Текст, вводимый пользователем, выделен полужирным шрифтом, а вывод программы – светлым шрифтом:

```
% mysql -h localhost -p -u root
Enter password: ******
mysql> GRANT ALL ON cookbook.* TO 'cbuser'@'localhost' IDENTIFIED BY 'cbpass';
Query OK, 0 rows affected (0.09 sec)
mysql> QUIT
Bye
```

Если после ввода команды *mysql* (первая строка) вы получите сообщение, указывающее на то, что программа не найдена или введена неправильная команда, обратитесь к рецепту 1.7. Если же этого не произойдет, то в ответ на запрос пароля введите пароль пользователя `root` там, где в примере вы видите `******`. (Если MySQL-пользователь `root` не имеет пароля, просто нажмите клавишу Return). Затем введите предложение `GRANT`, как показано выше.

Если вы работаете с базой данных, отличной от `cookbook`, подставьте ее имя везде, где в предложении `GRANT` встречается `cookbook`. Обратите внимание на то, что даже если учетная запись пользователя уже заведена, выдача прав на доступ к базе данных все равно необходима. Однако в этом случае можно опустить часть `IDENTIFIED BY 'cbpass'`, так как оставив ее, вы измените текущий пароль пользователя.

Часть `'cbuser'@'localhost'` содержит информацию о хосте, *с которого* вы соединяетесь с сервером MySQL, чтобы получить доступ к базе данных `cookbook`. Чтобы создать пользователя для соединения с сервером, работающим на локальной машине, используйте `localhost`, как показано. Если же вы планируете устанавливать соединения с сервером с другого компьютера, подставьте его имя в предложение `GRANT`. Например, если вы как `cbuser` соединяетесь с сервером с хоста *xyz.com*, то предложение `GRANT` должно выглядеть так:

```
mysql> GRANT ALL ON cookbook.* TO 'cbuser'@'xyz.com' IDENTIFIED BY 'cbpass';
```

Вероятно, некоторых из вас могла смутить некоторая парадоксальность описанной процедуры. Для того чтобы создать учетную запись пользователя для

Учетные записи MySQL и учетные записи для входа в систему

Учетные записи MySQL – это не то же самое, что учетные записи для входа в вашу операционную систему. Например, MySQL-пользователь `root` не имеет ничего общего с UNIX-пользователем `root`, несмотря на то что их имена совпадают. То есть вполне вероятно, что у них разные пароли. Кроме того, это означает, что нельзя создать учетную запись MySQL, создав учетные записи для входа в систему; вместо этого необходимо использовать предложение `GRANT`.

соединения с сервером MySQL, необходимо установить соединение с сервером с целью выполнить предложение GRANT. Предполагается, что вы уже имеете возможность соединиться с сервером как MySQL-пользователь root, так как GRANT может применяться только таким пользователем, как root, – с правами администратора по созданию учетных записей. Если вы не можете соединиться с сервером от имени root, обратитесь к вашему администратору MySQL с тем, чтобы он выполнил для вас предложение GRANT. После того как это сделано, вы можете соединяться с сервером под собственным именем, создавать свою базу данных и вообще работать самостоятельно.

1.2. Создание базы данных и тестовой таблицы

Задача

Вы хотите создать базу данных и таблицы в ней.

Решение

Применяйте предложение CREATE DATABASE для создания базы данных, предложение CREATE TABLE – для создания каждой из таблиц, которые предполагается использовать, и предложение INSERT – для добавления записей в таблицы.

Обсуждение

Предложение GRANT определяет права для работы с базой данных cookbook, но не создает ее. Прежде чем вы сможете работать с базой данных, необходимо ее явно создать. В данном разделе показано, как это сделать, а также как создать таблицу и наполнить ее тестовыми данными, которые будут использоваться в примерах следующих разделов.

После того как учетная запись cbuser создана, проверьте, удастся ли с ее помощью соединиться с сервером MySQL. Установив соединение, создайте базу данных. Выполните приведенные ниже команды на компьютере, имя которого было указано в предложении GRANT (за *-h* должно следовать имя хоста, на котором работает MySQL):

```
% mysql -h localhost -p -u cbuser
Enter password: cbpass
mysql> CREATE DATABASE cookbook;
Query OK, 1 row affected (0.08 sec)
```

Теперь у вас есть база данных, и можно создавать в ней таблицы. Следующие предложения указывают cookbook в качестве базы данных по умолчанию, создают простую таблицу и заполняют ее записями:[1]

[1] Если вы не хотите вводить весь текст предложений INSERT (и я вас за это не осуждаю), загляните вперед – в рецепт 1.12, в котором приведена сокращенная форма записи. При желании сократить количество вводимого текста *во всех* предложениях опять-таки загляните вперед – в рецепт 1.15.

```
mysql> USE cookbook;
mysql> CREATE TABLE limbs (thing VARCHAR(20), legs INT, arms INT);
mysql> INSERT INTO limbs (thing,legs,arms) VALUES('human',2,2);
mysql> INSERT INTO limbs (thing,legs,arms) VALUES('insect',6,0);
mysql> INSERT INTO limbs (thing,legs,arms) VALUES('squid',0,10);
mysql> INSERT INTO limbs (thing,legs,arms) VALUES('octopus',0,8);
mysql> INSERT INTO limbs (thing,legs,arms) VALUES('fish',0,0);
mysql> INSERT INTO limbs (thing,legs,arms) VALUES('centipede',100,0);
mysql> INSERT INTO limbs (thing,legs,arms) VALUES('table',4,0);
mysql> INSERT INTO limbs (thing,legs,arms) VALUES('armchair',4,2);
mysql> INSERT INTO limbs (thing,legs,arms) VALUES('phonograph',0,1);
mysql> INSERT INTO limbs (thing,legs,arms) VALUES('tripod',3,0);
mysql> INSERT INTO limbs (thing,legs,arms) VALUES('Peg Leg Pete',1,2);
mysql> INSERT INTO limbs (thing,legs,arms) VALUES('space alien',NULL,NULL);
```

Таблица с именем `limbs` содержит три столбца для записи количества ног/ножек (`legs`) и рук/ручек (`arms`) различных одушевленных и неодушевленных объектов (`thing`). (Физиология инопланетянина из последней строки такова, что невозможно определить значения, соответствующие количеству рук и ног, поэтому использованы значения `NULL` — «неизвестная величина».)

Чтобы проверить, содержит ли таблица предполагаемые данные, выполните предложение `SELECT`:

```
mysql> SELECT * FROM limbs;
+--------------+------+------+
| thing        | legs | arms |
+--------------+------+------+
| human        |    2 |    2 |
| insect       |    6 |    0 |
| squid        |    0 |   10 |
| octopus      |    0 |    8 |
| fish         |    0 |    0 |
| centipede    |  100 |    0 |
| table        |    4 |    0 |
| armchair     |    4 |    2 |
| phonograph   |    0 |    1 |
| tripod       |    3 |    0 |
| Peg Leg Pete |    1 |    2 |
| space alien  | NULL | NULL |
+--------------+------+------+
12 rows in set (0.00 sec)
```

Теперь у вас есть база данных и таблица, которые можно использовать для выполнения тестовых запросов.

1.3. Запуск и остановка mysql

Задача

Вы хотите запустить и остановить программу *mysql*.

Решение

Запуск *mysql* выполняется из командной строки с указанием необходимых параметров соединения (connection). Для выхода из *mysql* используйте предложение `QUIT`.

Обсуждение

Для запуска *mysql* просто введите ее имя в командной строке. Если программа будет корректно запущена, вы увидите приветственное сообщение, за которым будет следовать приглашение на ввод `mysql>` — программа готова принимать запросы. Посмотрите, как может выглядеть приветственное сообщение (для экономии места оно не приводится в последующих примерах):

```
% mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 18427 to server version: 3.23.51-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Если *mysql* запускается, но сразу же завершает работу с сообщением «access denied», это означает, что необходимо указать параметры соединения. Обычно требуются такие параметры, как имя хоста для установления соединения (компьютера, на котором работает сервер MySQL, например `localhost`), ваше имя пользователя MySQL и пароль, например:

```
% mysql -h localhost -p -u cbuser
Enter password: cbpass
```

Далее в примерах я буду приводить команды *mysql* без опций параметров соединения. Будем считать, что все необходимые параметры введены или в командной строке, или в файле опций (рецепт 1.4), так что нет необходимости набирать их заново при каждом вызове *mysql*.

Если у вас нет имени пользователя и пароля для работы с MySQL, необходимо получить разрешение (permission) на использование сервера MySQL, как описано в рецепте 1.1.

Синтаксис и значения опций параметров соединения по умолчанию приведены в табл. 1.1. Опции могут записываться в сокращенной форме с одним дефисом или в полной форме с двумя дефисами.

Таблица 1.1. Параметры соединения

Тип параметра	Возможные форматы	Значение по умолчанию
Имя хоста	*-h имя_хоста* *--host=имя_хоста*	`localhost`
Имя пользователя	*-u имя_пользователя* *--user=имя_пользователя*	Имя вашей учетной записи
Пароль	*-p* *--password*	Отсутствует

Как видно из таблицы, для пароля нет значения по умолчанию. Введите *--password* или *-p*, а когда *mysql* выведет приглашение, введите свой пароль:

```
% mysql -p
Enter password:              ← введите здесь свой пароль
```

При желании можно указать пароль непосредственно в командной строке, используя *-пароль* (обратите внимание на то, что после *-p* нет пробела) или *--password=пароль*. Я бы не рекомендовал выполнять такую операцию на многопользовательских компьютерах, поскольку ваш пароль становится доступным другим пользователям, применяющим такие команды, как *ps*, которые выводят информацию о процессе.

Если при попытке запуска *mysql* вы получаете сообщение о том, что программа *mysql* не найдена или выдана неправильная команда, это означает, что командный интерпретатор не знает, где установлена программа *mysql*. Обратитесь к рецепту 1.7.

Чтобы завершить сеанс *mysql*, выполните предложение `QUIT`:

```
mysql> QUIT
```

Кроме того, сеанс можно завершить при помощи предложения `EXIT` или (в UNIX) нажатием клавиш Ctrl-D.

Способ, которым вы определяете параметры соединения для *mysql*, подходит и для других MySQL-программ, таких как *mysqldump* и *mysqladmin*. Например, некоторые операции *mysqladmin* доступны только пользователю MySQL `root` и для их выполнения необходимо указать опции имени и пароля для этого пользователя:

```
% mysqladmin -p -u root shutdown
Enter password:
```

1.4. Задание параметров соединения в файлах опций

Задача

Вы не хотите вводить параметры соединения в командной строке при каждом вызове *mysql*.

Решение

Поместите параметры в файл опций (option file).

Обсуждение

Чтобы не вводить параметры соединения вручную, можно поместить их в файл опций, чтобы *mysql* считывала их автоматически. В UNIX ваш персональный файл опций называется *.my.cnf* и хранится в домашнем каталоге.

Существуют также общие файлы опций, используемые администраторами для указания параметров, которые будут применены глобально ко всем пользователям. Используйте файл */etc/my.cnf* или *my.cnf* из каталога данных сервера MySQL. Если вы работаете в Windows, то можете использовать такие файлы опций, как *C:\my.cnf*, файл *my.ini* из системного каталога или *my.cnf* из каталога данных сервера.

При отображении файлов Windows может не показывать расширения имен, так что файл *my.cnf* может быть показан просто как *my*. Используя средства Windows, вы можете включить отображение расширений. В окне DOS для вывода полных имен используйте команду `DIR`.

Посмотрим на формат, используемый при создании файла опций MySQL:

```
# общие опции соединения клиентской программы
[client]
host=localhost
user=cbuser
password=cbpass

# специальные опции программы mysql
[mysql]
no-auto-rehash
# программа постраничного вывода
# для интерактивной работы
pager=/usr/bin/less
```

Приведем основные характеристики этого формата:

- Строки записываются в группы. В первой строке группы указывается имя группы в квадратных скобках, а в остальных – опции, относящиеся к данной группе. Файл в примере содержит группы `[client]` и `[mysql]`. Внутри группы строки записываются в формате *имя=значение*, где *имя* – это название опции (без дефиса в начале), а *значение* – это значение опции. Если опция не принимает никаких значений (как в случае с `no-auto-rehash`), название приводится в списке само по себе, без части *=значение*.
- Если какие-то параметры вам не нужны, просто исключите соответствующие строки. Например, если вы обычно соединяетесь с хостом по умолчанию (`localhost`), строка `host` не нужна. Если имя пользователя MySQL совпадает с регистрационным именем для операционной системы, можно пропустить строку `user`.
- В файлах опций разрешается использовать только несокращенный формат опции (в отличие от командной строки, где параметры могут указываться как в краткой, так и в полной форме). Например, в командной строке имя хоста может быть задано как *-h имя_хоста* или *--host=имя_хоста*, а в файле опций разрешена только запись `host=`*имя_хоста*.
- Опции часто используются для параметров соединения (таких как `host`, `user` и `password`). Однако в файле могут содержаться опции и другого назначения. Опция `pager` из группы `[mysql]` задает программу постраничного

вывода, которую *mysql* должна использовать для отображения выходных данных в интерактивном режиме. Эта опция никак не связана со способом соединения программы с сервером.

- Обычно параметры клиентского соединения указываются в группе `[client]`. Эта группа используется всеми стандартными клиентскими программами MySQL, то есть создавая файл опций для *mysql*, вы тем самым упрощаете и вызов других программ, таких как *mysqldump* и *mysqladmin*.
- В файле опций может быть определено несколько групп. Общепринятым правилом является следующее: программа считывает параметры из группы `[client]` и группы, названной по имени самой программы. Благодаря этому можно выделить в отдельный список общие параметры, которые должны использовать все клиентские программы, сохранив при этом возможность указывать специальные опции только для какой-то конкретной программы. Предыдущий пример файла опций иллюстрирует правило для программы *mysql*: общие параметры соединения берутся из группы `[client]`, к ним добавляются опции `no-auto-rehash` и `pager` группы `[mysql]`. (Если поместить специальные опции *mysql* в группу `[client]`, то для всех остальных программ, использующих эту группу, будут выданы сообщения об ошибке «unknown option» (неизвестная опция), что приведет к невозможности их корректной работы).
- Если один параметр встречается в файле опций несколько раз, более высокий приоритет имеет значение, указанное последним. Это значит, что следует помещать группы опций для какой-то конкретной программы после группы `[client]`, с тем, чтобы в случае перекрывания опций, заданных в обеих группах, вместо общих опций использовались опции, определенные именно для данной программы.
- Строки, начинающиеся с `#` или `;`, воспринимаются как комментарии и игнорируются. Пустые строки также игнорируются.
- Файлы опций должны быть простыми текстовыми файлами. Если вы создаете файл опций в текстовом процессоре, использующем по умолчанию нетекстовый формат, необходимо явно сохранить файл как текстовый. Пользователи Windows, обратите внимание на это замечание!
- В опциях, определяющих пути к файлам или каталогам, в качестве разделителей должны использоваться символы /, даже в Windows.

Если вы хотите посмотреть, какие опции *mysql* возьмет из файла, выполните команду:

```
% mysql --print-defaults
```

Также можно использовать программу *my_print_defaults*, которая принимает в качестве аргументов названия групп файла опций, которые она должна прочитать. Например, *mysql* берет опции из групп `[client]` и `[mysql]`, поэтому чтобы проверить, какие значения она получит из файла опций, необходимо выполнить команду:

```
% my_print_defaults client mysql
```

1.5. Защита файлов опций

Задача

Ваше имя пользователя и пароль для MySQL хранятся в файле опций, и не хотелось бы, чтобы их прочитали другие пользователи.

Решение

Измените режим работы с файлом, сделав его доступным только для вас.

Обсуждение

Если вы работаете в многопользовательской ОС, такой как UNIX, то необходимо защитить файл опций, чтобы другие пользователи не смогли воспользоваться вашим паролем для работы под вашим именем с сервером MySQL. Команда *chmod* делает файл приватным (private), изменяя его режим так, чтобы доступ был возможен только для вас:

```
% chmod 600 .my.cnf
```

1.6. Комбинирование параметров файла опций с параметрами командной строки

Задача

Вы не хотите хранить MySQL-пароль в файле опций, но и не хотите вручную вводить имя пользователя и имя хоста.

Решение

Поместите имя пользователя и имя хоста в файл опций, а пароль указывайте интерактивно при вызове *mysql*; программа просматривает в поиске параметров соединения и командную строку, и файл опций. Если опция определена и там и там, приоритет имеет значение, указанное в командной строке.

Обсуждение

Сначала *mysql* считывает параметры соединения из файла опций, затем проверяет, нет ли дополнительных параметров в командной строке. То есть одни опции можно задать первым способом, а другие – вторым.

Параметры командной строки имеют более высокий приоритет, поэтому если по какой-то причине требуется изменить значение параметра из файла опций, просто укажите его в командной строке. Например, в файле опций могут быть приведены ваше обычное имя пользователя и пароль. Если же вдруг необходимо подключиться к серверу как пользователь `root`, укажите опции имени и пароля пользователя в командной строке, чтобы заменить значения из файла опций:

```
% mysql -p -u root
```

Если в файле опций задан пароль, а требуется явно указать «не использовать пароль», введите в командной строке *-p*, а затем, когда *mysql* выведет приглашение на ввод пароля, просто нажмите клавишу Return:

```
% mysql -p
Enter password:        ← здесь нажмите клавишу Return
```

1.7. Что делать, если не удается найти mysql

Задача

При вызове *mysql* из командной строки может случиться так, что командный интерпретатор не обнаружит ее.

Решение

Добавьте каталог установки *mysql* в переменную PATH. Тогда запуск *mysql* из любого каталога не будет представлять проблемы.

Обсуждение

Если вы вызываете программу *mysql*, а оболочка или командный интерпретатор не находит ее, будет выведено сообщение об ошибке, которое может выглядеть, например, так (в UNIX):

```
% mysql
mysql: Command not found.
```

А при работе в Windows вы можете увидеть:

```
C:\> mysql
Bad command or invalid filename
```

Чтобы сообщить оболочке, где искать программу *mysql*, можно каждый раз вводить полное путевое имя при запуске. В UNIX команда может быть такой:

```
% /usr/local/mysql/bin/mysql
```

А в Windows:

```
C:\> C:\mysql\bin\mysql
```

Но ввод длинных полных имен быстро надоедает. Чтобы избежать этого, можно перед запуском переходить в каталог установки *mysql*. Но я лично *не* рекомендовал бы так поступать. Этот путь неизбежно приведет к тому, что все ваши файлы данных и командные файлы запросов попадут в тот же каталог, что и *mysql*, загромождая тем самым каталог, предназначенный исключительно для размещения программ.

Более удачный способ решения заключается в том, чтобы включить каталог установки *mysql* в переменную окружения PATH, которая содержит перечень полных имен каталогов, которые оболочка просматривает в поиске команд (см. рецепт 1.8). Вы сможете вызывать программу *mysql* из любого каталога, просто вводя ее имя, и оболочка будет находить ее. И не нужно набивать

длинное полное имя. Есть и дополнительное преимущество: поскольку вы без труда можете вызывать *mysql* откуда угодно, необязательно помещать файлы с данными в тот же каталог, что и *mysql*. А если над вами не довлеет необходимость вызова *mysql* из строго определенного каталога, вы получаете возможность организовать файлы любым представляющимся разумным способом, не завися от искусственных ограничений. Например, можно создать внутри домашнего каталога специальный каталог для каждой рабочей базы данных и помещать файлы, относящиеся к каждой базе данных, в соответствующий каталог.

Я обращаю ваше внимание на пути поиска, так как очень часто сталкиваюсь с вопросами пользователей на эту тему. Люди не подозревают о существовании таких путей и проводят всю работу, связанную с MySQL, в каталоге *bin*, где установлен сервер *mysql*. Особенно это касается пользователей Windows. Возможно, причина в том, что не считая Windows NT и ее производных, справочная система Windows (Windows Help) не предлагает никакой информации о поиске, осуществляемом командным интерпретатором, и о его настройке. (Похоже, справочная система Windows считает, что людям опасно знать, как сделать что-то полезное для себя.)

Пользователям Windows можно предложить еще одно решение, позволяющее избежать ввода полных имен и перехода в каталог *mysql*, – создать ярлык (shortcut) и разместить его в удобном месте. Тогда для запуска *mysql* достаточно будет просто открыть этот ярлык. Чтобы указать опции командной строки или каталог запуска, отредактируйте свойства ярлыка. Если вы вызываете *mysql* с разными опциями, вероятно, удобно будет создать отдельный ярлык для каждого набора опций. Например, один ярлык – для соединения с сервером под именем обычного пользователя для повседневной работы, а второй – для соединения от имени MySQL-пользователя `root` в целях администрирования.

1.8. Установка переменных окружения

Задача

Необходимо изменить настройки рабочей среды, например такой параметр оболочки, как `PATH`.

Решение

Отредактируйте соответствующий загрузочный файл (startup file) оболочки. Для пользователей Windows NT альтернативным вариантом является использование панели управления системы.

Обсуждение

Оболочка, или командный интерпретатор (shell), который вы используете для запуска программ из командной строки, включает в себя окружение, в котором можно хранить значения переменных. Некоторые из этих переменных

используются самой оболочкой. Например, она использует переменную PATH для определения того, в каких каталогах следует искать программы, например *mysql*. Другие переменные могут использоваться другими программами (например, переменная PERL5LIB указывает, где искать библиотечные файлы, используемые сценариями Perl).

Оболочка определяет синтаксис, применяемый при установке переменных окружения, а также загрузочный файл, в который они помещаются. Стандартные загрузочные файлы для различных оболочек приведены в табл. 1.2. Если вы никогда ранее не заглядывали в загрузочные файлы вашей оболочки, это отличная возможность познакомиться с их содержимым.

Таблица 1.2. Загрузочные файлы оболочек

Оболочка	Возможные загрузочные файлы
csh, *tcsh*	*.login*, *.cshrc*, *.tcshrc*
sh, *bash*, *ksh*	*.profile.bash_profile*, *.bash_login*, *.bashrc*
DOS-оболочка	*C:\AUTOEXEC.BAT*

В следующем примере показано, как определить переменную окружения PATH так, чтобы она содержала каталог установки *mysql*. Предполагается, что в одном из ваших загрузочных файлов для PATH уже задано какое-то значение. Если это не так, просто добавьте в один из файлов соответствующую строку (или несколько строк).

Если вы читаете этот раздел по ссылке из другой главы, вам, вероятно, требуется изменить отличную от PATH переменную. Следуйте этой же инструкции, поскольку синтаксис остается неизменным.

Переменная PATH содержит имена одного или нескольких каталогов. При работе в UNIX, если переменная окружения содержит несколько полных имен, принято использовать в качестве их разделителя символ двоеточия (:). В Windows полные путевые имена сами могут содержать двоеточия, поэтому разделителем является точка с запятой (;).

Чтобы установить переменную PATH, используйте инструкции, относящиеся к вашей оболочке:

- В *csh* и *tcsh* найдите в загрузочных файлах команду `setenv PATH` и добавьте в эту строку соответствующий каталог. Предположим, что путь поиска задается у вас следующей строкой в файле *.login*:

  ```
  setenv PATH /bin:/usr/bin:/usr/local/bin
  ```

  Если программа *mysql* установлена в каталог */usr/local/mysql/bin*, добавьте этот каталог в полное имя, изменив строку `setenv` следующим образом:

  ```
  setenv PATH /usr/local/mysql/bin:/bin:/usr/bin:/usr/local/bin
  ```

  Также можно задать путь при помощи `set path`:

  ```
  set path = (/usr/local/mysql/bin /bin /usr/bin /usr/local/bin)
  ```

- Для оболочки из семейства оболочек Борна, таких как *sh*, *bash* или *ksh*, найдите в загрузочных файлах строку, в которой устанавливается и экспортируется переменная PATH:

```
export PATH=/bin:/usr/bin:/usr/local/bin
```

  Присваивание значения и экспортирование могут находиться в разных строках:

```
PATH=/bin:/usr/bin:/usr/local/bin
export PATH
```

  Измените установку следующим образом:

```
export PATH=/usr/local/mysql/bin:/bin:/usr/bin:/usr/local/bin
```

  или:

```
PATH=/usr/local/mysql/bin:/bin:/usr/bin:/usr/local/bin
export PATH
```

- При работе в Windows найдите в файле *AUTOEXEC.BAT* строку, в которой устанавливается переменная PATH. Она может выглядеть так:

```
PATH=C:\WINDOWS;C:\WINDOWS\COMMAND
```

  Или так:

```
SET PATH=C:\WINDOWS;C:\WINDOWS\COMMAND
```

  Измените значение переменной PATH так, чтобы она содержала каталог установки *mysql*. Например, если программа установлена в каталоге *C:\mysql\bin*, то переменная PATH должна выглядеть так:

```
PATH=C:\mysql\bin;C:\WINDOWS;C:\WINDOWS\COMMAND
```

  или:

```
SET PATH=C:\mysql\bin;C:\WINDOWS;C:\WINDOWS\COMMAND
```

- При работе в Windows NT есть еще один способ изменения значения переменной PATH – с помощью панели управления System (вкладка Advanced или Environment). В других версиях Windows можно использовать редактор реестра. К сожалению, ключ реестра, содержащий значение пути, в разных версиях Windows называется по-разному. Я, например, видел, что названия соответствующих ключей в Windows ME и Windows 98 отличаются, а в Windows 95 я вообще не смог обнаружить этот ключ. Так что, вероятно, проще все же отредактировать *AUTOEXEC.BAT*.

После того как переменная окружения установлена, необходимо сделать так, чтобы сделанные изменения вошли в силу. В UNIX можно выйти из системы и войти в нее заново. В Windows, если значение PATH задано через панель управления, можно просто открыть новое окно DOS. Если же редактировался файл *AUTOEXEC.BAT*, вам придется перезагрузить компьютер.

1.9. Создание запросов

Задача

Вы запустили *mysql* и теперь хотите отправить запрос серверу MySQL.

Решение

Просто введите запрос, не забыв сообщить программе *mysql* о том, где он заканчивается.

Обсуждение

Чтобы выдать запрос в командной строке после приглашения на ввод `mysql>`, наберите текст запроса, добавьте в конце предложения точку с запятой и нажмите клавишу Return. Необходимо явно вводить символ конца предложения, так как *mysql* не интерпретирует нажатие клавиши Return как признак конца (предложению разрешено занимать несколько строк ввода). Наиболее распространенным символом конца является точка с запятой, но можно также использовать `\g` (от go – «иди»). То есть два приведенных ниже примера являются равносильными способами формирования одного и того же запроса, хотя они введены по-разному и заканчиваются также по-разному:[1]

```
mysql> SELECT NOW();
+---------------------+
| NOW()               |
+---------------------+
| 2001-07-04 10:27:23 |
+---------------------+
mysql> SELECT
    -> NOW()\g
+---------------------+
| NOW()               |
+---------------------+
| 2001-07-04 10:27:28 |
+---------------------+
```

Обратите внимание, что во второй введенной строке второго примера изменился вид приглашения на ввод: вместо `mysql>` отображается `->`. Таким способом *mysql* оповещает вас о том, что она все еще ждет появления символа конца запроса.

Убедитесь в том, что сам запрос не содержит ни точку с запятой (`;`), ни последовательность `\g`, служащую признаком конца запроса. Выполнение этого условия необходимо именно для *mysql*, так как программа распознает такие символы и удаляет их из введенного текста до того, как отправить запрос на сервер MySQL. Не забывайте об этом, когда будете писать собственные

[1] В примерах этой книги ключевые слова SQL, такие как `SELECT`, будут отображаться в верхнем регистре. Это сделано только для большей наглядности; вы можете вводить ключевые слова в любом регистре.

программы, отправляющие запросы на сервер (в следующей главе будут такие примеры). В этом случае не нужно добавлять символы конца; конец строки запроса сам по себе означает конец запроса. В действительности добавление завершающего символа может привести к ошибке выполнения запроса.

1.10. Выбор базы данных

Задача

Вы хотите сообщить программе *mysql*, с какой базой ей следует работать.

Решение

Укажите имя базы данных в командной строке *mysql* или выполните внутри *mysql* предложение `USE`.

Обсуждение

Для выполнения запроса, который обращается к таблице (а так поступает большинство запросов), необходимо указать, какой базе данных эта таблица принадлежит. Можно использовать полную ссылку на таблицу, начинающуюся с имени базы данных. (Например, `cookbook.limbs` – это ссылка на таблицу `limbs` базы данных `cookbook`.) Для удобства MySQL также предоставляет возможность выбрать базу данных по умолчанию (для текущей работы) и ссылаться на таблицы такой базы без явного повторения имени базы данных. Можно задать имя базы данных в командной строке при запуске *mysql*:

```
% mysql cookbook
```

Если при запуске *mysql* вы указываете в командной строке опции, такие как параметры соединения, они должны предшествовать определяемому имени базы данных:

```
% mysql -h host -p -u user cookbook
```

Если программа *mysql* уже запущена, вы можете задать базу данных (или изменить уже сделанную установку), выполнив предложение `USE`:

```
mysql> USE cookbook;
Database changed
```

Если вы забыли, с какой базой данных работаете в настоящий момент, или не уверены, что помните правильно (что нередко случается при работе с несколькими базами при обращении то к одной, то к другой в течение одного сеанса *mysql*), используйте следующее предложение:

```
mysql> SELECT DATABASE();
+------------+
| DATABASE() |
+------------+
| cookbook   |
+------------+
```

DATABASE() – это функция, которая возвращает имя текущей базы данных. Если база данных для работы еще не выбрана, функция возвращает пустую строку:

```
mysql> SELECT DATABASE();
+------------+
| DATABASE() |
+------------+
|            |
+------------+
```

Команда STATUS (и ее синоним \s) также выводит имя текущей базы данных, а кроме того – некоторую дополнительную информацию:

```
mysql> \s
--------------
Connection id:      5589
Current database:   cookbook
Current user:       cbuser@localhost
Current pager:      stdout
Using outfile:      ''
Server version:     3.23.51-log
Protocol version:   10
Connection:     Localhost via UNIX socket
Client characterset:    latin1
Server characterset:    latin1
UNIX socket:        /tmp/mysql.sock
Uptime:         9 days 39 min 43 sec

Threads: 4  Questions: 42265  Slow queries: 0  Opens: 82  Flush tables: 1
Open tables: 52 Queries per second avg: 0.054
--------------
```

Временное обращение к таблице другой базы данных

Чтобы на время переключиться на работу с таблицей другой базы данных, можно перейти к работе с той другой базой данных, а когда операции с таблицей произведены, – вернуться к первой. Но чтобы не осуществлять переход от базы к базе, можно просто сослаться на таблицу, указав ее полное имя. Например, чтобы использовать находящуюся в отличной от текущей базы данных other_db таблицу other_tbl, укажите ее имя в виде other_db.other_tbl.

1.11. Отмена частично введенного запроса

Задача

Вы начали вводить запрос, а потом решили не выполнять его.

Решение

Отмените запрос, используя символ аннулирования строки или последовательность \c.

Обсуждение

Если вы передумали выполнять вводимый запрос, отмените его. Если запрос занимает одну строку, используйте символ аннулирования строки для того, чтобы стереть всю строку целиком. (У каждого из вас это будет какой-то свой символ, определяемый настройками терминала, например, у меня строка аннулируется при помощи клавиш Ctrl-U.) Если же запрос введен на нескольких строках, то символ аннулирования строки удалит только последнюю из них. Чтобы отменить весь запрос, введите \c и нажмите клавишу Return. Вы вернетесь к приглашению на ввод mysql>:

```
mysql> SELECT *
    -> FROM limbs
    -> ORDER BY\c
mysql>
```

Иногда кажется, что \c ничего не делает (то есть, приглашение mysql> не появляется), вы оказываетесь «в ловушке» и не можете выйти из запроса. Если команда \c не действует, это обычно означает, что вы начали набирать строку в кавычках и не ввели соответствующую закрывающую кавычку. Приглашение на ввод *mysql* поможет вам разобраться в том, что произошло. Если вместо mysql> вы видите">, это означает, что *mysql* не нашла завершающую двойную кавычку. Если же приглашение на ввод выглядит как '> – необходима завершающая одинарная кавычка. Введите кавычку для завершения строки, затем введите \c и нажмите клавишу Return. Все должно получиться!

1.12. Повторение и редактирование запросов

Задача

Только что введенный запрос содержит ошибку, которую хотелось бы исправить, не прибегая к повторному вводу всего запроса. Или же хочется повторить выполнение уже введенного предложения, не вводя его заново.

Решение

Используйте встроенный редактор запросов *mysql*.

Обсуждение

Что делать, если вы ввели длинный запрос и обнаружили, что он содержит синтаксическую ошибку? Заново с нуля вводить исправленный запрос? Не стоит – *mysql* ведет историю предложений и поддерживает возможность редактирования внутри строки. Запросы можно вызывать повторно, редактировать и вновь выполнять. Существует огромное количество команд редак-

тирования, но большинство пользователей обычно используют для редактирования лишь ограниченный набор команд.[1] Базовый набор полезных команд приведен в табл. 1.3. Обычно для повторного обращения к предыдущей строке используется клавиша <↑>, для перемещения по строке – клавиши <←> и <→>, для удаления символов – клавиша Delete или Backspace. Чтобы добавить в строку новые символы, просто переместите курсор на нужную позицию и введите символ. Когда редактирование завершено, нажмите клавишу Return, чтобы отправить запрос на выполнение (при этом необязательно, чтобы курсор находился в конце строки).

Таблица 1.3. Операции редактирования

Клавиши редактирования	Выполняемое действие
↑	Прокрутка истории запроса вверх
↓	Прокрутка истории запроса вниз
←	Перемещение влево по строке
→	Перемещение вправо по строке
Ctrl-A	Перейти в начало строки
Ctrl-E	Перейти в конец строки
Backspace	Удалить предыдущий символ
Ctrl-D	Удалить символ под курсором

Строковое редактирование полезно не только для исправления ошибок. Вы получаете возможность опробовать несколько форм одного запроса, не вводя каждый раз весь текст целиком. Кроме того, это удобно для ввода ряда похожих предложений. Например, если вы хотите выполнить серию предложений `INSERT` для создания таблицы `limbs` (см. рецепт 1.2), введите сначала первое предложение. Для выполнения каждого последующего предложения нажимайте клавишу <↑>, чтобы вернуть предыдущую строку с курсором в конце, используйте клавишу Backspace для удаления старых значений столбцов, введите новые значения и нажмите клавишу Return.

Процедура редактирования предложения, введенного в нескольких строках, чуть более сложна. Необходимо последовательно ввести и выполнить каждую строку запроса. Например, если вы ввели двухстрочный запрос, содержащий ошибку, дважды нажмите клавишу <↑>, чтобы повторно вызвать первую строку запроса. Выполните все необходимые изменения и нажмите клавишу Return. Затем опять дважды нажмите клавишу <↑>, чтобы заново вызвать вторую строку. Измените ее, нажмите клавишу Return, и запрос будет выполнен.

---

[1] Возможности редактирования строк *mysql* обеспечиваются библиотекой GNU Readline. Предоставляемые функции редактирования описаны в документации на библиотеку. Если вам необходима дополнительная информация, обратитесь к учебнику по Bash, доступному в Интернете по адресу *http://www.gnu.org/manual/*.

В Windows *mysql* допускает повторный вызов предложений только для систем, основанных на NT. При работе в Windows 98 или ME можно использовать специальную клиентскую программу *mysqlc*. Но для *mysqlc* необходим дополнительный библиотечный файл *cygwinb19.dll*. Если этот файл находится в том каталоге, где установлена программа *mysqlc* (подкаталог *bin* каталога установки MySQL), вы готовы действовать. Если библиотека расположена в подкаталоге *lib* каталога установки MySQL, скопируйте его в системный каталог Windows. Для запуска приведенной ниже команды на вашем компьютере необходимо подставить реальные пути к двум каталогам:

```
C:\> copy C:\mysql\lib\cygwinb19.dll C:\Windows\System
```

Убедившись в том, что библиотека расположена там, где *mysqlc* может ее найти, запустите *mysqlc*, и вы получите возможность редактировать строки.

Недостатком использования *mysqlc* является то, что эта программа достаточно стара (даже в дистрибутив MySQL 4.x входит *mysqlc* версии 3.22.7). Это означает, что программа не понимает предложения, появившиеся позже, такие как SOURCE.

1.13. Автоматическое завершение ввода имен баз данных и таблиц

Задача

Вы бы хотели, чтобы ввод имен баз данных и таблиц занимал меньше времени.

Решение

Используйте функциональность, предоставляемую программой *mysql*, – автоматическое завершение ввода имен.

Обсуждение

Обычно при интерактивной работе с *mysql* программа при запуске считывает перечень имен баз данных и список таблиц и столбцов текущей базы данных. Эта информация запоминается, что и обеспечивает возможности *mysql* по завершению ввода имен, которые удобно использовать для ввода предложений всего несколькими нажатиями клавиш:

- Введите часть имени базы данных, таблицы или столбца, затем нажмите клавишу Tab.
- Если введенное частичное имя уникально, *mysql* завершит его за вас. Иначе снова нажмите клавишу Tab, чтобы увидеть другие предлагаемые варианты.
- Введите дополнительные символы и нажмите клавишу Tab один раз, чтобы завершить ввод, и дважды – чтобы просмотреть список совпадений.

Для завершения ввода имен *mysql* использует имена таблиц текущей базы данных, поэтому для применения такой возможности в рамках сеанса *mysql* необходимо выбрать рабочую базу данных (в командной строке или при помощи предложения `USE`).

Автозавершение ввода позволяет сократить объем вводимых данных. Однако если вы не используете эту функцию, считывание информации для завершения имен с сервера MySQL может привести к обратным результатам: если в вашей БД большое количество таблиц, *mysql* будет запускаться медленнее. Чтобы исключить замедление запуска, нужно сообщить программе *mysql* о том, что не следует считывать эту информацию. Для этого укажите в командной строке *mysql* опцию *-A* (или *--no-auto-rehash*). Можно также добавить строку `no-auto-rehash` в группу `[mysql]` файла опций MySQL:

```
[mysql]
no-auto-rehash
```

Для того чтобы заставить *mysql* читать информацию для завершения имен несмотря на то, что программа была запущена в режиме без автозавершения ввода, введите команду `REHASH` или `\#` после приглашения на ввод `mysql>`.

1.14. Использование в запросах переменных SQL

Задача

Вы хотите сохранить значение, возвращенное запросом, чтобы использовать его в дальнейшем.

Решение

Используйте для хранения такого значений переменную SQL.

Обсуждение

В MySQL 3.23.6 вы можете присвоить переменной значение, возвращенное предложением `SELECT`, а затем ссылаться на эту переменную в рамках вашего сеанса *mysql*. Появляется возможность сохранить результат одного запроса и использовать его в последующих запросах. Присваивание значения переменной SQL в запросе `SELECT` имеет такой синтаксис: `@`*имя_переменной* `:=` *значение*, где *имя_переменной* — имя переменной, а *значение* — извлекаемое значение. Далее переменная может использоваться в запросах везде, где разрешены выражения, например в инструкции (clause) `WHERE` или в предложении (statement) `INSERT`.

Общепринятым является использование переменных SQL при выполнении ряда запросов к нескольким таблицам, имеющим общее значение ключа. Пусть, например, у вас есть таблица `customers` (клиенты) со столбцом `cust_id`, идентифицирующим каждого клиента, и таблица `orders` (заказы), также содержащая столбец `cust_id` для сопоставления каждого заказа определенному клиенту. Если вы знаете имя клиента и хотите удалить запись о нем, а также

обо всех его заказах, необходимо определить для такого клиента значение cust_id, а затем удалить из таблиц customers и orders записи, соответствующие данному номеру. Можно сначала сохранить значение идентификатора в переменной, а затем сослаться на эту переменную в предложениях DELETE:[1]

```
mysql> SELECT @id := cust_id FROM customers WHERE cust_id='customer name';
mysql> DELETE FROM customers WHERE cust_id = @id;
mysql> DELETE FROM orders WHERE cust_id = @id;
```

В первом запросе переменной присваивается значение столбца, но возможно и присвоение значений произвольных выражений. Следующее предложение вычисляет наибольшую сумму столбцов arms и legs таблицы limbs и присваивает его переменной @max_limbs:

```
mysql> SELECT @max_limbs := MAX(arms+legs) FROM limbs;
```

Кроме того, можно использовать переменную для хранения результата выполнения LAST_INSERT_ID() после создания новой записи в таблице, содержащей столбец AUTO_INCREMENT:

```
mysql> SELECT @last_id := LAST_INSERT_ID();
```

LAST_INSERT_ID() возвращает новое значение AUTO_INCREMENT. Сохранив его в переменной, вы сможете ссылаться на это значение сколько угодно раз в последующих предложениях, даже если какие-то предложения создают собственные значения AUTO_INCREMENT, изменяя тем самым значение, возвращаемое LAST_INSERT_ID(). Подробно об этом будет рассказано в главе 11.

Переменные SQL хранят одиночные значения. Если вы присваиваете переменной значение, используя предложение, возвращающее несколько строк, будет взято значение последней строки:

```
mysql> SELECT @name := thing FROM limbs WHERE legs = 0;
+----------------+
| @name := thing |
+----------------+
| squid          |
| octopus        |
| fish           |
| phonograph     |
+----------------+
mysql> SELECT @name;
+------------+
| @name      |
+------------+
| phonograph |
+------------+
```

[1] В MySQL 4 для выполнения подобных задач можно в одном запросе использовать предложения DELETE, обращенные к нескольким таблицам сразу. Примеры приведены в главе 12.

Если предложение не возвращает строк, присваивание не выполняется, и переменная сохраняет прежнее значение. Если переменная ранее не использовалась, ее значение — NULL:

```
mysql> SELECT @name2 := thing FROM limbs WHERE legs < 0;
Empty set (0.00 sec)
mysql> SELECT @name2;
+--------+
| @name2 |
+--------+
| NULL   |
+--------+
```

Чтобы явно присвоить переменной какое-то конкретное значение, используйте предложение SET. В синтаксисе этого предложения для присваивания вместо := применяется =:

```
mysql> SET @sum = 4 + 7;
mysql> SELECT @sum;
+------+
| @sum |
+------+
|   11 |
+------+
```

Присвоенное переменной значение сохраняется до тех пор, пока не будет присвоено новое значение, или же до конца сеанса *mysql* (в зависимости от того, какое из двух событий произойдет раньше).

Имена переменных чувствительны к регистру:

```
mysql> SET @x = 1; SELECT @x, @X;
+------+------+
| @x   | @X   |
+------+------+
|    1 | NULL |
+------+------+
```

Переменные SQL могут использоваться только там, где разрешены выражения, но не вместо констант или литералов. И хотя нередко пытаются использовать переменные для имен таблиц, из этого ничего не получается. Например, вы можете попробовать сформировать имя временной таблицы с помощью переменной, но результатом будет лишь сообщение об ошибке:

```
mysql> SET @tbl_name = CONCAT('tbl_',FLOOR(RAND()*1000000));
mysql> CREATE TABLE @tbl_name (int_col INT);
ERROR 1064 at line 2: You have an error in your SQL syntax near '@tbl_name
(int_col INT)' at line 1
```

Переменные SQL являются специальным расширением MySQL и не будут работать с другими процессорами баз данных.

1.15. Чтение запросов из файла

Задача

Вы хотите, чтобы программа *mysql* считывала запросы, хранящиеся в файле (при этом отпадает необходимость вводить их вручную).

Решение

Перенаправьте ввод *mysql* или используйте команду `SOURCE`.

Обсуждение

По умолчанию программа *mysql* интерактивно считывает ввод с терминала, но вы можете «питать» ее запросами в режиме пакетной обработки, используя другие источники ввода, такие как файлы, программы или аргументы команд. Источником ввода запроса также могут быть копирование и вставка. Данный раздел посвящен чтению запросов из файла. Несколько следующих рецептов расскажут о том, как получить ввод из других источников.

Чтобы создать SQL-сценарий, переводящий *mysql* в режим пакетной обработки, поместите предложения в текстовый файл, вызовите *mysql* и перенаправьте ее ввод так, чтобы он поступал из данного файла:

```
% mysql cookbook < имя_файла
```

Предложения, считываемые из файла ввода, заменяют те, что вы обычно вводите вручную, поэтому они также должны заканчиваться точкой с запятой (или `\g`). Одним из различий интерактивного и пакетного режимов является формат вывода по умолчанию. В интерактивном режиме по умолчанию используется табличный (блочный) формат. В пакетном режиме значения столбцов по умолчанию разделяются символами табуляции. Вы можете задать удобный формат вывода, используя соответствующие опции командной строки. О выборе формата вывода будет подробно рассказано далее в этой же главе (рецепт 1.21).

Работа в пакетном режиме удобна тогда, когда нужно выполнить некоторое множество предложений в различных ситуациях (нет необходимости каждый раз вводить их вручную). Например, пакетный режим упрощает настройку заданий *cron*, работающих без вмешательства пользователя. SQL-сценарии также весьма полезны для передачи запросов другим пользователям. Многие из примеров данной книги могут быть запущены при помощи файлов сценариев, входящих в дистрибутив `recipes` (см. приложение А). Вы можете в пакетном режиме передать эти файлы *mysql*, чтобы избежать ручного ввода. Если в примере приводится предложение `CREATE TABLE`, описывающее, как выглядит какая-то таблица, в дистрибутиве будет присутствовать командный файл, который может быть использован для создания таблицы (и в некоторых случаях – для загрузки в нее данных). Например, ранее в главе рассматривались предложения создания и заполнения данными таблицы `limbs`. В дистрибутив `recipes` входит файл *limbs.sql*, который содержит предложения, делающие то же самое. Файл выглядит так:

```
DROP TABLE IF EXISTS limbs;
CREATE TABLE limbs
(
    thing   VARCHAR(20),    # what the thing is
    legs    INT,            # number of legs it has
    arms    INT             # number of arms it has
);

INSERT INTO limbs (thing,legs,arms) VALUES('human',2,2);
INSERT INTO limbs (thing,legs,arms) VALUES('insect',6,0);
INSERT INTO limbs (thing,legs,arms) VALUES('squid',0,10);
INSERT INTO limbs (thing,legs,arms) VALUES('octopus',0,8);
INSERT INTO limbs (thing,legs,arms) VALUES('fish',0,0);
INSERT INTO limbs (thing,legs,arms) VALUES('centipede',100,0);
INSERT INTO limbs (thing,legs,arms) VALUES('table',4,0);
INSERT INTO limbs (thing,legs,arms) VALUES('armchair',4,2);
INSERT INTO limbs (thing,legs,arms) VALUES('phonograph',0,1);
INSERT INTO limbs (thing,legs,arms) VALUES('tripod',3,0);
INSERT INTO limbs (thing,legs,arms) VALUES('Peg Leg Pete',1,2);
INSERT INTO limbs (thing,legs,arms) VALUES('space alien',NULL,NULL);
```

Чтобы выполнить предложения данного файла SQL-сценария в пакетном режиме, перейдите в каталог *tables* дистрибутива `recipes`, в котором расположены сценарии создания таблицы, затем выполните команду:

```
% mysql cookbook < limbs.sql
```

Обратите внимание на то, что сценарий содержит команду удаления ранее существующей таблицы (если таковая была) перед созданием новой и заполнением ее данными. Благодаря этому вы сможете экспериментировать с таблицей, не беспокоясь об изменении ее содержимого, так как вы в любой момент сможете вернуть таблицу в ее первоначальное состояние, еще раз запустив сценарий.

Только что приведенная команда показывает, как задавать файл ввода для *mysql* в командной строке. В MySQL 3.23.9 вы можете в рамках сеанса *mysql* считывать файл с предложениями SQL, используя команду `SOURCE` *имя_файла* (или ее синоним `\.` *имя_файла*). Предположим, что файл сценария SQL *test.sql* содержит такие предложения:

```
SELECT NOW();
SELECT COUNT(*) FROM limbs;
```

Вы можете выполнить этот файл из *mysql* так:

```
mysql> SOURCE test.sql;
+---------------------+
| NOW()               |
+---------------------+
| 2001-07-04 10:35:08 |
+---------------------+
1 row in set (0.00 sec)
```

```
+----------+
| COUNT(*) |
+----------+
|       12 |
+----------+
1 row in set (0.01 sec)
```

Сценарии SQL могут содержать команды `SOURCE` или `\.` для включения других сценариев. В связи с этим существует опасность возникновения зацикливания источников. Обычно следует избегать подобных петель, но если вы непослушны и вам хочется специально создать такую ситуацию, чтобы посмотреть, какую глубину вложения файлов ввода поддерживает *mysql*, покажу, как это делается. Сначала вручную выполните два предложения, чтобы создать таблицу `counter`, которая будет отслеживать глубину исходного файла, и установите начальный уровень вложения в ноль:

```
mysql> CREATE TABLE counter (depth INT);
mysql> INSERT INTO counter SET depth = 0;
```

Затем создайте файл сценария *loop.sql*, содержащий следующие строки (проследите, чтобы каждая строка заканчивалась точкой с запятой):

```
UPDATE counter SET depth = depth + 1;
SELECT depth FROM counter;
SOURCE loop.sql;
```

Наконец, вызовите *mysql* и выполните команду `SOURCE` для чтения файла сценария:

```
% mysql cookbook
mysql> SOURCE loop.sql;
```

Два первых предложения *loop.sql* увеличивают счетчик вложенности и отображают текущее значение глубины – `depth`. В третьем предложении файл *loop.sql* использует в качестве источника самого себя, тем самым создавая петлю ввода. Вы увидите, как вывод заполняет экран, при этом после каждого прохождения петли счетчик будет увеличиваться на единицу. В конце концов *mysql* исчерпает файловые дескрипторы и завершится с ошибкой:

```
ERROR:
Failed to open file 'loop.sql', error: 24
```

Что такое ошибка 24? Получим ответ, применив команду MySQL *perror* (print error – печатать ошибки):

```
% perror 24
Error code 24:  Too many open files
```

1.16. Чтение запросов из других программ

Задача

Вы хотите подать вывод другой программы на вход *mysql*.

Решение

Используйте канал (pipe).

Обсуждение

В предыдущем разделе для демонстрации того, как *mysql* может читать предложения SQL из файла, были использованы такие команды:

```
% mysql cookbook < limbs.sql
```

Программа *mysql* также может читать из канала, получая на вход выходные данные других программ. Для наглядности приведем эквивалент предыдущей команды:

```
% cat limbs.sql | mysql cookbook
```

Прежде чем вы поспешите вручить мне то, что я назвал бы «премией за самое бесполезное применение *cat*»,[1] позвольте заметить, что вы можете заменить *cat* любой другой командой. Идея в том, что *любую* команду, порождающую вывод, состоящий из SQL-предложений, заканчивающихся точкой с запятой, можно использовать как источник ввода для *mysql*. Такая возможность весьма полезна. Например, команда *mysqldump* применяется для создания резервных копий баз данных. Она записывает резервную копию в виде множества предложений SQL, пересоздающих базу данных, и для обработки вывода *mysqldump* вы можете подать его на вход *mysql*. Таким образом, можно использовать комбинацию *mysqldump* и *mysql* для копирования базы данных через сеть на другой сервер MySQL:

```
% mysqldump cookbook | mysql -h some.other.host.com cookbook
```

Программно генерируемый SQL также может быть полезен для заполнения таблицы тестовыми данными, если вы не хотите писать предложения `INSERT` вручную. Напишите простую программку, которая формирует предложения и отправляет их вывод в *mysql* через канал:

```
% generate-test-data | mysql cookbook
```

См. также

О *mysqldump* будет рассказано в главе 10.

1.17. Ввод запросов в командной строке

Задача

Вы хотите ввести запрос непосредственно в командной строке, чтобы он был выполнен программой *mysql*.

---

[1] При работе в Windows аналогом была бы премия за самое бесполезное применение *type*.

Решение

Программа *mysql* умеет читать запросы из списка аргументов. Используйте опцию *-e* (или *--execute*) для ввода запроса в командной строке.

Обсуждение

Например, чтобы узнать, сколько записей содержится в таблице `limbs`, выполните следующую команду:

```
% mysql -e "SELECT COUNT(*) FROM limbs" cookbook
+----------+
| COUNT(*) |
+----------+
|       12 |
+----------+
```

Чтобы задать при помощи опции *-e* несколько запросов, используйте в качестве разделителя точку с запятой:

```
% mysql -e "SELECT COUNT(*) FROM limbs;SELECT NOW()" cookbook
+----------+
| COUNT(*) |
+----------+
|       12 |
+----------+
+---------------------+
| NOW()               |
+---------------------+
| 2001-07-04 10:42:22 |
+---------------------+
```

См. также

По умолчанию результаты запросов, введенных при помощи *-e*, выводятся в табличной форме, если вывод передается на терминал, и в виде значений, разделенных знаками табуляции, в остальных случаях. О других возможных стилях вывода рассказано в рецепте 1.21.

1.18. Использование копирования и вставки для формирования ввода mysql

Задача

Вы хотите упростить работу с *mysql*, используя преимущества вашего графического интерфейса пользователя (GUI).

Решение

Используйте для формирования запросов, подаваемых на вход *mysql*, операции копирования (copy) и вставки (paste). С помощью возможностей графи-

ческого интерфейса вы расширите терминальный интерфейс, предлагаемый программой *mysql*.

Обсуждение

При работе в многооконной среде, когда вы можете одновременно запустить несколько программ и передавать информацию из одной в другую, удобно использовать копирование и вставку. Если в одном окне открыт документ, содержащий тексты запросов, просто скопируйте их и вставьте в окно, в котором вы работает с *mysql*. Это быстрее, чем вводить запросы с клавиатуры. Удобно хранить в отдельном окне часто выполняемые запросы – тогда они всегда будут под рукой и легко доступны.

1.19. Борьба с исчезновением с экрана вывода запроса

Задача

Начало вывода запроса исчезает с экрана раньше, чем вы успеваете его прочитать.

Решение

Укажите программе *mysql*, что необходимо отобразить вывод постранично, или запустите *mysql* в окне, допускающем прокрутку.

Обсуждение

Если запрос выводит несколько строк выходных данных, обычно они сразу же пропадают из верхней части экрана. Чтобы избежать этого, сообщите программе *mysql*, что следует показывать вывод постранично, установив опцию *--pager*.[1] Конструкция *--pager=программа* показывает, какую именно программу нужно использовать для разбиения вывода на страницы:

```
% mysql --pager=/usr/bin/less
```

Если ввести просто *--pager*, то *mysql* будет использовать вашу программу постраничного вывода по умолчанию, заданную в переменной окружения PAGER:

```
% mysql --pager
```

Если переменная окружения PAGER не установлена, вы должны или определить ее, или применять первую форму команды с явным указанием программы. Для определения переменной PAGER следуйте инструкции рецепта 1.8 по установке переменных окружения.

В рамках сеанса *mysql* вы можете включать и отключать разбиение на страницы при помощи \P и \n. Команда \P без аргумента разрешает разбиение на страницы посредством программы, указанной в переменной PAGER. Команда \P

[1] Опция --pager недоступна в Windows.

с аргументом разрешает разбиение на страницы посредством программы, название которой указано как аргумент:

```
mysql> \P
PAGER set to /bin/more
mysql> \P /usr/bin/less
PAGER set to /usr/bin/less
mysql> \n
PAGER set to stdout
```

Постраничный вывод впервые появился в MySQL 3.23.28.

В качестве еще одного способа обработки больших результирующих множеств можно предложить использование терминальной программы, позволяющей просматривать предыдущий вывод путем обратной прокрутки. Такие программы, как *xterm* для X Window System, Terminal для Mac OS X, MacSSH или BetterTelnet для Mac OS, Telnet для Windows, позволяют указать количество строк вывода, сохраняемых в буфере обратной прокрутки. При работе в Windows NT, 2000 или XP вы можете создать окно DOS, допускающее обратную прокрутку, выполнив следующие операции:

1. Создайте ярлык для запуска окна «Командная строка» («Console») там, где бы вы хотели его видеть (например, на рабочем столе).
2. Выделите ярлык и нажмите правую кнопку мыши, в появившемся меню выберите Свойства (Properties).
3. В открывшемся окне перейдите на вкладку Расположение (Layout).
4. Установите высоту буфера экрана в то количество строк, которое вы хотите хранить, и нажмите кнопку OK.

Теперь при запуске ярлыка откроется окно DOS, в котором можно при помощи полосы прокрутки просмотреть вывод команд, сформированный в данном окне.

1.20. Перенаправление вывода запроса в файл или программу

Задача

Вы хотите отправить вывод программы *mysql* не на экран.

Решение

Перенаправьте (redirect) вывод *mysql* или используйте канал.

Обсуждение

Формат вывода *mysql* по умолчанию зависит от того, в каком режиме запущена программа: интерактивно или нет. При интерактивном использовании *mysql* обычно отправляет вывод на терминал и записывает результат запроса в табличном формате:

```
mysql> SELECT * FROM limbs;
+--------------+------+------+
| thing        | legs | arms |
+--------------+------+------+
| human        |    2 |    2 |
| insect       |    6 |    0 |
| squid        |    0 |   10 |
| octopus      |    0 |    8 |
| fish         |    0 |    0 |
| centipede    |  100 |    0 |
| table        |    4 |    0 |
| armchair     |    4 |    2 |
| phonograph   |    0 |    1 |
| tripod       |    3 |    0 |
| Peg Leg Pete |    1 |    2 |
| space alien  | NULL | NULL |
+--------------+------+------+
12 rows in set (0.00 sec)
```

В неинтерактивном режиме (то есть когда или ввод, или вывод перенаправлены), *mysql* отображает вывод, используя в качестве разделителей знаки табуляции:

```
% echo "SELECT * FROM limbs" | mysql cookbook
thing   legs    arms
human   2       2
insect  6       0
squid   0       10
octopus 0       8
fish    0       0
centipede       100     0
table   4       0
armchair        4       2
phonograph      0       1
tripod  3       0
Peg Leg Pete    1       2
space alien     NULL    NULL
```

Однако в любом из режимов вы можете определить свой формат вывода *mysql*, указав соответствующие опции командной строки. В данном разделе будет рассказано, как отправить вывод *mysql* не на терминал. Несколько последующих рецептов описывают различные форматы вывода *mysql* и показывают, как явно выбрать необходимый формат, если предлагаемый по умолчанию формат вам не подходит.

Чтобы сохранить вывод *mysql* в файле, используйте стандартное для вашей оболочки перенаправление вывода:

```
% mysql cookbook > файл_вывода
```

Но если задав перенаправление вывода, вы попытаетесь запустить *mysql* интерактивно, то не сможете увидеть, что вводите. Поэтому ввод в таких случаях обычно также получают из файла (или другой программы):

```
% mysql cookbook < файл_ввода > файл_вывода
```

Вы можете направить вывод запроса в другую программу. Например, если вы хотите отправить кому-то по почте результаты запроса, выполните такую команду:

```
% mysql cookbook < файл_ввода | mail paul
```

Обратите внимание, что поскольку в данном случае *mysql* работает неинтерактивно, вывод будет состоять из элементов, разделенных табуляциями, и читать его получателю будет гораздо труднее, чем данные в табличной форме. В рецепте 1.21 показано, как решить эту проблему.

1.21. Выбор формата вывода: таблица или элементы, разделенные табуляцией

Задача

Программа *mysql* формирует вывод в табличной форме, когда хотелось бы видеть столбцы, разделенные знаками табуляции, или наоборот.

Решение

Явно укажите желаемый формат вывода, задав соответствующую опцию командной строки.

Обсуждение

Когда вы применяете программу *mysql* неинтерактивно (например, для чтения запросов из файла или отправки их результатов в канал), она по умолчанию отображает вывод, используя в качестве разделителей знаки табуляции. В некоторых случаях предпочтительнее табличная форма вывода. Например, если вы хотите распечатать результаты запроса или отправить их по почте, формат по умолчанию не слишком удобен. Используйте опцию *-t* (или *--table*) для формирования более удобной табличной формы вывода:

```
% mysql -t cookbook < файл_ввода | lpr
% mysql -t cookbook < файл_ввода | mail paul
```

Обратная операция заключается в формировании пакетного (разделенного знаками табуляции) вывода в интерактивном режиме. Для ее выполнения используйте опцию *-B* или *--batch*.

1.22. Задание произвольного разделителя для столбцов вывода

Задача

Вы хотите, чтобы программа *mysql* формировала вывод запроса, используя разделитель, отличный от знака табуляции.

Решение

Выполните пост-обработку вывода *mysql*.

Обсуждение

При работе в неинтерактивном режиме *mysql* разделяет столбцы вывода знаками табуляции, и не существует опции для указания разделителя вывода. Но в некоторых ситуациях удобнее разделять вывод другими символами. Предположим, что вам необходимо создать файл вывода, который будет использоваться другой программой, воспринимающей как разделители символы двоеточия (:). В UNIX вы можете преобразовать знаки табуляции в произвольные разделители с помощью таких команд, как *tr* и *sed*. Например, чтобы заменить знаки табуляции на двоеточия, используйте любую из приведенных ниже команд (*TAB* обозначает место ввода знака табуляции):[1]

```
% mysql cookbook < файл_ввода | sed -e "s/TAB/:/g" > файл_вывода
% mysql cookbook < файл_ввода | tr "TAB" ":" > файл_вывода
% mysql cookbook < файл_ввода | tr "\011" ":" > файл_вывода
```

Команда *sed* мощнее, чем *tr*: она понимает регулярные выражения и разрешает несколько замен. Такая возможность удобна, когда необходимо сформировать что-то типа списка значений, разделенных запятыми (comma-separated values – CSV), для чего требуются три замены:

- Продублируйте все кавычки, встречающиеся в данных, дабы они не были восприняты как разделители столбцов при использовании результирующего CSV-файла.
- Замените знаки табуляции на запятые.
- Заключите значения столбцов в кавычки.

Команда *sed* позволяет выполнить все три замены в одной команде:

```
% mysql cookbook < файл_ввода \
    | sed -e 's/"/""/g' -e 's/TAB/","/g' -e 's/^/"/' -e 's/$/"/' > файл_вывода
```

Выглядит, мягко говоря, достаточно загадочно. Того же результата можно достичь, используя другой, более удобочитаемый язык. Приведем короткий сценарий Perl, который делает то же, что и команда *sed* (преобразует вывод, разделенный знаками табуляции, в CSV-список), и содержит комментарии, поясняющие, как он работает:

```
#! /usr/bin/perl -w
while (<>)              # прочитать следующую строку ввода
{
    s/"/""/g;           # продублировать все кавычки в значениях столбцов
```

[1] Некоторые версии *tr* могут иметь другой синтаксис; обратитесь к документации по вашей системе. А в некоторых оболочках знак табуляции используется в специальных целях, например для завершения имен файлов. В таких оболочках вводите в команде символ табуляции, предварив его нажатием комбинации клавиш Ctrl-V.

```
    s/\t/"","/g;        # поместить `","' между значениями столбцов
    s/^/"/;             # добавить `"' перед первым значением
    s/$/"/;             # добавить `"' после последнего значения
    print;              # вывести результат
}
exit (0);
```

Если назвать сценарий *csv.pl*, можно использовать его так:

```
% mysql cookbook < файл_ввода | csv.pl > файл_вывода
```

Если вы запускаете команды в версии Windows, которая не умеет сопоставлять файлы *.pl* интерпретатору Perl, может потребоваться явный вызов Perl:

```
C:\> mysql cookbook < файл_ввода | perl csv.pl > файл_вывода
```

Если вам необходимо межплатформенное решение, вероятно, удобнее использовать Perl, так как он работает и в UNIX, и в Windows, в то время как *tr* и *sed* обычно недоступны в Windows.

См. также

Еще более удачным способом формирования CSV-вывода является использование модуля Perl *Text::CSV_XS*, специально предназначенного для этого. Модель будет обсуждаться в главе 10, где он применяется для создания более универсального переформатировщика файлов.

1.23. Формирование HTML-вывода

Задача

Вы хотели бы преобразовать результат запроса к формату HTML.

Решение

За вас может поработать *mysql*.

Обсуждение

Если указать опцию *-H* (или *--html*), *mysql* сгенерирует результирующее множество в виде таблиц HTML. Это быстрый способ получения пробного вывода для включения его в веб-страницу, показывающую, как выглядит результат запроса.[1] Приведем пример, иллюстрирующий разницу между табличным форматом вывода и выводом в формате HTML-таблиц (для повышения читабельности HTML-вывода в него было добавлено несколько разделителей строк):

[1] В данном случае я говорю о создании статических HTML-страниц. Если вы пишете сценарий, который формирует веб-страницы динамически, существуют более подходящие способы вывода результата запроса в виде HTML. Подробная информация о написании веб-сценариев приведена в главе 16.

```
% mysql -e "SELECT * FROM limbs WHERE legs=0" cookbook
+------------+------+------+
| thing      | legs | arms |
+------------+------+------+
| squid      |    0 |   10 |
| octopus    |    0 |    8 |
| fish       |    0 |    0 |
| phonograph |    0 |    1 |
+------------+------+------+
% mysql -H -e "SELECT * FROM limbs WHERE legs=0" cookbook
<TABLE BORDER=1>
<TR><TH>thing</TH><TH>legs</TH><TH>arms</TH></TR>
<TR><TD>squid</TD><TD>0</TD><TD>10</TD></TR>
<TR><TD>octopus</TD><TD>0</TD><TD>8</TD></TR>
<TR><TD>fish</TD><TD>0</TD><TD>0</TD></TR>
<TR><TD>phonograph</TD><TD>0</TD><TD>1</TD></TR>
</TABLE>
```

Первая строка таблицы содержит заголовки столбцов. Если вам не нужна строка заголовков, обратитесь к рецепту 1.25.

Опции *-H* и *--html* формируют вывод только для запросов, которые порождают результирующее множество. Для таких запросов, как `INSERT` или `UPDATE`, вывод не записывается.

Опции *-H* и *--html* могут использоваться, начиная с MySQL 3.22.26. (Фактически они были введены в более ранней версии, но вывод формировался не совсем корректно.)

1.24. Формирование XML-вывода

Задача

Вы хотели бы преобразовать результат запроса к формату XML.

Решение

За вас может поработать *mysql*.

Обсуждение

Если указать опцию *-X* (или *--xml*), *mysql* создаст из результата запроса XML-документ. Рассмотрим пример, иллюстрирующий отличие табличной формы вывода результата запроса от XML-документа, созданного из результата того же запроса:

```
% mysql -e "SELECT * FROM limbs WHERE legs=0" cookbook
+------------+------+------+
| thing      | legs | arms |
+------------+------+------+
| squid      |    0 |   10 |
| octopus    |    0 |    8 |
| fish       |    0 |    0 |
```

```
| phonograph |    0 |    1 |
+------------+------+------+
% mysql -X -e "SELECT * FROM limbs WHERE legs=0" cookbook
<?xml version="1.0"?>

<resultset statement="SELECT * FROM limbs WHERE legs=0">
  <row>
    <thing>squid</thing>
    <legs>0</legs>
    <arms>10</arms>
  </row>

  <row>
    <thing>octopus</thing>
    <legs>0</legs>
    <arms>8</arms>
  </row>

  <row>
    <thing>fish</thing>
    <legs>0</legs>
    <arms>0</arms>
  </row>

  <row>
    <thing>phonograph</thing>
    <legs>0</legs>
    <arms>1</arms>
  </row>
</resultset>
```

Опции *-X* и *--xml* могут применяться, начиная с версии MySQL 4.0. Если вы работаете с более старой версией MySQL, то можете написать собственный генератор XML (см. рецепт 10.41).

1.25. Исключение заголовков столбцов из вывода запроса

Задача

Вы не хотите, чтобы вывод запроса содержал заголовки столбцов.

Решение

Используйте соответствующую опцию командной строки для отключения отображения заголовков столбцов. Обычно это опция *-N* или *--skip-column-names*, но вы также можете использовать *-ss*.

Обсуждение

Формат со знаками табуляции в качестве разделителей удобен для формирования файлов данных, которые могут импортироваться в другие программы.

Но в первой строке вывода каждого запроса по умолчанию приводится перечень заголовков столбцов, что может быть уже не всегда удобно. Предположим, у вас есть программа *summarize*, которая генерирует различные количественные показатели распределения для столбца чисел. Если вы хотите, чтобы вывод *mysql* использовался такой программой, вам необходимо, чтобы строки заголовков не было, иначе результаты будут испорчены. То есть если выполнить следующую команду, вывод будет некорректным, так как *summarize* примет в расчет заголовок столбца:

```
% mysql -e "SELECT arms FROM limbs" cookbook | summarize
```

Чтобы создать вывод, который содержит только значения данных, устраните строку заголовков столбцов при помощи опции *-N* или *--skip-column-names*:

```
% mysql -N -e "SELECT arms FROM limbs" cookbook | summarize
```

Опции *-N* и *--skip-column-names* появились в MySQL 3.22.20. При работе с более старыми версиями вы можете добиться того же результата, дважды указав опцию «молчания» (silent) *-s* или *--silent*:

```
% mysql -ss -e "SELECT arms FROM limbs" cookbook | summarize
```

В UNIX альтернативой является использование утилиты *tail* для отбрасывания первой строки:

```
% mysql -e "SELECT arms FROM limbs" cookbook | tail +2 | summarize
```

1.26. Нумерация строк вывода запроса

Задача

Вы хотите пронумеровать строки результата запроса.

Решение

Необходима пост-обработка вывода *mysql* или же использование переменной SQL.

Обсуждение

Если вы хотите пронумеровать строки вывода при работе в UNIX, полезной может оказаться опция *-N* в сочетании с *cat -n*:

```
% mysql -N -e "SELECT thing, arms FROM limbs" cookbook | cat -n
     1  human   2
     2  insect  0
     3  squid   10
     4  octopus 8
     5  fish    0
     6  centipede       0
     7  table   0
     8  armchair        2
```

```
 9  phonograph      1
10  tripod  0
11  Peg Leg Pete    2
12  NULL
```

Также можно воспользоваться переменной SQL. Выражения, содержащие переменные, вычисляются для каждой строки результата запроса – это свойство можно использовать для формирования столбца номеров строк вывода запроса:

```
mysql> SET @n = 0;
mysql> SELECT @n := @n+1 AS rownum, thing, arms, legs FROM limbs;
+--------+--------------+------+------+
| rownum | thing        | arms | legs |
+--------+--------------+------+------+
|      1 | human        |    2 |    2 |
|      2 | insect       |    0 |    6 |
|      3 | squid        |   10 |    0 |
|      4 | octopus      |    8 |    0 |
|      5 | fish         |    0 |    0 |
|      6 | centipede    |    0 |  100 |
|      7 | table        |    0 |    4 |
|      8 | armchair     |    2 |    4 |
|      9 | phonograph   |    1 |    0 |
|     10 | tripod       |    0 |    3 |
|     11 | Peg Leg Pete |    2 |    1 |
|     12 | space alien  | NULL | NULL |
+--------+--------------+------+------+
```

1.27. Улучшение читаемости длинных строк

Задача

Строки вывода запроса слишком длинные. Они переносятся и создают беспорядок на экране.

Решение

Используйте вертикальный формат вывода.

Обсуждение

Некоторые запросы генерируют строки вывода, занимающие по несколько строк на терминале, что усложняет чтение результатов запроса. Посмотрим, как могут выглядеть на экране чрезмерно длинные строки вывода запроса:[1]

```
mysql> SHOW FULL COLUMNS FROM limbs;
+-------+-------------+------+-----+---------+-------+------------------------
```

[1] Если вы работаете с более старой, чем MySQL 3.23.32, версией, уберите ключевое слово `FULL` из предложения `SHOW COLUMNS`.

```
--------+
| Field | Type        | Null | Key | Default | Extra | Privileges
        |
+-------+-------------+------+-----+---------+-------+-----------------------
--------+
| thing | varchar(20) | YES  |     | NULL    |       | select,insert,update,ref
erences |
| legs  | int(11)     | YES  |     | NULL    |       | select,insert,update,ref
erences |
| arms  | int(11)     | YES  |     | NULL    |       | select,insert,update,ref
erences |
+-------+-------------+------+-----+---------+-------+-----------------------
--------+
```

Альтернативой является формирование «вертикального» вывода: значение каждого столбца выводится на отдельной строке. Для этого запрос следует заканчивать символами \G вместо точки с запятой или \g. Покажем, как будет выглядеть результат предыдущего запроса при отображении в вертикальном формате:

```
mysql> SHOW FULL COLUMNS FROM limbs\G
*************************** 1. row ***************************
     Field: thing
      Type: varchar(20)
      Null: YES
       Key:
   Default: NULL
     Extra:
Privileges: select,insert,update,references
*************************** 2. row ***************************
     Field: legs
      Type: int(11)
      Null: YES
       Key:
   Default: NULL
     Extra:
Privileges: select,insert,update,references
*************************** 3. row ***************************
     Field: arms
      Type: int(11)
      Null: YES
       Key:
   Default: NULL
     Extra:
Privileges: select,insert,update,references
```

Чтобы установить вертикальный вывод в командной строке, используйте при вызове *mysql* опцию *-E* (или *--vertical*). Такая установка будет применена ко всем запросам, выполняемым в течение сеанса, что может быть удобно при использовании *mysql* для выполнения сценария. (Если вы используете в качестве завершающего символа предложений в файле сценария SQL тра-

диционную точку с запятой, то можете задать обычный или вертикальный формат вывода в командной строке путем выборочного использования *-E*.)

1.28. Управление уровнем подробности mysql

Задача

Вы хотите, чтобы *mysql* порождала больше вывода. Или меньше.

Решение

Используйте опции *-v* или *-s* для формирования более или менее подробного вывода.

Обсуждение

Когда вы запускаете *mysql* неинтерактивно, меняется не только формат вывода по умолчанию, сам вывод становится более кратким. Например, *mysql* не выводит счетчики строк и не указывает время выполнения запроса. Чтобы заставить программу *mysql* быть более многословной, используйте опцию *-v* или *--verbose*. Для еще большей подробности опции можно указать несколько раз. Попробуйте выполнить приведенные ниже команды, чтобы посмотреть, чем отличаются их вывод:

```
% echo "SELECT NOW()" | mysql
% echo "SELECT NOW()" | mysql -v
% echo "SELECT NOW()" | mysql -vv
% echo "SELECT NOW()" | mysql -vvv
```

Противоположностью *-v* и *--verbose* являются *-s* и *--silent*. Эти опции также могут применяться несколько раз для усиления эффекта.

1.29. Протоколирование интерактивных сеансов mysql

Задача

Вы хотите записать то, что было сделано в течение сеанса *mysql*.

Решение

Создайте tee-файл.

Обсуждение

Если вы храните журнал интерактивного сеанса MySQL, то сможете вернуться к нему впоследствии, чтобы посмотреть, что и как делали. В UNIX используйте программу *script* для сохранения журнала терминального сеанса. Она работает для любых команд, а значит, и для интерактивных сеансов *mysql*. Но *script* добавляет к каждой строке символ возврата каретки и

включает в журнал все возвраты курсора и исправления, которые вы сделали при вводе с клавиатуры. Для протоколирования интерактивной сеанса *mysql* без захламления файла журнала ненужными данными следует (это будет работать и в UNIX, и в Windows) запустить *mysql* с опцией *--tee*, указывающей имя файла, в котором будет записываться сеанс:[1]

```
% mysql --tee=tmp.out cookbook
```

Для управления протоколированием сеанса внутри *mysql* используйте опции `\T` и `\t` для включения и выключения протоколирования. Такая возможность удобна, если вы хотите записать только части сеанса:

```
mysql> \T tmp.out
Logging to file 'tmp.out'
mysql> \t
Outfile disabled.
```

Файл протокола содержит введенные запросы, а также их выводы, так что это удобный способ их полного учета. Подобное протоколирование удобно для распечатывания или отправки сеанса по почте, а также для фиксации вывода запроса для включения в качестве примера в какой-то документ. Также удобно тестировать запросы на корректность синтаксиса перед помещением их в файл сценария – затем можно редактированием создать сценарий из tee-файла, удалив все, кроме тех запросов, которые следует сохранить.

mysql не перезаписывает tee-файл, а добавляет вывод сеанса в его конец. Если вы хотите, чтобы существующий файл включал в себя содержание только одного сеанса, удалите этот файл перед запуском *mysql*.

Возможность создавать tee-файлы появилась в MySQL 3.23.28.

1.30. Создание сценариев mysql из ранее выполненных запросов

Задача

Вы хотите повторно использовать запросы, которые выполнялись в предыдущих сеансах *mysql*.

Решение

Используйте tee-файл предыдущего сеанса или обратитесь к файлу истории ввода *mysql*.

Обсуждение

Одним из возможных способов создания командного файла является ввод запросов в файл вручную с нуля в текстовом редакторе в надежде, что при

[1] Название «tee» происходит от UNIX-утилиты *tee*. Те, кого интересует дополнительная информация, могут выполнить команду: `% man tee`.

вводе не сделано ошибок. Но часто бывает проще использовать запросы, корректность которых уже проверена. Как это сделать? Для начала «вручную» протестируйте запросы, запустив *mysql* в интерактивном режиме и убедившись, что они работают правильно. Затем извлеките запросы из записи сеанса для создания командного файла. При создании сценариев SQL особенно полезны следующие источники информации:

- Вы можете журналировать весь сеанс *mysql* или его части, используя опцию командной строки *--tee* или команду `\T` внутри *mysql*. (Подробная информация приведена в рецепте 1.29.)
- В UNIX есть еще одна возможность – использовать файл истории. Программа *mysql* хранит историю запросов в файле *.mysql_history* вашего домашнего каталога.

Журнал сеанса в tee-файле содержит не только сами запросы, но и их ввод и вывод. Эта дополнительная информация может упростить поиск интересующих вас частей сеанса. (Естественно, для создания командного файла из tee-файла необходимо будет удалить из последнего все ненужное.) Файл истории, напротив, более краток. Он содержит только выполняемые запросы, так что удалять придется меньше строк. Выберите наиболее подходящий вам источник информации.

1.31. Использование mysql в качестве калькулятора

Задача

Вам нужно быстро вычислить значение выражения.

Решение

Используйте *mysql* как калькулятор. MySQL не требует, чтобы каждое предложение `SELECT` ссылалось на какую-то таблицу, так что вы можете выбирать результаты произвольных выражений.

Обсуждение

Обычно предложения `SELECT` ссылаются на некоторую таблицу или несколько таблиц, из которых извлекаются строки. Однако в MySQL ссылка `SELECT` на таблицу необязательна, то есть вы можете использовать это предложение как калькулятор для вычисления значений выражений:

```
mysql> SELECT (17 + 23) / SQRT(64);
+----------------------+
| (17 + 23) / SQRT(64) |
+----------------------+
|           5.00000000 |
+----------------------+
```

Также с помощью SELECT удобно проверять, как производится сравнение. Например, чтобы определить, чувствительна ли к регистру операция сравнения строк, выполните следующий запрос:

```
mysql> SELECT 'ABC' = 'abc';
+---------------+
| 'ABC' = 'abc' |
+---------------+
|             1 |
+---------------+
```

Результатом сравнения является 1 (то есть «истина»; как правило, ненулевые значения соответствуют логической «истине»). Так вы узнали, что по умолчанию операция сравнения строк нечувствительна к регистру. Если выражение ложно, возвращается ноль:

```
mysql> SELECT 'ABC' = 'abcd';
+----------------+
| 'ABC' = 'abcd' |
+----------------+
|              0 |
+----------------+
```

Если определить значение выражения невозможно, результатом является NULL:

```
mysql> SELECT 1/0;
+------+
| 1/0  |
+------+
| NULL |
+------+
```

Результаты промежуточных вычислений можно хранить в переменных SQL. Следующие предложения именно так применяют переменные для вычисления общей суммы счета за гостиницу:

```
mysql> SET @daily_room_charge = 100.00;
mysql> SET @num_of_nights = 3;
mysql> SET @tax_percent = 8;
mysql> SET @total_room_charge = @daily_room_charge * @num_of_nights;
mysql> SET @tax = (@total_room_charge * @tax_percent) / 100;
mysql> SET @total = @total_room_charge + @tax;
mysql> SELECT @total;
+--------+
| @total |
+--------+
|    324 |
+--------+
```

1.32. Использование mysql в сценариях оболочки

Задача

Вы хотите вместо интерактивного запуска *mysql* вызывать ее из сценариев оболочки.

Решение

Ничто этому не мешает. Просто укажите правильные аргументы для команды.

Обсуждение

Если требуется обработать результаты запроса внутри программы, вы обычно прибегаете к программному интерфейсу MySQL, разработанному специально для используемого языка (например, в сценарии Perl вы используете интерфейс DBI). Но для простых, коротких или быстрых и приблизительных задач, вероятно, удобнее вызвать *mysql* напрямую из сценария оболочки с возможной пост-обработкой результатов другими командами. Например, в качестве простого способа написания программы, проверяющей статус сервера MySQL, можно предложить сценарий оболочки, вызывающий *mysql* (см. далее в этом же разделе). Кроме того, сценарии оболочки полезны для создания прототипов программ, которые в дальнейшем предполагается конвертировать для использования со стандартным API.

Что касается подготовки сценариев оболочек UNIX, я рекомендовал бы придерживаться оболочек семейства Борна, таких как *sh*, *bash* или *ksh*. (Оболочки *csh* и *tcsh* лучше приспособлены для интерактивного использования, чем для создания сценариев.) В данном разделе приведено несколько примеров, показывающих, как писать сценарии UNIX для */bin/sh*, а также сделаны краткие замечания о подготовке сценариев DOS. Во врезке «Использование исполняемых программ» рассказывается, как сделать сценарии исполняемыми и запустить их.

Создание сценариев оболочки для UNIX

Рассмотрим сценарий оболочки, который выводит текущее время непрерывной работы сервера MySQL. Он выполняет запрос `SHOW STATUS` для получения значения переменной состояния `Uptime`, которая содержит время работы сервера в секундах:

```
#! /bin/sh
# mysql_uptime.sh - report server uptime in seconds

mysql -B -N -e "SHOW STATUS LIKE 'Uptime'"
```

Обратите внимание на первую строку сценария, которая начинается с символов `#!`. В ней задается полное имя программы, которая должна быть вызвана для выполнения остальной части сценария, в данном случае – */bin/sh*. Чтобы использовать сценарий, создайте файл с именем *mysql_uptime.sh*, содержащий приведенные выше строки, и сделайте его исполняемым при помощи

chmod +x. Сценарий *mysql_uptime.sh* запускает программу *mysql*, используя *-e* для указания строки запроса, *-B* для формирования пакетного вывода (разделенного знаками табуляции) и *-N* для устранения из вывода строки заголовков столбцов. В результате вывод выглядит следующим образом:

```
% ./mysql_uptime.sh
Uptime  1260142
```

Команда начинается с `./`; это означает, что сценарий находится в текущем каталоге. Если перенести сценарий в каталог, указанный в переменной `PATH`, вы сможете вызывать его откуда угодно, но тогда нужно будет убрать из

Использование исполняемых программ

Прежде чем запустить написанную вами программу, необходимо сделать ее исполняемой. В UNIX используйте команду *chmod* для установки соответствующего режима доступа к файлу:

```
% chmod +x myprog
```

Для запуска программы введите ее название в командной строке:

```
% myprog
```

Однако если программа находится в текущем каталоге, ваша оболочка может не найти ее. Оболочка просматривает в поисках программ каталоги, указанные в переменной окружения `PATH`, но часто из соображений безопасности из пути поиска для оболочек UNIX специально исключается текущий каталог `./`. В этом случае для явного указания местоположения программы необходимо включить начальную часть пути – `./`:

```
% ./myprog
```

Некоторые из программ, представленных в данной книге, создаются исключительно для иллюстрации какой-то идеи и, вероятно, никогда не будут запускаться не из текущего каталога, поэтому в соответствующих примерах при вызове программ перед ними обычно указывается `./`. Если же программа предназначена для многократного использования, лучше разместить ее в каталоге, включенном в переменную `PATH`. Тогда для ее вызова не потребуется указывать начальную часть пути. Вышесказанное относится и к общим утилитам UNIX (таким как *chmod*), которые устанавливаются в стандартные системные каталоги.

В Windows исполняемость программы характеризует расширение имени файла (например, *.exe* или *.bat*), так что *chmod* не требуется. Кроме того, командный интерпретатор включает текущий каталог в путь поиска, так что вы можете вызывать расположенные там программы, не указывая полный путь. (То есть если вы работаете в Windows и хотите выполнить команду, которая в примере приведена с `./`, уберите эти символы.)

команды ./. Имейте в виду, что если вы измените местоположение сценария, может случиться так, что *csh* или *tcsh* смогут определить, где он находится, только после перезагрузки. Чтобы исключить необходимость перезагрузки, после перемещения сценария используйте *rehash*. Рассмотрим пример:

```
% ./mysql_uptime.sh
Uptime  1260348
% mv mysql_uptime.sh /usr/local/bin
% mysql_uptime.sh
mysql_uptime.sh: Command not found.
% rehash
% mysql_uptime.sh
Uptime  1260397
```

Если вы хотите, чтобы время в отчете выводилось не просто в секундах, а в днях, часах, минутах и секундах, то можно использовать вывод предложения STATUS *mysql*, предоставляющий следующую информацию:

```
mysql> STATUS;
Connection id:          12347
Current database:       cookbook
Current user:           cbuser@localhost
Current pager:          stdout
Using outfile:          ''
Server version:         3.23.47-log
Protocol version:       10
Connection:             Localhost via UNIX socket
Client characterset:    latin1
Server characterset:    latin1
UNIX socket:            /tmp/mysql.sock
Uptime:                 14 days 14 hours 2 min 46 sec
```

Для отчета о времени непрерывной работы сервера нужна только строка, начинающаяся с Uptime. Напишем сценарий, который отправляет на сервер команду STATUS и фильтрует вывод при помощи *grep* для извлечения соответствующей строки:

```
#! /bin/sh
# mysql_uptime2.sh - report server uptime

mysql -e STATUS | grep "^Uptime"
```

Результат выглядит так:

```
% ./mysql_uptime2.sh
Uptime:                 14 days 14 hours 2 min 46 sec
```

Два предыдущих сценария указывают, какие предложения необходимо выполнить при помощи опции командной строки *-e*, но можно использовать и другие источники ввода *mysql*, изученные ранее, такие как файлы и каналы. Например, скрипт *mysql_uptime3.sh* подобен *mysql_uptime2.sh*, но передает ввод *mysql* по каналу:

```
#! /bin/sh
```

```
# mysql_uptime3.sh - report server uptime

echo STATUS | mysql | grep "^Uptime"
```

Некоторые оболочки поддерживают так называемый «here-документ», который в принципе подобен файловому вводу для команды, только в данном случае нет явного имени файла. (Другими словами, документ расположен «прямо здесь» в сценарии, а не хранится во внешнем файле.) Для передачи команде ввода посредством here-документа используйте следующий синтаксис:

```
command <<MARKER
input line 1
input line 2
input line 3
...
MARKER
```

<<MARKER оповещает о начале ввода и указывает, какой символ будет маркером конца ввода. Используемый символ может быть произвольным, но стоит выбрать какой-то характерный идентификатор, который не может встретиться во вводе команды.

Если ввод запроса достаточно длинный, то удобнее применять here-документы, а не опцию *-e*, поскольку в подобных ситуациях работа с *-e* становится неудобной, а here-документы, наоборот, предлагают простой и удобный способ записи. Предположим, что у вас есть таблица журнала `log_tbl`, которая содержит столбец `date_added`, показывающий, когда была добавлена каждая строка. Запрос, выводящий количество строк, добавленных вчера, будет выглядеть так:

```
SELECT COUNT(*) As 'New log entries:'
FROM log_tbl
WHERE date_added = DATE_SUB(CURDATE(),INTERVAL 1 DAY);
```

Запрос может быть задан в сценарии при помощи опции *-e*, но такую командную строку будет тяжело читать, ведь запрос очень длинный. В данном случае лучше использовать here-документ, чтобы иметь возможность записать запрос в более удобной для чтения форме:

```
#! /bin/sh
# new_log_entries.sh - count yesterday's log entries

mysql cookbook <<MYSQL_INPUT
SELECT COUNT(*) As 'New log entries:'
FROM log_tbl
WHERE date_added = DATE_SUB(CURDATE(),INTERVAL 1 DAY);
MYSQL_INPUT
```

При использовании *-e* или here-документов вы можете во вводе запроса ссылаться на переменные оболочки (хотя в следующем примере показано, что таких приемов следует избегать). Пусть у вас есть простой сценарий *count_rows.sh* для вычисления количества строк любой таблицы базы данных `cookbook`:

```
#! /bin/sh
# count_rows.sh - count rows in cookbook database table

# require one argument on the command line
if [ $# -ne 1 ]; then
    echo "Usage: count_rows.sh tbl_name";
    exit 1;
fi

# use argument ($1) in the query string
mysql cookbook <<MYSQL_INPUT
SELECT COUNT(*) AS 'Rows in table:' FROM $1;
MYSQL_INPUT
```

Сценарий использует переменные оболочки `$#` (хранит количество аргументов командной строки) и `$1` (хранит первый аргумент после имени сценария). Сценарий *count_rows.sh* убеждается в том, что задан ровно один аргумент, затем использует его как имя таблицы в запросе по подсчету количества строк. Для запуска сценария вызовите его с аргументом-именем таблицы:

```
% ./count_rows.sh limbs
Rows in table:
12
```

Подстановка переменных может быть весьма полезной при формировании запросов, но следует осторожно пользоваться этой возможностью. Злоумышленник может вызвать сценарий так:

```
% ./count_rows.sh "limbs;DROP TABLE limbs"
```

В данном случае результирующий ввод запроса в *mysql* будет таким:

```
SELECT COUNT(*) AS 'Rows in table:' FROM limbs;DROP TABLE limbs;
```

Вычисляется количество строк, а затем таблица удаляется! Так что стоит ограничить использование переменных в личных сценариях. Вместо этого для обеспечения безопасности перепишите сценарий, используя API, разрешающий применение таких специальных символов, как `;`, и умеющий обращаться с ними безопасно (см. рецепт 2.7).

Создание сценариев оболочки в Windows

При работе в Windows вы можете запускать *mysql* из командного файла (файла с расширением *.bat*). Рассмотрим командный файл Windows, *mysql_uptime.bat*, аналогичный представленному ранее сценарию *mysql_uptime.sh* оболочки UNIX:

```
@ECHO OFF
REM mysql_uptime.bat - report server uptime in seconds

mysql -B -N -e "SHOW STATUS LIKE 'Uptime'"
```

Командные файлы можно вызывать без указания расширения *.bat*:

```
C:\> mysql_uptime
Uptime  9609
```

Однако на создание сценариев DOS наложены серьезные ограничения. Например, не поддерживаются here-документы, использование кавычек в аргументах команд также доступно не в полной мере. Можно бороться с этим, установив более разумную рабочую среду (см. врезку «Считать ли оболочку DOS ограниченной?»).

Считать ли оболочку DOS ограниченной?

Если вы относитесь к пользователям UNIX, которые комфортно чувствуют себя при работе с оболочками и утилитами, являющимися частью интерфейса командной строки UNIX, то, вероятно, знакомы с некоторыми командами, использованными в данной главе, такими как *grep*, *sed*, *tr* и *tail*. Эти инструменты обычно всегда доступны в UNIX-системах, поэтому для вас может стать неприятным откровением тот факт, что вы не сможете с ними работать, если вдруг придется работать в оболочке DOS в Windows.

Чтобы сделать окружение командной строки DOS более приятной в работе, можно установить инструментарий Cygnus для Windows (Cygwin) или UNIX для Windows (UWIN). Эти пакеты содержат некоторые из наиболее распространенных оболочек UNIX, а также множество утилит, увидеть которые так надеются пользователи UNIX. Каждому пакету сопутствуют такие программные средства, как компиляторы. Дистрибутивы пакетов доступны в Интернете по следующим адресам:

http://www.cygwin.com/

http://www.research.att.com/sw/tools/uwin/

Если вы будете использовать эти дистрибутивы при работе в Windows, знайте, что для вас не всегда будут правдой мои слова о том, что какая-то команда не работает в Windows. Если вы установите Cygwin или UWIN, то информация данной книги о возможности запуска некоторых команд только в UNIX, но не Windows, окажется неуместной.

2

Создание программы для MySQL

2.0. Введение

В этой главе рассказывается о том, как писать программы, использующие MySQL. Описаны базовые операции API, необходимые для понимания рецептов следующих глав, такие как соединение с сервером MySQL, запуск запросов и извлечение результатов.

Программные интерфейсы приложений для MySQL

В этой книге показано, как создавать программы, работающие с MySQL, с помощью языков программирования Perl, PHP, Python и Java (можно использовать и другие языки). Все клиентские программы MySQL, вне зависимости от того, на каком языке они написаны, имеют одно общее свойство: они устанавливают соединение с сервером через программный интерфейс приложения (application programming interface – API), реализующий протокол передачи данных. Это касается любых программ, будь то утилита командной строки, автоматически запускаемое по заданному расписанию задание или сценарий, используемый на веб-сервере для публикации содержимого базы данных в Интернете. API MySQL обеспечивает разработчикам приложений стандартные возможности работы с базами данных. Каждый API преобразует ваши инструкции в нечто понятное серверу MySQL.

Сам сервер использует протокол низкого уровня, который я буду называть «сырым» (raw). Это уровень прямого сетевого взаимодействия сервера с его клиентами. Клиент устанавливает соединение с портом-слушателем сервера и взаимодействует с ним по протоколу клиент-сервер, осуществляя самые элементарные действия. (По существу, клиент заполняет структуры данных и отправляет их через сеть.) Работа на этом уровне – не самый эффективный путь ни для взаимодействия с сервером (см. врезку «Хотите подключиться к серверу MySQL по Telnet?»), ни для программирования такого взаимодействия. Сырой протокол представляет собой поток двоичных данных – эффективный, но не очень-то удобный в использовании, поэтому разработчики обычно воздерживаются от написания программ, взаимодействующих с сервером

подобным образом. Более удобный способ работы с сервером MySQL обеспечивается программными интерфейсами, стоящими на уровень выше, чем сырой протокол. Интерфейс манипулирует сырым протоколом от имени вашей программы. Он обеспечивает вызовы для таких операций, как соединение с сервером, отправка запросов, извлечение результатов запросов и получение информации о статусе запроса.

В Java-драйверах реализован непосредственно низкоуровневый протокол. Они встраиваются в интерфейс JDBC (Java Database Connectivity), так что вы можете писать свои программы, используя стандартные вызовы JDBC. JDBC передает заявки на выполнение операций с базой данных драйверу MySQL, который отображает их на операции взаимодействия с сервером MySQL по сырому протоколу.

MySQL-драйверы для Perl, PHP и Python придерживаются другого подхода. Они не реализуют непосредственно сырой протокол. Вместо этого они используют клиентскую библиотеку MySQL, входящую в состав дистрибутива. Эта библиотека написана на C и представляет собой основу программного интерфейса приложения, обеспечивающего взаимодействие написанных на C клиентских программ с сервером. Большинство стандартных клиентских приложений дистрибутива MySQL написано на C и используют данный API. Вы также можете применять его в своих программах, и вам просто необходимо сделать это, если вы хотите добиться максимальной эффективности. Однако основная часть сторонних разработок ведется не на C. Тогда API для C может применяться опосредованно, как вложенная библиотека внутри других языков. Именно так взаимодействие с MySQL реализовано для Perl, PHP, Python и многих других языков. API для данных языков высокого уровня представляет собой «обертку» («wrapper»), в которую упакованы программы на C и с помощью которой они встраиваются в эти трансляторы.

Преимущество данного подхода в том, что вместо вас с сервером общается языковой процессор, использующий для этой цели программы на C, вы же работаете с интерфейсом, более подходящим для выполнения операций с базой данных. Например, языки сценариев, такие как Perl, имеют развитые средства работы с текстом и не требуют явного создания и уничтожения строковых буферов, как это бывает при работе с C. Языки высокого уровня позволяют вам сконцентрироваться на том, что вы пытаетесь сделать, не тратя времени на детали, без которых невозможно программирование непосредственно на C.

В этой книге нет подробного описания API для C, так как он никогда не применяется напрямую. Создаваемые программы используют интерфейсы высокого уровня, работающие поверх данного API. Если же вы хотите попробовать писать клиентские программы MySQL на C, вам могут пригодиться следующие источники информации:

- В MySQL Reference Manual (справочник по MySQL) есть глава, содержащая информацию о функциях API для C. Следует также просмотреть исходные тексты написанных на C стандартных клиентских программ MySQL, входящих в дистрибутив в исходных текстах. Учебник и сам

дистрибутив доступны в Интернете на сайте MySQL по адресу *http://www.mysql.com/*; кроме того, учебник издан O'Reilly & Associates.

- Справочный материал по API для C приведен в книге «MySQL» (New Riders), которая также содержит главу с подробнейшими инструкциями по написанияю MySQL-программ на C. Нет необходимости покупать книгу, чтобы прочитать одну эту главу – она доступна в формате PDF по адресу *http://www.kitebird.com/mysql-book/*. Исходные тексты программ-примеров, обсуждаемых в главе, также размещены на сайте. Программы были написаны специально в учебных целях, поэтому их, вероятно, проще понять, чем стандартные клиентские программы дистрибутива исходных текстов MySQL.

Хотите подключиться к серверу MySQL по Telnet?

Некоторые протоколы сетевого взаимодействия, например SMTP и POP, основаны на ASCII. Поэтому по таким протоколам можно напрямую общаться с сервером, используя Telnet для соединения с портом-слушателем сервера и вводя информацию с клавиатуры. Некоторые считают, что так же можно взаимодействовать и с сервером MySQL: установить по Telnet соединение с сервером и вводить команды. Но ничего не получится: дело в том, что используемый сервером сырой протокол по природе своей является двоичным. Можете проверить самостоятельно. Предположим, что сервер MySQL работает на локальной машине и прослушивает порт по умолчанию (3306). Подключимся к нему, используя такую команду:

```
% telnet localhost 3306
```

Вы увидите нечто напоминающее номер версии, а также, вероятно, еще ряд непонятных символов. Это и есть сырой протокол. Вряд ли вы далеко продвинетесь, общаясь с сервером таким способом, поэтому ответом на часто задаваемый вопрос «Как установить Telnet-соединение с сервером MySQL?» должен быть: «Не стоит этим заниматься.» Единственное, что вы можете узнать таким путем: в рабочем ли сервер состоянии и прослушивает ли он данный порт.

Клиентские API для MySQL обеспечивают следующие возможности, каждая из которых раскрыта далее в главе:

Соединение с сервером MySQL, выбор базы данных, отключение от сервера. Каждая программа, использующая MySQL, должна сначала установить соединение (connection) с сервером, кроме того, большинство программ позволяет указывать базу данных для работы. Некоторые API требуют указания имени базы данных в момент соединения (поэтому соединение и выбор базы данных будут рассмотрены в одном разделе). В других API выбор базы данных осуществляется посредством явного вызова. Кроме того, «добропорядочные» программы MySQL закрывают соединение с сервером после окончания работы.

Контроль ошибок. Многие программисты пишут программы для MySQL, которые вообще не проводят никаких проверок на ошибки, что делает эти программы неудобными для отладки в случае возникновения проблем. Любая операция с базой данных может не выполниться, и вы должны знать, как определить, когда это произошло и почему. Необходимо сделать так, чтобы вы могли выйти из программы или уведомить пользователя о возникшей проблеме.

Запуск запросов и получение результатов. Смысл соединения с сервером базы данных состоит в выполнении запросов. Каждый API предоставляет по крайней мере один способ запуска запросов и множество функций по обработке результатов запроса. Из-за обилия предлагаемых возможностей соответствующий раздел будет самым большим в этой главе.

Использование в запросах подготовленных предложений и заполнителей. Для того чтобы написать запрос, использующий определенные значения данных, можно вставить их непосредственно в строку запроса. Но большинство API предлагают и другой механизм, позволяющий заранее подготовить запрос, ссылающийся на данные в символической форме. Тогда при выполнении предложения вы отдельно вводите значения данных, а API подставляет их в строку запроса.

Использование в запросах специальных символов и значений NULL. Некоторые символы, например, кавычки или обратная косая черта (обратный слэш), имеют специальное значение. Поэтому, составляя запросы, содержащие такие символы, следует соблюдать меры предосторожности. То же самое относится и к значениям `NULL`. Если вы не будете их корректно обрабатывать, ваши программы могут породить ошибочные или выдающие непредвиденные результаты SQL-предложения. В данном разделе будет рассказано о том, как обойти такие ситуации.

Обработка значений NULL в результирующих множествах. На значения `NULL` необходимо обратить внимание не только при построении запросов, но и при извлечении их результатов. Каждый API использует свои правила при работе с `NULL`.

Чтобы писать программы (независимо от того, какой именно язык используется), необходимо уметь выполнять все перечисленные выше фундаментальные операции API, поэтому реализация каждой из них представлена на нескольких языках (PHP, Perl, Python и Java). После того как вы посмотрите, как каждый из API выполняет указанные операции, станут видны аналогии и легче будет понять рецепты, в которых примеры даны на языке, с которым вы не очень близко знакомы. (В последующих главах рецепты обычно иллюстрируются программами на одном-двух языках.)

Признаю, что тем, кого интересует лишь один какой-то API, может показаться, что приводить каждый рецепт на всех четырех языках излишне. В подобных случаях я рекомендовал бы следующий подход: читаете только вводную часть рецепта, в которой представлены общие сведения о предмете, затем переходите непосредственно к разделу, относящемуся к интересующему вас языку. Остальные языки пропускаете. Если в дальнейшем понадобится

написать программу на другом языке, вы всегда сможете вернуться к рецепту и прочитать другие разделы.

В главе также рассматриваются операции, которые не входят непосредственно в API для MySQL, но могут упростить вашу работу с API:

Создание библиотечных файлов. Вы пишете одну программу за другой и рано или поздно обнаруживаете, что некоторые операции выполняются снова и снова. Есть возможность инкапсулировать код таких операций в библиотечный файл, тогда при выполнении их во множестве сценариев не понадобится включать весь код в каждый сценарий. Уменьшается количество дублируемого кода, программы становятся более переносимыми. В разделе будет показано, как написать для каждого API библиотечный файл, который будет содержатьщий функцию соединения с сервером – операцию, которую должна выполнять любая программа, использующая MySQL. (В последующих главах библиотека будет пополняться программами и для других операций.)

Создание объектно-ориентированного интерфейса MySQL для PHP. API для Perl, Python и Java основаны на классах и предлагают объектно-ориентированную модель программирования, архитектура которой не зависит от базы данных. В основе же встроенного интерфейса PHP лежат вызовы специальных функций MySQL. В разделе описано, как написать PHP-класс, который можно использовать для реализации объектно-ориентированного подхода при создании сценариев MySQL.

Способы получения параметров соединения. Предложенный ранее раздел об установлении соединения с сервером MySQL использует параметры соединения, «зашитые» в код. Но существует множество способов получения параметров, начиная с хранения их в специальном файле и заканчивая пользовательским вводом с клавиатуры в процессе работы программы.

Чтобы не вводить текст примеров с клавиатуры, вы можете воспользоваться исходными текстами из дистрибутива `recipes` (см. приложение А). Когда в примере сказано что-то вроде «создайте файл *xyz*, содержащий такую-то информацию», вы просто будете использовать соответствующий файл из `recipes`. Сценарии этой главы расположены в каталоге *api*, а библиотечные файлы выделены в отдельный каталог *lib*.

В примерах главы используется основная таблица с именем `profile`. Она создается в рецепте 2.4 (говорю об этом для тех, кто не читает книгу подряд, а перескакивает с места на место). Обратите внимание на замечание в самом конце главы, касающееся необходимости вернуть таблицу `profile` в исходное состояние для использования в других главах.

Принятые допущения

Для наиболее эффективной работы с предложенным в главе материалом необходимо выполнить ряд условий:

- Должна быть установлена MySQL-поддержка тех языковых процессоров, которые планируется использовать. Если вам необходимо установить какой-то API, обратитесь к приложению A.
- Должны быть созданы учетная запись пользователя MySQL для доступа к серверу и база данных для опробования запросов. Как уже говорилось в главе 1, в примерах используется учетная запись MySQL с именем `cbuser` и паролем `cbpass`, соединение устанавливается с сервером MySQL, работающим на локальном компьютере, доступ осуществляется к базе данных `cookbook`. Если вам необходимо создать пользователя MySQL или базу данных, следуйте инструкциям данной главы.
- Предлагаемые рецепты предполагают наличие некоторых базовых знаний по языкам API. Если в рецепте использована незнакомая вам конструкция, обратитесь к общему руководству по используемому языку. Некоторые полезные источники информации перечислены в приложении C.
- Для корректного выполнения некоторых программ может потребоваться установить соответствующие переменные окружения. Этот процесс подробно описан в рецепте 1.8.

2.1. Соединение с сервером MySQL, выбор базы данных и отключение

Задача

Необходимо установить соединение с сервером для работы с базой данных, а по ее окончании – завершить соединение.

Решение

Каждый API содержит функции подключения к серверу и отключения. Программы, устанавливающие соединение, требуют ввода параметров для указания учетной записи MySQL, которую вы хотите использовать. Вы также можете указать имя базы данных, с которой собираетесь работать. Некоторые API позволяют указать эти сведения при установлении соединения, другие же осуществляют специальный вызов уже после того, как соединение установлено.

Обсуждение

Программы, приведенные в данном разделе, показывают, как выполнять три фундаментальные операции, встречающиеся в подавляющем большинстве программ MySQL:

Установление соединения с сервером MySQL. Каждая работающая с MySQL программа делает это, вне зависимости от того, какой API используется. В разных API могут по-разному указываться параметры соединения, какие-то API могут быть более гибкими, чем другие. Однако сходных черт очень много. Например, необходимо указывать название хоста, на котором

работает сервер, а также имя и пароль пользователя MySQL, осуществляющего доступ к серверу.

Выбор базы данных. Большая часть программ MySQL выбирают базу данных или при соединении с сервером, или непосредственно после этого.

Отключение от сервера. Каждый API обладает средствами завершения соединения. Лучше закрывать соединение сразу же после окончания работы, чтобы освободить ресурсы, занятые на обслуживание соединения. В противном случае, если программа после обращения к серверу выполняет какие-то дополнительные вычисления, соединение останется открытым дольше, чем это необходимо. Кроме того, рекомендуется закрывать соединение явно. Если программа просто завершается, не закрывая соединение, сервер MySQL со временем узнает об этом, но если вы явно закроете соединение, сервер сможет прекратить его поддерживать незамедлительно.

Программы для каждого API в примерах данного раздела показывают, как установить соединение с сервером, выбрать базу данных `cookbook` и отключиться. Однако может оказаться, что вам необходимо написать программу MySQL, которая не выбирает базу данных. Например, если вы планируете написать запрос, которому не требуется база данных по умолчанию, такой как `SHOW VARIABLES` или `SHOW DATABASES`. Или если вы пишете интерактивную программу, которая подключается к серверу и запрашивает имя базы данных у пользователя после того, как соединение установлено. Чтобы охватить и такие ситуации, при обсуждении API будет показано, как установить соединение, не выбирая конкретную базу данных для работы.

Локальный хост и MySQL

Одним из параметров, указываемых при соединении с сервером MySQL, является имя хоста, на котором работает сервер. Большинство программ воспринимает как синонимы имя *localhost* и IP-адрес `127.0.0.1`. В UNIX программы MySQL ведут себя по-другому: они интерпретируют имя *localhost* специальным образом и пытаются осуществить соединение с сервером, используя сокет домена UNIX. Для того чтобы принудительно установить TCP/IP-соединение с локальной машиной, используйте IP-адрес `127.0.0.1` вместо имени *localhost*. (При работе в Windows имя *localhost* и IP-адрес `127.0.0.1` трактуются одинаково, так как в Windows нет сокетов домена UNIX.)

По умолчанию для TCP/IP-соединений используется порт 3306. Путевое имя сокета UNIX-домена может быть разным, часто это */tmp/mysql.sock*. В рецептах показано, как явно указывать путевое имя файла сокета или номер порта TCP/IP (если не хочется использовать значения по умолчанию).

Perl

Для того чтобы писать сценарии для MySQL на Perl, необходимо установить модуль DBI, а также специальный драйвер MySQL, *DBD::mysql*. Если у вас

они еще не установлены, обратитесь к приложению А. Существует более ранний интерфейс для Perl, называемый MysqlPerl, но он уже устарел и здесь рассматриваться не будет.

Рассмотрим пример простого сценария Perl, который осуществляет соединение с базой данных `cookbook`, а потом отключается от нее:

```
#! /usr/bin/perl -w
# connect.pl - установить соединение с сервером MySQL

use strict;
use DBI;
my $dsn = "DBI:mysql:host=localhost;database=cookbook";
my $dbh = DBI->connect ($dsn, "cbuser", "cbpass")
                or die "Cannot connect to server\n";
print "Connected\n";
$dbh->disconnect ();
print "Disconnected\n";
exit (0);
```

Чтобы опробовать этот сценарий, создайте файл с именем *connect.pl*, содержащий приведенный выше код. Если вы хотите запустить *connect.pl* в UNIX, может потребоваться изменить путевое имя в первой строке (если у вас Perl находится не в каталоге */usr/bin/perl*). Используя команду *chmod +x*, сделайте файл исполняемым и вызовите его:

```
% chmod +x connect.pl
% ./connect.pl
Connected
Disconnected
```

В Windows нет необходимости использовать *chmod* – можно просто запускать *connect.pl*:

```
C:\> perl connect.pl
Connected
Disconnected
```

Если у вас определено соответствие имен файлов, позволяющее исполнять файлы с расширением *.pl* непосредственно из командной строки, то явно вызывать Perl не нужно:

```
C:\> connect.pl
Connected
Disconnected
```

Более подробная информация о запуске самостоятельно написанных программ приведена во врезке «Использование исполняемых программ» рецепта 1.32.

Опция *-w* включает режим предупреждений, в котором Perl выдает предупреждения для каждой сомнительной конструкции. В нашем сценарии таких конструкций нет, но стоит завести привычку использовать *-w*. Производя изменения в сценарии по ходу работы, вы заметите, что зачастую Perl делает очень полезные замечания о таких изменениях.

Строка `use strict` включает полную проверку переменных и заставляет Perl жаловаться на любые переменные, которые используются, не будучи предварительно определенными. Это разумная мера предосторожности, помогающая обнаружить ошибки, которые могли бы остаться незамеченными. Предложение `use DBI` указывает Perl на необходимость использования модуля DBI. Загружать модуль драйвера MySQL (*DBD::mysql*) явно не нужно; DBI сделает это самостоятельно, когда сценарий будет осуществлять соединение с сервером баз данных.

Две следующие строки устанавливают соединение с MySQL, определяя имя источника данных (DSN, Data Source Name) и вызывают DBI-метод `connect()`. Аргументы `connect()` – DSN, имя пользователя MySQL, пароль и все атрибуты соединения, которые вы хотите указать. Обязательно указывать только DSN, хотя обычно все же указывают имя пользователя и пароль.

DSN определяет, какой драйвер базы данных следует использовать, а также другие параметры, указывающие, с чем будет осуществляться соединение. DSN для программ MySQL имеет следующий формат: `DBI:mysql:`*опции*. Рассмотрим эти три составляющие:

- Первый компонент – это всегда `DBI` (регистр ввода не имеет значения; подходит и `dbi`, и `DBI`).
- Второй компонент указывает DBI, какой драйвер базы данных следует использовать. Для MySQL необходимо ввести `mysql`, и на этот раз регистр имеет значение. Нельзя использовать `MySQL`, `MYSQL` или другие комбинации строчных и заглавных букв.
- Третья составляющая, если она присутствует, представляет собой список пар *имя=значение*, разделенных запятыми и указывающих дополнительные опции соединения. Порядок ввода пар не имеет значения. В нашем случае наиболее существенными опциями являются `host` и `database`. Они задают имя хоста, на котором работает сервер MySQL, и базу данных, с которой предполагается работать. Обратите внимание, что второе двоеточие в DSN *не* является необязательным и должно присутствовать, даже если никакие опции не задаются.

DSN для соединения с базой данных `cookbook`, размещенной на локальном хосте *localhost*, выглядит так:

```
DBI:mysql:host=localhost;database=cookbook
```

Если не указать опцию `host`, будет использовано значение по умолчанию – `localhost`. Поэтому следующие два DSN эквивалентны:

```
DBI:mysql:host=localhost;database=cookbook
DBI:mysql:database=cookbook
```

Если не задать опцию `database`, база данных при соединении не выбирается.

Вторым и третьим аргументами вызова `connect()` являются имя пользователя MySQL и пароль. Можно ввести и четвертый аргумент, который будет определять поведение DBI в случае возникновения ошибок. По умолчанию при

ошибке DBI выводит соответствующее сообщение, но не прекращает выполнение сценария. Поэтому для того чтобы сообщить о неудачном выполнении сценария, *connect.pl* проверяет, возвращает ли `connect()` значение `undef`:

```
my $dbh = DBI->connect ($dsn, "cbuser", "cbpass")
                or die "Cannot connect to server\n";
```

Возможно применение других стратегий обработки ошибок. Например, вы можете задать автоматическое завершение сценария в случае ошибки. Для этого выключите атрибут `PrintError` и включите `RaiseError`. Теперь вам уже не придется самим проверять наличие ошибок:

```
my $dbh = DBI->connect ($dsn, $user_name, $password,
                        {PrintError => 0, RaiseError => 1});
```

Контроль ошибок также будет рассмотрен в рецепте 2.2.

Если метод `connect()` успешно выполнен, он возвращает дескриптор базы данных, содержащий информацию о состоянии соединения. (На языке DBI ссылки на объекты называются «дескрипторами».) Позже вы познакомитесь с другими дескрипторами, такими как дескриптор предложения, сопоставленный конкретному запросу. Сценарии DBI, приведенные в книге, используют следующие обозначения для дескрипторов базы данных и предложения: `$dbh` и `$sth`.

Дополнительные параметры соединения

При соединениях с *локальным хостом* (*lokalhost*) вы можете указывать в DSN опцию `mysql_socket` для определения пути к сокету домена UNIX:

```
my $dsn = "DBI:mysql:host=localhost;mysql_socket=/var/tmp/mysql.sock"
          . ";database=cookbook";
```

Опция `mysql_socket` доступна в версиях MySQL 3.21.15 и выше.

Для соединений с нелокальным хостом (non-*lokalhost*) можно указать опцию `port`, чтобы задать номер порта.

```
my $dsn = "DBI:mysql:host=mysql.snake.net;port=3307;database=cookbook";
```

PHP

Для написания сценариев PHP, использующих MySQL, необходимо, чтобы интерпретатор PHP был скомпилирован с поддержкой MySQL. Если это условие не выполнено, сценарий завершится сообщением об ошибке:

```
Fatal error: Call to undefined function: mysql_connect()
```

Если вы увидели такое сообщение, обратитесь к инструкциям, содержащимся в дистрибутиве PHP, чтобы включить поддержку MySQL.

Сценарии PHP обычно создаются для веб-серверов. Я буду считать, что если вы собираетесь использовать PHP подобным образом, то имеете возможность вставить PHP-сценарий в каталог документов сервера и запросить его из броузера, чтобы сценарий был выполнен. Например, если вы работаете

с веб-сервером Apache, расположенным по адресу *http://apache.snake.net/*, и устанавливаете PHP-сценарий *myscript.php* на верхнем уровне дерева документов Apache, обратиться к этому сценарию можно будет, запрашивая такой URL:

http://apache.snake.net/myscript.php

В данной книге в качестве расширения имен файлов сценариев PHP используется *.php*. Если вы работаете с другим расширением, например *.php3* или *.phtml*, следует изменить названия сценариев или настроить сервер так, чтобы он распознавал расширение *.php*. В противном случае в ответ на запрос сценария броузер выведет его текст. Это не то, чего хотелось бы, особенно если в сценарии присутствуют имя пользователя MySQL и его пароль. (Подробная информация о конфигурировании сервера Apache для работы с PHP приведена в рецепте 16.2.)

Сценарии PHP часто пишутся как смесь HTML- и PHP-кода, где код PHP заключен в специальные теги `<?php` и `?>`. Рассмотрим простой пример:

```
<html>
<head><title>A simple page</title></head>
<body>
<p>
<?php
    print ("I am PHP code, hear me roar!\n");
?>
</p>
</body>
</html>
```

Для краткости будем опускать внешние теги `<?php` и `?>` в примерах для PHP, целиком состоящих из кода. Если же примеры содержат и HTML-, и PHP-код, в них будут присутствовать теги.

Для того чтобы использовать MySQL в сценарии PHP, вы устанавливаете соединение с сервером MySQL и выбираете базу данных для работы. Выбор базы данных происходит в два этапа посредством вызова функций `mysql_connect()` и `mysql_select_db()`. Напишем наш первый PHP-сценарий, *connect.php*, и посмотрим, как он работает:

```
# connect.php - установить соединение с сервером MySQL
if (!($conn_id = @mysql_connect ("localhost", "cbuser", "cbpass")))
    die ("Cannot connect to server\n");
print ("Connected\n");
if (!@mysql_select_db ("cookbook", $conn_id))
    die ("Cannot select database\n");
mysql_close ($conn_id);
print ("Disconnected\n");
```

Функция `mysql_connect()` принимает три аргумента: хост, на котором работает сервер MySQL, а также имя и пароль учетной записи MySQL, которая будет использоваться. Если соединение успешно установлено, `mysql_connect()`

возвращает идентификатор соединения, который в дальнейшем может быть передан другим функциям, работающим с MySQL. В сценариях PHP данной книги принято использовать для обозначения идентификаторов соединений `$conn_id`.

Если же попытка соединения оказывается неудачной, `mysql_connect()` выводит предупреждение и возвращает `FALSE`. (Сценарий препятствует выводу подобных предупреждений, помещая символ `@` (оператор подавления предупреждений) перед названием функции, чтобы печатать в соответствующих ситуациях собственные сообщения.)

Функция `mysql_select_db()` принимает в качестве аргументов имя базы данных и необязательный идентификатор соединения. Если второй аргумент не задается, функция считает, что следует использовать текущее соединение (то есть открытое последним). Только что приведенный сценарий вызывает `mysql_select_db()` сразу же после того, как соединение установлено, поэтому два следующих вызова эквивалентны:

```
if (!@mysql_select_db ("cookbook", $conn_id))
    die ("Cannot select database\n");

if (!@mysql_select_db ("cookbook"))
    die ("Cannot select database\n");
```

Если функции `mysql_select_db()` удается выбрать базу данных, она возвращает `TRUE`. В противном случае она выводит предупреждение и возвращает `FALSE`. (И опять-таки, как и в случае с вызовом `mysql_connect()`, сценарий использует оператор `@` для подавления вывода предупреждений.) Если вам не нужно выбирать базу данных, просто пропустите вызов.

Чтобы опробовать сценарий *connect.php*, скопируйте его в дерево документов веб-сервера и запросите из броузера. Если у вас есть автономная версия интерпретатора PHP, которую можно запускать из командной строки, вы можете протестировать сценарий, не прибегая к помощи веб-сервера и броузера:

```
% php -q connect.php
Connected
Disconnected
```

В действительности PHP предоставляет две функции, обеспечивающие соединение с сервером MySQL. Сценарий *connect.php* использует функцию `mysql_connect()`, но вы можете использовать вместо нее `mysql_pconnect()`, если хотите установить постоянное соединение, которое не будет завершено по окончании работы сценария. Такое соединение может повторно использоваться несколькими сценариями PHP, запускаемыми веб-сервером, что избавляет от «накладных расходов» на открытие нового соединения. Однако MySQL так рационально открывает соединения, что вы можете не заметить особой разницы между двумя функциями. Кроме того, необходимо учитывать, что применение `mysql_pconnect()` может приводить к слишком большому количеству открытых соединений. В результате сервер MySQL перестает принимать новые соединения, так как предельное количество постоянных соеди-

нений уже открыто процессами веб-сервера. В подобном случае для разрешения проблемы следует использовать `mysql_connect()`.

Дополнительные параметры соединения

При соединениях с *локальным хостом* (*localhost*) вы можете указать путевое имя сокета домена UNIX, добавив к имени хоста в вызове *:/путь/к/сокету*:

```
$hostname = "localhost:/var/tmp/mysql.sock";
if (!($conn_id = @mysql_connect ($hostname, "cbuser", "cbpass")))
    die ("Cannot connect to server\n");
```

Для соединений с нелокальным хостом можно задать номер порта, добавив к имени хоста опцию *:номер_порта*:

```
$hostname = "mysql.snake.net:3307";
if (!($conn_id = @mysql_connect ($hostname, "cbuser", "cbpass")))
    die ("Cannot connect to server\n");
```

Опция указания путевого имени сокета появилась в PHP 3.0.B4, а опция, задающая номер порта, – в PHP 3.0.10.

В PHP 4 можно использовать инициализационный файл PHP, чтобы определить значения по умолчанию для имени хоста, имени пользователя, пароля, пути к сокету или номера порта; установите конфигурационные директивы `mysql.default_host`, `mysql.default_user`, `mysql.default_password`, `mysql.default_socket` или `mysql.default_port` в соответствующие значения.

Python

Тем, кто хочет писать MySQL-программы на языке Python, необходим модуль MySQLdb, который обеспечивает взаимодействие MySQL с интерфейсом DB-API языка Python. Если в вашем распоряжении нет этого модуля, обратитесь к приложению A. DB-API, как и модуль Perl DBI, предоставляет независящий от базы данных доступ к серверу БД и заменяет собой более ранние модули Perl доступа к базам данных, каждый их которых имел собственный интерфейс и набор правил вызова функций. В этой книге устаревший интерфейс Python для MySQL не рассматривается.

Python избегает использования функций, возвращающих специальное значение, чтобы указать на наличие ошибки. Другими словами, вы не можете написать такой код:

```
if (func1 () == some_bad_value or func2 () == another_bad_value):
    print "An error occurred"
else:
    print "No error occurred"
```

Вместо этого следует поместить исполняемые предложения в блок `try`. Ошибки порождают исключения, которые вы можете «отлавливать» в блоке `except`, содержащем код обработки ошибок:

```
try:
    func1 ()
```

```
    func2 ()
except:
    print "An error occurred"
```

Исключения, возникающие на верхнем уровне сценария (то есть вне блоков `try`), улавливаются обработчиком исключений по умолчанию, который выводит трассировку стека и завершает программу.

Для работы с интерфейсом DB-API импортируйте тот модуль драйвера базы данных, который планируете использовать (MySQLdb для программ MySQL). Затем создайте объект соединения с базой данных, вызвав метод драйвера `connect()`. Этот объект обеспечивает доступ к другим методам DB-API, таким как `close()`, который разрывает соединение с сервером БД. Приведем короткую программу на Python, *connect.py*, чтобы проиллюстрировать эти операции:

```
#! /usr/bin/python
# connect.py - установить соединение с сервером MySQL

import sys
import MySQLdb

try:
    conn = MySQLdb.connect (db = "cookbook",
                            host = "localhost",
                            user = "cbuser",
                            passwd = "cbpass")
    print "Connected"
except:
    print "Cannot connect to server"
    sys.exit (1)

conn.close ()
print "Disconnected"
sys.exit (0)
```

Строки `import` предоставляют сценарию доступ к модулю `sys` (необходимому для функции `sys.exit()`) и модулю MySQLdb. Затем сценарий пытается установить соединение с сервером MySQL, вызывая `connect()` для получения объекта соединения, `conn`. В сценариях Python данной книги для объектов соединений будет использоваться обозначение `conn`.

Если соединение установить не удается, возникает исключение, сценарий выводит сообщение об ошибке. В противном случае он закрывает соединение, используя метод `close()`.

Аргументы `connect()` поименованы, поэтому их порядок не имеет значения. Если при вызове `connect()` вы не укажете аргумент `host`, будет использовано значение по умолчанию `localhost`. Если вы не укажете аргумент `db` или зададите для него значение `""` (пустая строка), база данных не будет выбрана. Если же указать значение `None`, вызов завершится с ошибкой.

Чтобы опробовать сценарий, создайте файл *connect.py*, содержащий рассмотренный код. Если вы работаете в UNIX и ваш интерпретатор Python нахо-

дится не в каталоге */usr/bin/python*, придется изменить путь в первой строке сценария. Затем сделайте сценарий исполняемым при помощи *chmod +x* и запустите его:

```
% chmod +x connect.py
% ./connect.py
Connected
Disconnected
```

В Windows запустите сценарий следующим образом:

```
C:\> python connect.py
Connected
Disconnected
```

Если у вас определено соответствие имен файлов, позволяющее исполнять файлы с расширением *.py* непосредственно из командной строки, то явно вызывать Python не нужно:

```
C:\> connect.py
Connected
Disconnected
```

Дополнительные параметры соединения

При соединениях с локальным хостом (*localhost*) вы можете указать путь к сокету домена UNIX, задав параметр `unix_socket`:

```
conn = MySQLdb.connect (db = "cookbook",
                        host = "localhost",
                        unix_socket = "/var/tmp/mysql.sock",
                        user = "cbuser",
                        passwd = "cbpass")
```

Для соединений с нелокальным хостом вы можете указать параметр `port`, чтобы задать номер порта:

```
conn = MySQLdb.connect (db = "cookbook",
                        host = "mysql.snake.net",
                        port = 3307,
                        user = "cbuser",
                        passwd = "cbpass")
```

Java

Java-программы, работающие с базами данных, пишутся при помощи интерфейса JDBC и драйвера той конкретной машины базы данных, к которой вы хотите получить доступ. Архитектура JDBC подобна используемой модулями Perl DBI и Python DB-API: общий интерфейс, используемый в сочетании со специальным драйвером для каждой базы данных. Сам язык Java также похож на Python: вызываемые функции не проверяются на специальное значение, сигнализирующее о возникшей ошибке. Вместо этого применяются обработчики ошибок, вызываемые при генерировании исключений.

Для программирования на Java необходим набор инструментальных средств разработки программного обеспечения (SDK, software development kit). Если вам нужно установить такой набор, обратитесь за пояснениями к врезке «Установка Java SDK». Кроме того, для написания Java-программ, работающих с MySQL, вам понадобится специальный JDBC-драйвер для MySQL. В приложении A перечислено несколько таких драйверов. Я предпочитаю MySQL Connector/J, так как он распространяется бесплатно и активно поддерживается. Вы же, если хотите, можете использовать другие драйверы. (MySQL Connector/J является преемником MM.MySQL, и если MM.MySQL у вас уже установлен, используйте его. В коде придется сделать лишь одно небольшое изменение: каждый раз, когда вы встретите `org.gjt.mm.mysql`, заменяйте это на `com.mysql.jdbc`.)

Установка Java SDK

Java SDK для Solaris, Linux и Windows доступны в Интернете по адресу *java.sun.com*, но вполне может быть, что необходимый инструментарий у вас уже установлен или же может быть получен другим способом. Например, *javac*, *jikes* и другие средства, необходимые для создания Java-приложений в Mac OS X, находятся в дистрибутиве Developer Tools, который можно получить по адресу *connect.apple.com*.

Если Java SDK еще не установлен, обратитесь к *java.sun.com*, выполните установку и задайте для переменной окружения `JAVA_HOME` значение, соответствующее путевому имени каталога установки SDK. В рассматриваемых примерах за каталог установки SDK принимается */usr/local/java/jdk* для UNIX и *D:\jdk* для Windows, так что `JAVA_HOME` определяется так:

```
export JAVA_HOME=/usr/local/java/jdk      (sh, bash и т. д.)
setenv JAVA_HOME=/usr/local/java/jdk      (csh, tcsh и т. д.)
set JAVA_HOME=D:\jdk                      (Windows)
```

Укажите путевое имя, используемое в вашей системе. Чтобы изменение значения переменной окружения вступило в силу, следует выйти из системы и войти в нее заново, если вы работаете в UNIX, или же перезагрузить компьютер, если речь идет о Windows. Подробная информация об установке переменных окружения приведена в рецепте 1.8.

Рассмотрим программу на Java, *Connect.java*, которая осуществляет соединение с сервером MySQL и отключение от него:

```
// Connect.java - установить соединение с сервером MySQL

import java.sql.*;

public class Connect
{
    public static void main (String[] args)
    {
```

```
        Connection conn = null;
        String url = "jdbc:mysql://localhost/cookbook";
        String userName = "cbuser";
        String password = "cbpass";

        try
        {
            Class.forName ("com.mysql.jdbc.Driver").newInstance ();
            conn = DriverManager.getConnection (url, userName, password);
            System.out.println ("Connected");
        }
        catch (Exception e)
        {
            System.err.println ("Cannot connect to server");
        }
        finally
        {
            if (conn != null)
            {
                try
                {
                    conn.close ();
                    System.out.println ("Disconnected");
                }
                catch (Exception e) { /* ignore close errors */ }
            }
        }
    }
}
```

Предложение `import java.sql.*` ссылается на классы и интерфейсы, которые обеспечивают доступ к типам данных, используемым для управления различными аспектами взаимодействия с сервером базы данных. Это необходимо во всех программах JDBC.

Соединение с сервером происходит в два этапа. Сначала зарегистрируйте драйвер базы данных при помощи JDBC, вызвав `Class.forName()`. Затем инициируйте соединение, вызвав `DriverManager.getConnection()` и получите объект `Connection`, содержащий информацию о состоянии соединения. Java-программы данной книги используют для объектов соединений обозначение `conn`.

Если вы используете драйвер MySQL Connector/J JDBC, указывайте его имя как `com.mysql.jdbc.Driver`. Те, кто работает с другим драйвером, должны найти его название в документации. `DriverManager.getConnection()` принимает три аргумента: URL, описывающий, с каким адресом следует установить соединение и с какой базой данных работать, имя пользователя MySQL, пароль. Формат строки URL таков:

```
jdbc:драйвер://имя_хоста/имя_базы_данных
```

Этот формат придерживается общепринятого в Java соглашения о том, что URL сетевого ресурса должен начинаться с указателя протокола. Для JDBC-

программ таким протоколом является `jdbc`. Кроме того, необходимо указать обозначение подпротокола – название драйвера (в программах MySQL это `mysql`). Многие составляющие URL соединения необязательны, но начальные указатели протокола и подпротокола к ним не относятся. Если вы не введете *имя_хоста*, будет использовано значение по умолчанию – `localhost`. Если не указать *имя_базы_данных*, то база данных не будет выбрана в момент установки соединения. Но в любом случае нельзя пренебрегать слэшами (символами косая черта). Например, чтобы установить соединение с локальным хостом без выбора базы данных, следует указать такой URL:

```
jdbc:mysql:///
```

Чтобы протестировать программу, нужно скомпилировать ее и выполнить. Предложение `class` указывает имя программы (в данном случае – `Connect`). Файл, содержащий программу, должен иметь имя, совпадающее с указанным, и расширение *.java*, поэтому файл для программы из примера будет называться *Connect.java*.[1] Компилируем программу с помощью компилятора *javac*:

```
% javac Connect.java
```

Если вы предпочитаете другой компилятор Java, просто подставьте его название в команды компиляции. Например, если вы собираетесь работать с Jikes, компилируйте файл так:

```
% jikes Connect.java
```

Компилятор *javac* (или *jikes*, или еще какой-нибудь) формирует компилированный байт-код в файле класса *Connect.class*. Используйте программу *java* для запуска файла класса (обратите внимание на то, что имя файла класса указывается без расширения *.class*):

```
% java Connect
Connected
Disconnected
```

Перед компиляцией и запуском программы из примера может потребоваться установить переменную `CLASSPATH`. Значение `CLASSPATH` должно включать, как минимум, ваш текущий каталог (`.`) и путь к драйверу MySQL Connector/J JDBC. У меня в системе этот драйвер находится в */usr/local/lib/java/lib/mysql-connector-java-bin.jar*, поэтому для *tcsh* или *csh* я бы установил `CLASSPATH` так:

```
setenv CLASSPATH .:/usr/local/lib/java/lib/mysql-connector-java-bin.jar
```

Для таких оболочек, как *sh*, *bash* и *ksh*, я бы указал:

```
export CLASSPATH=.:/usr/local/lib/java/lib/mysql-connector-java-bin.jar
```

[1] При желании сделать копию *Connect.java*, чтобы использовать ее в качестве основы для новой программы, необходимо изменить имя класса в предложении `class` так, чтобы оно совпадало с именем нового файла.

При работе в Windows, если бы драйвер находился в каталоге *D:\Java\lib*, я бы установил CLASSPATH так:

```
CLASSPATH=.;D:\Java\lib\mysql-connector-java-bin.jar
```

Вам может понадобиться добавить другие каталоги классов или библиотеки в переменную CLASSPATH; все зависит от того, как именно настроена ваша система.

Внимательно относитесь к Class.forName()!

Программа *Connect.java* регистрирует драйвер JDBC так:

```
Class.forName ("com.mysql.jdbc.Driver").newInstance ();
```

Предполагается, что вы можете регистрировать драйверы, не вызывая newInstance():

```
Class.forName ("com.mysql.jdbc.Driver");
```

Однако в некоторых реализациях Java такой вызов не сработает, поэтому не пренебрегайте вызовом newInstance(), чтобы не оказалось, что вы претворяете в жизнь лозунг Java «Разрабатываете один раз – отлаживаете повсюду» (write once, debug everywhere).[1]

Некоторые драйверы JDBC (в том числе и MySQL Connector/J) позволяют указать в конце URL такие параметры, как имя пользователя и пароль. В этом случае вы опускаете второй и третий аргумент при вызове getConnection(). Если использовать такой формат URL, то код, устанавливающий соединение в рассматриваемой программе, мог бы выглядеть так:

```
// connect using username and password included in URL
Connection conn = null;
String url = "jdbc:mysql://localhost/cookbook?user=cbuser&password=cbpass";

try
{
    Class.forName ("com.mysql.jdbc.Driver").newInstance ();
    conn = DriverManager.getConnection (url);
    System.out.println ("Connected");
}
```

Параметры user и password должны быть разделены символом &, а не ;.

Дополнительные параметры соединения

При соединении с нелокальным хостом можно явно указать номер порта, добавив :*номер_порта* к имени хоста для URL соединения:

```
String url = "jdbc:mysql://mysql.snake.net:3307/cookbook";
```

[1] Пародия на лозунг фирмы Sun «Write once, run anywhere» («Разрабатываете один раз – используете повсюду»). – *Примеч. перев.*

Что касается соединений с локальным хостом (*localhost*), не существует опции для указания путевого имени сокета домена UNIX, по крайней мере, в MySQL Connector/J. Может быть, в других драйверах MySQL JDBC это возможно. Тем, кто использует такие драйверы, следует обратиться к соответствующей документации.

2.2. Контроль ошибок

Задача

Что-то в вашей программе пошло не так, и вы не знаете, что именно.

Решение

Каждый сталкивался с проблемами на пути к получению корректно работающих программ. Но если вы не упреждаете возможные проблемы, вводя контроль ошибок, то еще более усложняете себе жизнь. Добавьте в программу немного кода для отслеживания ошибок, и программа поможет вам понять, где и что случилось.

Обсуждение

Вы уже знаете, как установить соединение с сервером MySQL. Также весьма полезно знать, как осуществлять проверку на наличие ошибок и как извлекать из API информацию об ошибках, относящихся к MySQL. Об этом мы и поговорим. Когда возникает ошибка, MySQL выводит ее числовой код и соответствующее текстовое сообщение. Рецепты данного раздела рассказывают, как получить доступ к такой информации. Вероятно, вам уже хочется узнать, как делать более интересные вещи (например, запускать запросы и получать их результаты), но контроль и обнаружение ошибок – это задача первостепенной важности. Программы часто не работают, особенно на стадии разработки, и если вы не будете знать, как определить, где произошел сбой, придется действовать вслепую.

Примеры программ раздела показывают, как осуществлять контроль ошибок, но если ваша учетная запись MySQL задана корректно, они будут работать без каких бы то ни было проблем. Поэтому вам, возможно, придется несколько изменить примеры, спровоцировав появление ошибки, которая инициирует исполнение соответствующего обработчика ошибок (handler). Например, можно изменить вызов установки соединения, передав в него неправильный пароль. Так вы поймете, как ведет себя программа при возникновении ошибки.

Независимо от конкретного используемого API вы можете применить такое общедоступное средство отладки, как проверка журнала запросов (query log) MySQL, чтобы посмотреть, какие запросы обрабатываются сервером в настоящий момент. (Необходимо, чтобы был включен режим протоколирования запросов и чтобы вы имели доступ к журналу, расположенному на хосте сервера MySQL.) Журнал может показать вам, что запрос плохо построен, и по-

дать идею о том, почему программа не формирует корректную строку запроса. Если вы запускаете сценарий от имени веб-сервера и он не исполняется, проверьте журнал ошибок (error log) сервера.

Perl

Модуль DBI предоставляет два атрибута, которые контролируют ситуацию в случае, если вызов метода DBI не исполняется:

- `PrintError` – если установлен, то DBI выводит сообщение об ошибке посредством `warn()`.
- `RaiseError` – если установлен, то DBI выводит сообщение об ошибке, используя `die()`, при этом исполнение сценария прекращается.

По умолчанию атрибут `PrintError` включен, а `RaiseError` выключен, так что сценарий продолжает исполняться после вывода сообщения об ошибке (если таковые имеются). Один или оба атрибута могут быть указаны при вызове `connect()`. Установка атрибута в 1 или 0 соответственно включает или выключает его. Чтобы задать один или оба аргумента, передайте их в ссылке на хеш как четвертый аргумент вызова `connect()`. (Формат будет кратко описан ниже.)

Приведенный ниже код использует для атрибутов обработки ошибок установки по умолчанию. В результате, если вызов `connect()` не выполнится корректно, будет выведено предупреждение, но сценарий продолжит исполняться:

```
my $dbh = DBI->connect ($dsn, "cbuser", "cbpass");
```

Однако поскольку на самом деле вряд ли можно что-то сделать, когда не установлено соединение с сервером, разумнее выйти из сценария после вывода сообщения об ошибке:

```
my $dbh = DBI->connect ($dsn, "cbuser", "cbpass") or exit;
```

Для вывода собственных сообщений об ошибках оставьте атрибут `RaiseError` выключенным и выключите `PrintError`. Затем самостоятельно проверьте результаты вызовов методов DBI. Если метод исполняется некорректно, переменные `$DBI::err` и `$DBI::errstr` содержат числовой код ошибки MySQL и строку описания ошибки соответственно:

```
my $dbh = DBI->connect ($dsn, "cbuser", "cbpass", {PrintError => 0})
                or die "Connection error: $DBI::errstr ($DBI::err)\n";
```

Если ошибок не возникает, `$DBI::err` равна 0 или `undef`, а `$DBI::errstr` – пустой строке или `undef`.

Осуществляя контроль ошибок, вы должны обращаться к этим переменным сразу после вызова DBI-метода, устанавливающего их значения. Если перед обращением к переменным вы вызовете другой метод, значения переменных будут переназначены.

При выводе собственных сообщений об ошибках становится неудобно использовать значения атрибутов по умолчанию (атрибут `PrintError` включен,

RaiseError выключен). Получается, что сначала DBI автоматически выводит сообщение, а затем сценарий выводит еще и свое сообщение, что в лучшем случае просто избыточно, а в худшем – может запутать человека, работающего со сценарием.

Если вы включаете RaiseError, то можете вызывать методы DBI, не проверяя возвращаемые ими значения на специальные значения, сигнализирующие о возникновении ошибок. Если метод не выполняется, DBI выводит ошибку и завершает исполнение сценария. Если метод что-то возвращает, считаем, что он отработал корректно. Такой подход наиболее прост для создателей сценариев: пусть весь контроль ошибок осуществляет DBI! Но если включены оба атрибута, PrintError и RaiseError, DBI может последовательно вызывать warn() и die(), и сообщения об ошибках будут напечатаны дважды. Чтобы избежать повторений, лучше отключать атрибут PrintError, когда включен RaiseError. В этой книге обычно используется именно такой подход:

```
my $dbh = DBI->connect ($dsn, "cbuser", "cbpass",
                        {PrintError => 0, RaiseError => 1});
```

Если вам не нравится ни идеология «все или ничего», реализуемая при включении RaiseError, когда все происходит автоматически, ни необходимость самостоятельно контролировать наличие ошибок, вы можете использовать комбинированный подход. Отдельные дескрипторы имеют атрибуты PrintError и RaiseError, которые можно включать и выключать избирательно. Например, вы можете глобально включить RaiseError при вызове connect(), а затем выборочно запретить его на уровне дескрипторов. Предположим, что у вас есть сценарий, который считывает имя пользователя и пароль из аргументов командной строки, а затем в цикле выполняет вводимые пользователем запросы. Вероятно, в такой ситуации вы захотите, чтобы DBI завершал работу и автоматически выводил сообщение об ошибке, поскольку если пользователь не ввел правильные имя и пароль, он не сможет продолжить работу. С другой стороны, в случае успешно установленного соединения вы уже не захотите, чтобы сценарий прекращал работу только потому, что пользователь ввел синтаксически некорректный запрос. Удобнее было бы, если бы сценарий отлавливал ошибки, выводил соответствующее сообщение, а затем возвращался к ожиданию следующего запроса. Ниже приведен код, показывающий, как все это можно реализовать (использованный в примере метод do() выполняет запрос и возвращает undef в случае ошибки):

```
my $user_name = shift (@ARGV);
my $password = shift (@ARGV);
my $dbh = DBI->connect ($dsn, $user_name, $password,
                        {PrintError => 0, RaiseError => 1});
$dbh->{RaiseError} = 0; # отменить автоматическое завершение в случае ошибки
print "Enter queries to be executed, one per line; terminate with Control-D\n";
while (<>)              # читать и выполнять запросы
{
    $dbh->do ($_) or warn "Query failed: $DBI::errstr ($DBI::err)\en";
}
$dbh->{RaiseError} = 1; # заново включить автоматическое завершение в случае ошибки
```

Если включен атрибут RaiseError, можно выявлять ошибки без завершения программы, исполняя код блока eval. Если ошибка встречается внутри блока, eval возвращает сообщение об ошибке, записываемое в переменную $@. Обычно eval используется примерно так:

```
eval
{
    # здесь находятся предложения, в которых возможны ошибки...
};
if ($@)
{
    print "An error occurred: $@\n";
}
```

Подобная методика является общепринятой, например для реализации транзакций (см. главу 15). Отличия применения RaiseError в сочетании с eval от использования просто RaiseError заключаются в следующем:

- В случае ошибки осуществляется выход только из блока eval, но не из сценария.
- К выходу из блока eval приводит любая ошибка, в то время как RaiseError реагирует только на ошибки, относящиеся к DBI.

Если вы используете eval с включенным атрибутом RaiseError, убедитесь в том, что атрибут PrintError выключен. В противном случае в некоторых версиях DBI может случиться так, что при возникновении ошибки будет просто вызываться warn(), а ожидаемого выхода из блока eval не произойдет.

Для получения полезной информации об исполнении сценария (в дополнение к использованию атрибутов RaiseError и PrintError) вы можете включить механизм трассировки DBI. Вызовите метод trace() с аргументом, указывающим необходимый уровень трассировки. Уровни 1–9 включают трассировку с возрастающей подробностью вывода, а уровень 0 отключает ее:

```
DBI->trace (1);      # включить трассировку, минимальный вывод
DBI->trace (3);      # повысить уровень трассировки
DBI->trace (0);      # выключить трассировку
```

У дескрипторов отдельных баз данных и предложений тоже имеются методы trace(). То есть при желании вы можете отслеживать один-единственный дескриптор.

Вывод трассировки обычно попадает на терминал (или, если речь идет о веб-сценарии, то в журнал ошибок веб-сервера). Вы можете направить вывод трассировки в файл, указав его имя в качестве второго аргумента:

```
DBI->trace (1, "/tmp/trace.out");
```

Если такой файл уже существует, вывод трассировки будет дописан в его конец; содержимое файла не стирается. Включайте трассировку на время разработки сценария, но не забудьте отключить ее, когда сценарий будет готов к работе. Иначе к вашему глубокому сожалению окажется, что трассировочный файл стал слишком большим (или, в самом худшем случае, файловая система вдруг переполнится, а вы даже не будете знать, что произошло!).

PHP

В PHP большинство функций, которые могут выполниться успешно или нет, сообщают о том, что происходит, посредством возвращаемого значения. Вам остается проверить значение и сделать выводы. Некоторые функции в случае сбоя дополнительно выводят предупреждение (например, так ведут себя `mysql_connect()` и `mysql_select_db()`). Иногда автоматический вывод предупреждений весьма полезен, но если задачей вашего сценария является формирование веб-страницы (что вполне вероятно), вы вряд ли захотите, чтобы в середине страницы появилось сообщение об ошибке. Подавить вывод предупреждений можно двумя способами. Для того чтобы запретить вывод предупреждений отдельной функции, поставьте перед ее именем оператор подавления предупреждений `@`. Затем проверьте возвращаемое значение и обрабатывайте ошибки самостоятельно. Именно такой подход использовался в предыдущем разделе при соединении с сервером MySQL для функции *connect.php*, которая выводит собственные сообщения:

```
if (!($conn_id = @mysql_connect ("localhost", "cbuser", "cbpass")))
    die ("Cannot connect to server\n");
print ("Connected\n");
if (!@mysql_select_db ("cookbook", $conn_id))
    die ("Cannot select database\n");
```

Кроме того, есть возможность отключить предупреждения глобально, используя функцию `error_reporting()` для установки уровня ошибок PHP в ноль:

```
error_reporting(0);
```

Однако не забывайте о том, что отключив предупреждения, вы не получите уведомлений о проблемах, возникших со сценарием, о которых вам следовало бы знать, например об ошибках синтаксического анализа.

Чтобы получить информацию о неудавшихся операциях, связанных с MySQL, используйте функции `mysql_errno()` и `mysql_error()`, которые возвращают код ошибки и строку ее описания соответственно. Каждая из функций принимает как необязательный аргумент идентификатор соединения. Если не указать идентификатор, обе функции будут считать, что вас интересует информация об ошибках соединения, открытого последним. Однако в версиях, предшествующих PHP 4.0.6, обе функции требовали, чтобы соединение уже *было установлено*. В ранних версиях PHP такое требование делало бесполезными функции ошибок для программ установки соединения. (Если происходит сбой в работе `mysql_connect()` или `mysql_pconnect()`, `mysql_errno()` и `mysql_error()` возвращают 0 и пустую строку, как если бы никаких ошибок не было.) Чтобы обойти это ограничение, можно использовать глобальную переменную PHP `$php_errormsg`, как показано в следующем примере. В коде выводятся сообщения об ошибках и для неудачных попыток соединения, и для ошибок, возникших после успешной установки соединения. Код пробует применить `mysql_errno()` и `mysql_error()` для проблем с соединением, проверяя, возвращают ли они полезную информацию. Если нет, то используется `$php_errormsg`:

```
if (!($conn_id = @mysql_connect ("localhost", "cbuser", "cbpass")))
{
    # Если mysql_errno()/mysql_error() работают для неудавшихся соединений,
    # используйте их (вызовите без аргумента). Иначе используйте $php_errormsg.
    if (mysql_errno ())
    {
        die (sprintf ("Cannot connect to server: %s (%d)\n",
                htmlspecialchars (mysql_error ()),
                mysql_errno ()));
    }
    else
    {
        die ("Cannot connect to server: "
                . htmlspecialchars ($php_errormsg) . "\n");
    }
}
print ("Connected\n");
if (!@mysql_select_db ("cookbook", $conn_id))
{
    die (sprintf ("Cannot select database: %s (%d)\n",
            htmlspecialchars (mysql_error ($conn_id)),
            mysql_errno ($conn_id)));
}
```

Функция `htmlspecialchars()` экранирует символы `<`, `>` и `&`, чтобы они корректно отображались на веб-страницах. Такая возможность полезна при отображении сообщений об ошибках, ведь вы не знаете, какие именно символы они будут содержать.

Для того чтобы использовать `$php_errormsg`, необходимо разблокировать переменную `track_errors` в инициализационном файле PHP. В моей системе это файл */usr/local/lib/php.ini*. Найдите такой файл у себя и убедитесь в том, что строка `track_errors` выглядит так:

```
track_errors = On;
```

Если вы используете PHP как модуль Apache и изменяете строку `track_errors`, необходимо будет перезапустить Apache для того, чтобы изменение вступило в силу.

Python

Программы, написанные на языке Python, оповещают об ошибках, порождая исключения, а обрабатывают ошибки, улавливая исключения в блоке `except`. Чтобы получить информацию об ошибке, относящуюся к MySQL, назначьте класс исключений и создайте переменную для получения информации. Рассмотрим пример:

```
try:
    conn = MySQLdb.connect (db = "cookbook",
                            host = "localhost",
                            user = "cbuser",
                            passwd = "cbpass")
```

```
    print "Connected"
except MySQLdb.Error, e:
    print "Cannot connect to server"
    print "Error code:", e.args[0]
    print "Error message:", e.args[1]
    sys.exit (1)
```

Если возникает исключение, первый и второй элементы `e.args` принимают значения числового кода и описания ошибки соответственно. Обратите внимание на то, что доступ к классу `Error` осуществляется через имя модуля драйвера MySQLdb.

Java

Java-программы обрабатывают ошибки, перехватывая исключения. Если вы хотите минимизировать свою работу, можете просто выводить содержимое стека для уведомления пользователя о месте возникновения проблемы:

```
catch (Exception e)
{
    e.printStackTrace ();
}
```

Запись трассировки стека указывает, где произошла ошибка, но не то, в чем она заключается. Такая информация может быть полезной только для того, кто написал программу. Чтобы сделать данные более значимыми, можно выводить сообщение об ошибке и код, которому сопоставлено исключение:

- Все объекты `Exception` поддерживают метод `getMessage()`. Методы JDBC могут порождать исключения, используя объекты `SQLException`, которые похожи на объекты `Exception`, но дополнительно поддерживают методы `getErrorCode()` и `getSQLState()`.
- Для ошибок MySQL методы `getErrorCode()` и `getMessage()` возвращают числовой код ошибки и строку описания ошибки.
- Метод `getSQLState()` возвращает строку, которая содержит значения ошибок, определенные согласно спецификации XOPEN SQL (которая может быть или не быть полезной для вас).
- Вы также можете получать информацию о нефатальных ошибках, которые некоторые методы генерируют посредством объектов `SQLWarning`. `SQLWarning` — это подкласс `SQLException`, но предупреждения накапливаются в списке, а не выводятся сразу же, так что программа не прерывается, и вы можете спокойно их выводить.

Рассмотрим программу *Error.java*, которая показывает, как получить доступ к сообщениям об ошибках, выводя всю возможную информацию. Она пытается установить соединение с сервером MySQL и выводит информацию об исключении, если попытка не удается. Затем она выдает запрос и выводит исключение и предупреждение, если запрос не исполняется:

```
// Error.java - пример обработки ошибок MySQL

import java.sql.*;
```

```
public class Error
{
    public static void main (String[] args)
    {
        Connection conn = null;
        String url = "jdbc:mysql://localhost/cookbook";
        String userName = "cbuser";
        String password = "cbpass";
        try
        {
            Class.forName ("com.mysql.jdbc.Driver").newInstance ();
            conn = DriverManager.getConnection (url, userName, password);
            System.out.println ("Connected");
            tryQuery (conn);                // запустить запрос
        }
        catch (Exception e)
        {
            System.err.println ("Cannot connect to server");
            System.err.println (e);
            if (e instanceof SQLException)  // это исключение JDBC?
            {
                // выводить общее сообщение и специальное сообщение базы данных
                // (обратите внимание на то, как e преобразуется из Exception
                // в SQLException для получения доступа
                // к собственным методам SQLException)
                System.err.println ("SQLException: " + e.getMessage ());
                System.err.println ("SQLState: "
                                + ((SQLException) e).getSQLState ());
                System.err.println ("VendorCode: "
                                + ((SQLException) e).getErrorCode ());
            }
        }
        finally
        {
            if (conn != null)
            {
                try
                {
                    conn.close ();
                    System.out.println ("Disconnected");
                }
                catch (SQLException e)
                {
                    // выводить общее сообщение и любые
                    // специальные сообщения базы данных
                    System.err.println ("SQLException: " + e.getMessage ());
                    System.err.println ("SQLState: " + e.getSQLState ());
                    System.err.println ("VendorCode: " + e.getErrorCode ());
                }
            }
        }
    }
```

```
    public static void tryQuery (Connection conn)
    {
        Statement s = null;

        try
        {
            // выдать простой запрос
            s = conn.createStatement ();
            s.execute ("USE cookbook");
            s.close ();

            // вывести накопившиеся предупреждения
            SQLWarning w = conn.getWarnings ();
            while (w != null)
            {
                System.err.println ("SQLWarning: " + w.getMessage ());
                System.err.println ("SQLState: " + w.getSQLState ());
                System.err.println ("VendorCode: " + w.getErrorCode ());
                w = w.getNextWarning ();
            }
        }
        catch (SQLException e)
        {
            // выводить общее сообщение и специальное сообщение базы данных
            System.err.println ("SQLException: " + e.getMessage ());
            System.err.println ("SQLState: " + e.getSQLState ());
            System.err.println ("VendorCode: " + e.getErrorCode ());
        }
    }
}
```

2.3. Создание библиотечных файлов

Задача

Вы обнаружили, что в нескольких программах приходится писать один и тот же код для реализации часто встречающихся операций.

Решение

Поместите функции, выполняющие такие операции, в библиотечный файл. Тогда вам придется написать этот код всего лишь раз.

Обсуждение

В разделе описано, как поместить код общих операций в библиотечные файлы. Фактически инкапсуляция (или модуляризация) – это скорее не «рецепт», а методика программирования. Ее основное преимущество заключается в том, что не приходится повторять код в каждой новой программе – вы просто вызываете функцию, которая находится в библиотеке. Например, если поместить в библиотечную функцию код установления соединения

с базой данных cookbook, то не придется переписывать все параметры выполнения этой операции. Просто вызовите функцию из своей программы, и вы уже подключены к базе данных!

Конечно, установление соединения – это не единственная операция, которую можно инкапсулировать. Далее в книге будут создаваться и помещаться в библиотечные файлы и другие полезные функции. Все такие файлы, в том числе приведенные в этом разделе, присутствуют в каталоге *lib* дистрибутива recipes. В процессе разработки программ вы наверняка замечали, что некоторые операции встречаются довольно часто; они и будут кандидатами на включение в библиотеку. Методика, представленная в разделе, поможет вам при создании собственных библиотечных файлов.

Достоинства библиотечных файлов не ограничиваются упрощением написания программ. Они также способствуют переносимости. Например, если вы указываете параметры соединения в каждой программе, подключающейся к серверу MySQL, то при их переносе на другую машину, работающую с другими параметрами, вам придется внести изменения во все такие программы. Если же программы соединяются с базой данных посредством вызова библиотечной функции, необходимо будет изменить только одну эту функцию, а не все использующие ее программы.

Инкапсуляция кода также поможет в обеспечении безопасности. Если вы создаете персональный библиотечный файл, доступный для чтения лишь вам лично, только запущенные вами сценарии смогут выполнять команды из этого файла. Предположим, что какие-то ваши сценарии помещены в дерево документов веб-сервера. Правильно настроенный сервер исполняет сценарии и отправляет их вывод удаленным клиентам. Но если конфигурация сервера будет нарушена, ваши сценарии могут быть отосланы клиентам в виде открытого текста, содержащего ваше имя пользователя и пароль для работы с сервером MySQL (и, к сожалению, вы можете обнаружить это слишком поздно). А если код установления соединения с сервером MySQL помещен в библиотечный файл, находящийся вне дерева документов, то эти параметры не будут доступны клиентам. (Но учтите, что если вы разрешите чтение библиотечного файла веб-серверу, все будет уже не так безопасно, если вы используете сервер совместно с другими разработчиками. Любой из них может написать веб-сценарий, который будет читать и отображать ваш библиотечный файл, так как по умолчанию сценарий будет запускаться с привилегиями веб-сервера, следовательно, получит доступ к библиотеке.)

Приведенные далее примеры показывают, как написать для каждого API библиотечный файл, содержащий функцию соединения с базой данных cookbook, хранящейся на сервере MySQL. Функции Perl, PHP и Python написаны так, что они возвращают значение соответствующего типа (дескриптор базы данных, идентификатор соединения или объект соединения) или же завершаются с сообщением об ошибке, если соединение установить не удается. (Для контроля ошибок функции используют приемы, описанные в рецепте 2.2.) Java-функция установления соединения реализует несколько другой подход. Если соединение успешно установлено, она возвращает объект

соединения, иначе порождает исключение, которое может обрабатываться вызывающей программой. Со своей стороны, чтобы помочь в обработке таких исключений, библиотека содержит вспомогательные функции, которые возвращают или выводят сообщение об ошибке, содержащее информацию, возвращенную MySQL.

Библиотеки сами по себе бесполезны, и способ использования каждой из них показан при помощи небольшой программы, являющейся средством поддержки тестирования. Вы можете использовать любую из этих программ как основу для собственных разработок: создайте копию файла и вставьте ваш собственный код между вызовами соединения с базой и отключения от нее.

Создание библиотечного файла – это не только выбор того, что следует поместить в файл, необходимо решить еще ряд дополнительных вопросов. Например, где разместить файл так, чтобы он был доступен вашим программам и (в многопользовательских системах, таких как UNIX) как задать права доступа так, чтобы содержимое файла не было доступно тем, кто не должен его видеть. Особенности создания библиотечного файла и настройки языкового процессора для работы с ним определяются конкретным используемым API; о них будет рассказано далее в соответствующих разделах. Напротив, установка права собственности на файл и режима доступа к нему являются общим вопросами, ответы на которые не зависят от используемого языка (по крайней мере для тех, кто работает в UNIX):

- Если библиотечный файл – ваш собственный и содержит код, предназначенный только для личного пользования, его можно поместить под вашей учетной записью и сделать доступным только вам. Если вы являетесь владельцем библиотечного файла *mylib*, то можете сделать его персональным (private) файлом следующим образом:

```
% chmod 600 mylib
```

- Если библиотечный файл должен использоваться только веб-сервером, можно поместить его в библиотечный каталог сервера и сделать файл принадлежащим и доступным только для учетной записи сервера. Для выполнения такой операции может потребоваться подключиться к серверу пользователем `root`. Например, если веб-сервер запускается как `wwwusr`, следующие команды сделают файл персональным файлом этого пользователя:

```
# chown wwwusr mylib
# chmod 600 mylib
```

- Если библиотечный файл предназначен для общего использования, можно поместить его в каталог, который ваш язык программирования автоматически просматривает в поиске библиотек (большинство языков просматривает с этой целью некоторый определенный набор каталогов). Для помещения файла в такой каталог может потребоваться подключиться к серверу как пользователь `root`. Затем сделайте файл доступным для чтения всем пользователям:

```
# chmod 444 mylib
```

В рассматриваемой программе предполагается, что библиотечные файлы помещаются в каталог, не просматриваемый трансляторами по умолчанию (это сделано для того, чтобы показать, как можно изменить алгоритм поиска так, чтобы просматривался выбранный каталог). Многие из программ, приведенных в этой книге, работают в среде Web, поэтому в примерах для размещения библиотечных файлов используются каталоги *perl*, *php*, *python* и *java*, находящиеся внутри */usr/local/apache/lib*. Если вы хотите расположить файлы в каком-то другом каталоге, просто соответственно измените в программах путевые имена. Кроме того можно воспользоваться возможностью, предоставляемой большинством языков программирования: укажите, в каких каталогах следует искать библиотечные файлы, посредством переменной окружения или конфигурации. Такие переменные для API, рассматриваемых в книге, приведены в табл. 2.1.

Таблица 2.1. Переменные для хранения путей к каталогам библиотечных файлов

Язык	Имя переменной	Тип переменной
Perl	PERL5LIB	Переменная окружения
PHP	include_path	Переменная конфигурации
Python	PYTHONPATH	Переменная окружения
Java	CLASSPATH	Переменная окружения

Значение каждой переменной – это каталог или набор каталогов. Например, если при работе в UNIX поместить библиотечные файлы Perl в каталог */u/paul/lib/perl*, можно задать в файле *.login* следующее значение для переменной окружения PERL5LIB в оболочке *tcsh*:

```
setenv PERL5LIB /u/paul/lib/perl
```

В Windows, если библиотечные файлы Perl находятся в каталоге *D:\lib\perl*, можно установить PERL5LIB в файле *AUTOEXEC.BAT*:

```
SET PERL5LIB=D:\lib\perl
```

Установка переменной в обоих случаях указывает Perl, что в поиске библиотечных файлов нужно просматривать перечисленные каталоги в дополнение к каталогам по умолчанию. Другие переменные окружения (PYTHONPATH и CLASSPATH) задаются аналогично. Подробная информация об установке переменных окружения приведена в рецепте 1.8.

В PHP путь поиска определяется значением переменной include_path инициализационного файла PHP (который обычно называется *php.ini* или *php3.ini*). В моей системе путевое имя файла – */usr/local/lib/php.ini*; в Windows файл, вероятно, будет находиться в системном каталоге Windows или внутри основного каталога установки PHP. Значение include_path задается такой строкой:

```
include_path = "значение"
```

Формат указания значения повторяет формат переменных окружения, содержащих имена каталогов. То есть это список имен каталогов, разделенных двоеточиями в UNIX и точками с запятой в Windows. Например, если вы хотите, чтобы PHP искал включаемые файлы в текущем каталоге и в каталоге *lib/php*, находящемся внутри корневого каталога веб-сервера */usr/local/apache*, переменная `include_path` должна быть установлена так (в UNIX):

```
include_path = ".:/usr/local/apache/lib/php"
```

Если вы изменяете инициализационный файл, и PHP запускается как модуль Apache, то для вступления изменений в силу необходимо перезапустить Apache.

Теперь давайте создадим по библиотеке для каждого API. Каждый раздел иллюстрирует создание библиотечного файла, затем описывает использование этот библиотеки в программах.

Perl

В Perl библиотечные файлы называются модулями и обычно имеют расширение *.pm* («Perl module»). Рассмотрим в качестве примера файл *Cookbook.pm*, реализующий модуль `Cookbook`. (В Perl для базового имени модуля принято использовать значение идентификатора в строке `package` файла.)

```
package Cookbook;
# Cookbook.pm - библиотечный файл, содержащий функцию соединения с MySQL

use strict;
use DBI;

# Устанавливает соединение с базой данных cookbook, возвращает дескриптор базы
# данных. Если не удалось установить соединение, завершается с выдачей сообщения.

sub connect
{
my $db_name = "cookbook";
my $host_name = "localhost";
my $user_name = "cbuser";
my $password = "cbpass";
my $dsn = "DBI:mysql:host=$host_name;database=$db_name";

    return (DBI->connect ($dsn, $user_name, $password,
                          { PrintError => 0, RaiseError => 1}));
}

1;  # возвращает значение true
```

Модуль инкапсулирует код установки соединения с сервером MySQL в функцию `connect()`, а идентификатор `package` порождает для модуля пространство имен `Cookbook`, так что функция `connect()` вызывается с использованием имени модуля:

```
$dbh = Cookbook::connect();
```

Последняя строка файла модуля – это предложение, которое заведомо вычисляется как «истина». Это необходимо, поскольку если модуль не возвращает

«истину», Perl считает, что возникли какие-то проблемы, и завершает работу после чтения такого модуля.

Perl определяет местоположение файлов модулей, просматривая каталоги, имена которых приведены в массиве `@INC`. Этот массив содержит список каталогов по умолчанию. Чтобы вывести список каталогов для просмотра по умолчанию в вашей системе, вызовите Perl в командной строке следующим образом:

```
% perl -V
```

В конце вывода команды приведены каталоги, перечисленные в массиве `@INC`. Если модуль находится в одном из таких каталогов, сценарии найдут его автоматически. Если же модуль расположен в каком-то другом каталоге, необходимо сообщить сценариям, где следует его искать, используя предложение `use lib`. Например, если файл модуля *Cookbook.pm* находится в каталоге */usr/local/apache/lib/perl*, вы можете написать в целях тестирования сценарий *harness.pl*, использующий этот модуль:

```
#! /usr/bin/perl -w
# harness.pl – средство тестирования для библиотеки Cookbook.pm

use strict;
use lib qw(/usr/local/apache/lib/perl);
use Cookbook;

my $dbh = Cookbook::connect ();
print "Connected\n";
$dbh->disconnect ();
print "Disconnected\n";

exit (0);
```

Обратите внимание, что *harness.pl* не содержит предложение `use DBI`. В нем нет необходимости, так как модуль `Cookbook` сам импортирует модуль DBI, поэтому любой сценарий, использующий `Cookbook`, также получает DBI.

Есть и другой способ указать, где Perl должен искать файлы модулей (в дополнение к каталогам, просматриваемым по умолчанию), – установить переменную окружения `PERL5LIB`. Если вы используете такую возможность, то вашим сценариям не понадобится предложение `use lib`. (Есть и недостаток – каждый пользователь, запускающий сценарии, которые используют модуль `Cookbook`, должен будет установить `PERL5LIB`.)

PHP

PHP включает в себя предложение `include`, которое позволяет прочитать содержимое файла и включить его в текущий сценарий. Это естественный механизм формирования библиотеки: библиотечный код помещается во включаемый файл, файл помещается в каталог, упомянутый в пути поиска PHP, и включается во все необходимые сценарии. Например, если вы создаете включаемый файл *Cookbook.php*, все нуждающиеся в нем сценарии могут использовать такое предложение:

```
include "Cookbook.php";
```

Содержимое включаемых файлов PHP представляет собой обычный сценарий. Создадим такой файл, *Cookbook.php*, содержащий функцию `cookbook_connect()`**:**

```
<?php
# Cookbook.php - библиотечный файл, содержащий функцию соединения с MySQL

# Устанавливает соединение с базой данных cookbook, возвращает идентификатор
# соединения. Если установить соединение не удалось, завершается с выводом сообщения.

function cookbook_connect ()
{
    $db_name = "cookbook";
    $host_name = "localhost";
    $user_name = "cbuser";
    $password = "cbpass";

    $conn_id = @mysql_connect ($host_name, $user_name, $password);
    if (!$conn_id)
    {
        # Если mysql_errno()/mysql_error() работают для неуспешных соединений,
        # используйте их (вызовите без аргумента). В противном случае
        # используйте $php_errormsg.
        if (mysql_errno ())
        {
            die (sprintf ("Cannot connect to server: %s (%d)\n",
                    htmlspecialchars (mysql_error ()),
                    mysql_errno ()));
        }
        else
        {
            die ("Cannot connect to server: "
                    . htmlspecialchars ($php_errormsg) . "\n");
        }
    }
    if (!@mysql_select_db ($db_name))
    {
        die (sprintf ("Cannot select database: %s (%d)\n",
                htmlspecialchars (mysql_error ($conn_id)),
                mysql_errno ($conn_id)));
    }
    return ($conn_id);
}

?>
```

В большинстве приведенных в книге примеров на PHP не показаны теги `<?php` **и** `?>`**; здесь же они приведены для того, чтобы подчеркнуть тот факт, что весь код PHP во включаемых файлах должен быть заключен в эти теги. Когда интерпретатор PHP приступает к разбору включаемого файла, у него нет никакого представления о его содержимом (ведь вы можете включить в проект файл, содержащий только HTML). Поэтому необходимо использовать теги** `<?php` **и** `?>`**,**

чтобы явно указать, какие части включаемого файла должны рассматриваться как PHP-код, а не HTML, как вы это делаете и в основном сценарии.

Будем считать, что файл *Cookbook.php* расположен в каталоге, указанном в пути поиска PHP (определяемом переменной `include_path` в инициализационном файле PHP), тогда он может использоваться из сценария *harness.php*, с помощью которого мы проводим тестирование:

```
<?php
# harness.php - средство тестирования библиотеки Cookbook.php

include "Cookbook.php";
$conn_id = cookbook_connect ();
print ("Connected\n");
mysql_close ($conn_id);
print ("Disconnected\n");

?>
```

Если у вас нет прав на изменение инициализационного файла PHP, укажите полное путевое имя включаемого файла, например:

```
include "/usr/local/apache/lib/php/Cookbook.php";
```

В PHP имеется также предложение `require`, которое похоже на `include`, но отличается от него тем, что PHP читает файл, даже если `require` встречается внутри никогда не исполняющейся управляющей структуры (например, внутри блока `if`, условие которого всегда ложно). В PHP 4 добавлены предложения `include_once` и `require_once`. Они аналогичны `include` и `require`; важное же отличие заключается в том, что если файл уже был прочитан, повторно его содержимое не обрабатывается. Такой возможностью удобно пользоваться во избежание проблем с повторными объявлениями переменных, которые могут возникнуть, если библиотечные файлы включают в себя другие библиотечные файлы.

Чтобы имитировать однократное включение в PHP 3, можно сопоставить библиотеке уникальный идентификатор и обрабатывать содержимое файла, только если идентификатор еще не определен. Например, библиотечный файл *MyLibrary.php* мог бы быть устроен следующим образом:

```
<?php
# MyLibrary.php - показывает, как имитировать однократное включение в PHP 3

# Проверить, определен ли идентификатор, сопоставленный файлу.
# Если нет - определить его и обработать содержимое файла.
# Иначе файл уже прочитан, пропустить остаток содержимого.

if (!defined ("_MYLIBRARY_PHP_"))
{
    define ("_MYLIBRARY_PHP_", 1);

    # ... поместить оставшуюся часть библиотеки сюда ...

}   # конец _MYLIBRARY_PHP_

?>
```

Куда следует помещать включаемые файлы PHP?

Сценарии PHP часто помещаются в дерево документов веб-сервера, и клиенты могут обращаться к ним напрямую. Что касается библиотечных файлов PHP, я бы рекомендовал не использовать дерево документов, особенно если файлы содержат имена пользователей и пароли (как *Cookbook.php*). Особая осторожность нужна, если включаемые файлы имеют другое расширение, например *.inc*. Если вы поместите файл с таким расширением в дерево документов, он может быть запрошен клиентами и будет показан им как открытый текст. Чтобы подобного не случилось, настройте Apache так, чтобы он воспринимал файлы с расширением *.inc* как PHP-код, который должен обрабатываться интерпретатором PHP, а не отображаться дословно.

Python

Библиотеки Python создаются как модули, а сценарии ссылаются на них при помощи предложений `import` или `from`. Чтобы поместить в функцию код установления соединения с MySQL, можно создать файл модуля *Cookbook.py*:

```
# Cookbook.py - библиотечный файл, содержащий программу,
# устанавливающую соединение с MySQL

import sys
import MySQLdb

# Устанавливает соединение с базой данных cookbook, возвращает объект соединения.
# Если соединение установить не удалось, завершается с выводом сообщения.

def connect ():
    host_name = "localhost"
    db_name = "cookbook"
    user_name = "cbuser"
    password = "cbpass"

    try:
        conn = MySQLdb.connect (db = db_name,
                                host = host_name,
                                user = user_name,
                                passwd = password)
        return conn
    except MySQLdb.Error, e:
        print "Cannot connect to server"
        print "Error code:", e.args[0]
        print "Error message:", e.args[1]
        sys.exit (1)
```

Базовое имя файла определяет имя модуля, поэтому модуль называется `Cookbook`. Для доступа к методам модуля необходимо указывать имя модуля, например, вызовем метод `connect()` модуля `Cookbook`:

```
conn = Cookbook.connect ();
```

Интерпретатор Python просматривает в поиске модулей каталоги, имена которых приведены в переменной sys.path. Аналогично массиву @INC в Perl, в качестве значения переменной sys.path изначально устанавливается некоторый набор каталогов по умолчанию. Чтобы узнать, какие каталоги входят в этот набор в вашей системе, запустите Python в интерактивном режиме и выполните пару команд:

```
% python
>>> import sys
>>> sys.path
```

Если поместить *Cookbook.py* в один из каталогов по умолчанию, вы сможете ссылаться на него из сценария, используя предложение import, и Python будет находить модуль автоматически:

```
import Cookbook
```

Если *Cookbook.py* расположен в каком-то другом каталоге, можно добавить его имя в переменную sys.path. Для этого импортируйте модуль sys и вызовите sys.path.insert(). Далее приведен сценарий, используемый нами для тестирования, *harness.py*, созданный в предположении, что модуль Cookbook размещен в каталоге */usr/local/apache/lib/python*:

```
#! /usr/bin/python
# harness.py - средство тестирования библиотеки Cookbook.py

# Импортирует модуль sys и добавляет каталог в путь поиска
import sys
sys.path.insert (0, "/usr/local/apache/lib/python")
import MySQLdb
import Cookbook

conn = Cookbook.connect ()
print "Connected"
conn.close ()
print "Disconnected"
sys.exit (0)
```

Есть и другой способ сообщить интерпретатору Python, где следует искать файлы модулей, – установить переменную окружения PYTHONPATH. Если эта переменная содержит каталог, в котором находится модуль, сценариям не нужно изменять sys.path.

Можно импортировать отдельные символы модуля, используя предложение from:

```
from Cookbook import connect
```

Тогда функция connect() будет доступна сценарию даже без указания имени модуля, и вы сможете использовать ее так:

```
conn = connect()
```

Java

Библиотечные файлы Java во многом похожи на программы Java:

- Строка `class` исходного файла указывает имя класса.
- Файл должен иметь такое же имя, как класс (с расширением *.java*).
- Вы компилируете файл *.java*, чтобы получить файл *.class*.

Однако в отличие от обычных программных файлов в библиотечных файлах Java нет функции `main()`. Кроме того, файл должен начинаться с идентификатора `package`, который указывает местоположение класса в пространстве имен Java. Существует общепринятое соглашение о том, чтобы начитать идентификаторы `package` с инвертированного имени домена автора кода. Такое соглашение способствует поддержанию уникальности идентификаторов и позволяет избегать конфликтов с классами, написанными другими авторами.[1] Мой домен – *kitebird.com*, так что если я хочу написать библиотечный файл и разместить его в пространстве имен под именем *mcb*, моя библиотека должна начинаться таким предложением `package`:

```
package com.kitebird.mcb;
```

Для обеспечения уникальности имен пакеты Java, приведенные в этой книге, будут размещаться в пространстве имен `com.kitebird.mcb`.

Библиотечный файл *Cookbook.java* определяет класс `Cookbook`, реализующий метод `connect()`, который устанавливает соединение с базой данных `cookbook`. В случае успешного соединения `connect()` возвращает объект `Connection`, в противном случае порождается исключение. Чтобы помочь вызывающей программе обрабатывать ошибки, класс `Cookbook` также определяет такие полезные методы, как `getErrorMessage()` и `printErrorMessage()`, возвращающие сообщение об ошибке в виде строки или выводящие его в *System.err*.

```
// Cookbook.java - библиотечный файл с функцией установления соединения с MySQL

package com.kitebird.mcb;

import java.sql.*;

public class Cookbook
{
    // Устанавливает соединение с базой данных cookbook, возвращая объект
    // соединения. Если не удалось установить соединение, порождается исключение.

    public static Connection connect () throws Exception
    {
        String url = "jdbc:mysql://localhost/cookbook";
        String user = "cbuser";
        String password = "cbpass";
```

[1] Имена доменов переходят от общего к частному, двигаясь справа налево, в то время как пространство имен классов Java движется слева направо от общего к частному. Другими словами, для того чтобы использовать имя домена в качестве префикса имени пакета в пространстве имен класса Java, необходимо перевернуть его.

```
        Class.forName ("com.mysql.jdbc.Driver").newInstance ();
        return (DriverManager.getConnection (url, user, password));
    }

    // Возвращает сообщение об ошибке в виде строки

    public static String getErrorMessage (Exception e)
    {
        StringBuffer s = new StringBuffer ();
        if (e instanceof SQLException)  // JDBC-specific exception?
        {
            // выводит общее сообщение и любые сообщения базы данных
            s.append ("Error message: " + e.getMessage () + "\n");
            s.append ("Error code: " + ((SQLException) e).getErrorCode () + "\n");
        }
        else
        {
            s.append (e + "\n");
        }
        return (s.toString ());
    }

    // Получает сообщение об ошибке и выводит его в System.err

    public static void printErrorMessage (Exception e)
    {
        System.err.println (Cookbook.getErrorMessage (e));
    }
}
```

Методы внутри класса объявляются посредством ключевого слова `static`, поэтому класс используется напрямую – не нужно создавать из него объект и вызывать методы через этот объект.

Для того чтобы использовать файл *Cookbook.java*, скомпилируйте его, получите *Cookbook.class*, затем поместите файл класса в каталог, соответствующий идентификатору пакета. То есть *Cookbook.class* должен быть помещен в каталог с именем *com/kitebird/mcb* (или *com\kitebird\mcb* в Windows), который находится в каком-то из каталогов, указанных в переменной `CLASSPATH`. Например, если вы работаете в UNIX, и `CLASSPATH` содержит */usr/local/apache/lib/java*, то можно поместить *Cookbook.class* в каталог */usr/local/apache/lib/java/com/kitebird/mcb*. (Подробная информация о переменной `CLASSPATH` приведена в рецепте 2.1.)

Чтобы использовать класс `Cookbook` в программе Java, необходимо предварительно импортировать его, затем вызвать метод `Cookbook.connect()`. Рассмотрим сценарий *Harness.java*, показывающий, как это сделать:

```
// Harness.java - средство тестирования для библиотечного класса Cookbook

import java.sql.*;
import com.kitebird.mcb.Cookbook;

public class Harness
{
```

```
    public static void main (String[] args)
    {
        Connection conn = null;
        try
        {
            conn = Cookbook.connect ();
            System.out.println ("Connected");
        }
        catch (Exception e)
        {
            Cookbook.printErrorMessage (e);
            System.exit (1);
        }
        finally
        {
            if (conn != null)
            {
                try
                {
                    conn.close ();
                    System.out.println ("Disconnected");
                }
                catch (Exception e)
                {
                    String err = Cookbook.getErrorMessage (e);
                    System.out.println (err);
                }
            }
        }
    }
}
```

Harness.java также показывает, как использовать функции сообщений об ошибках, входящие в класс `Cookbook`, в случае возникновения исключения, относящегося к MySQL. Функция `printErrorMessage()` принимает объект исключения и использует его для вывода сообщения об ошибке в файл *System.err*. Функция `getErrorMessage()` возвращает сообщение об ошибке в виде строки. Вы можете отобразить сообщение на терминале, записать его в файл или куда-то еще.

2.4. Запуск запросов и извлечение результатов

Задача

Вы хотите, чтобы программа отправила запрос на сервер MySQL и получила результат.

Решение

Одни предложения возвращают только код состояния, другие – результирующее множество (набор строк). Большинство API предоставляет свои функции

для каждого типа предложения; если это относится и к вашему API, используйте соответствующую вашему запросу функцию.

Обсуждение

Этот раздел самый длинный в главе, так как существуют два типа запросов, которые можно выполнять. Одни предложения извлекают информацию из базы данных, а другие вносят изменения в эту информацию. Эти два типа запросов обрабатываются по-разному. Более того, в некоторых API существуют еще и несколько различных функций для запуска запросов, что еще больше все усложняет. Прежде чем перейти к примерам, показывающим, как создавать запросы в каждом из API, рассмотрим таблицу, которая будет использоваться в примерах, поговорим о категориях запросов и наметим стратегию их обработки.

В главе 1 была создана таблица `limbs`, которая затем использовалась в примерах. В этой главе будет использоваться другая таблица, `profile`. Таблица строится как «список общения» («buddy list»), то есть включает тех людей, с которыми хотелось бы поддерживать контакт при работе в сети. Чтобы поддерживать ведение совокупности параметров («профиля», profile) для каждого человека, создадим такую таблицу:

```
CREATE TABLE profile
(
    id      INT UNSIGNED NOT NULL AUTO_INCREMENT,
    name    CHAR(20) NOT NULL,
    birth   DATE,
    color   ENUM('blue','red','green','brown','black','white'),
    foods   SET('lutefisk','burrito','curry','eggroll','fadge','pizza'),
    cats    INT,
    PRIMARY KEY (id)
);
```

Таблица `profile` показывает, что для нас важны такие сведения о человеке, как его имя, возраст, любимый цвет, любимое блюдо, а также количество имеющихся у него кошек, то есть очевидно, что это одна из тех бесполезных таблиц, которые нужны только для примеров.[1] В таблице есть столбец `id`, содержащий уникальные значения, так что одну запись всегда можно отличить от другой, даже если двух ваших товарищей зовут одинаково. Столбцы `id` и `name` не допускают использования `NULL`, они обязательно должны содержать значение. Другим же столбцам разрешено быть `NULL`, так как вполне возможно, что у нас не будет данных о каких-то пристрастиях (`NULL` указывает на то, что значение неизвестно). Обратите внимание: несмотря на то что нас интересует возраст, в таблице нет столбца `age`. Есть лишь столбец `birth` (дата рождения) типа `DATE`. Дело в том, что возраст изменяется, а вот день рожденья – нет. Если хранить возраст, придется обновлять его. Хранить дату

[1] На самом деле не такая уж она и бесполезная. Таблица использует разные типы данных для столбцов, что пригодится в дальнейшем для решения проблем, связанных со столбцами определенных типов.

рождения удобнее, так как она постоянна, и ее в любой момент можно использовать для определения возраста (вычисление возраста рассматривается в рецепте 5.19). Столбец color имеет тип ENUM; цвет может принимать любое значение из перечисленных в списке. Столбец foods имеет тип SET, что позволяет выбирать в качестве его значений любые комбинации членов множества, так что вы можете записать несколько любимых блюд для каждого приятеля.

Чтобы создать таблицу, используйте сценарий *profile.sql* из каталога *tables* дистрибутива recipes. Перейдите в этот каталог, затем выполните такую команду:

```
% mysql cookbook < profile.sql
```

Можно создать таблицу вручную из *mysql* при помощи предложения CREATE TABLE, хотя я рекомендовал бы использовать сценарий, так как он не только создает таблицу, но и загружает в нее некоторые тестовые данные. Вы можете проводить над таблицей различные эксперименты, а затем, запустив сценарий еще раз, вернуть ее в прежнее состояние.[1]

Изначально сценарий *profile.sql* загружает в таблицу profile такое содержимое:

```
mysql> SELECT * FROM profile;
+----+---------+------------+-------+----------------------+------+
| id | name    | birth      | color | foods                | cats |
+----+---------+------------+-------+----------------------+------+
|  1 | Fred    | 1970-04-13 | black | lutefisk,fadge,pizza |    0 |
|  2 | Mort    | 1969-09-30 | white | burrito,curry,eggroll |    3 |
|  3 | Brit    | 1957-12-01 | red   | burrito,curry,pizza  |    1 |
|  4 | Carl    | 1973-11-02 | red   | eggroll,pizza        |    4 |
|  5 | Sean    | 1963-07-04 | blue  | burrito,curry        |    5 |
|  6 | Alan    | 1965-02-14 | red   | curry,fadge          |    1 |
|  7 | Mara    | 1968-09-17 | green | lutefisk,fadge       |    1 |
|  8 | Shepard | 1975-09-02 | black | curry,pizza          |    2 |
|  9 | Dick    | 1952-08-20 | green | lutefisk,fadge       |    0 |
| 10 | Tony    | 1960-05-01 | white | burrito,pizza        |    0 |
+----+---------+------------+-------+----------------------+------+
```

Большинство столбцов таблицы profile разрешают значения NULL, но пока что ни одна из строк множества данных не содержит NULL. Это объясняется тем, что значения NULL несколько усложняют обработку запроса, а нам сложности пока не нужны – отложим их до рецептов 2.7 и 2.8.

Категории предложений SQL

Предложения SQL можно разделить на две большие категории:

- Предложения, которые не возвращают результирующее множество (result set), то есть набор строк. К этой категории относятся INSERT, DELETE и UPDATE.

[1] См. замечание о необходимости возвращения таблицы profile в исходное состояние в самом конце главы.

Обычно такие предложения вносят какие-то изменения в базу данных. Есть и некоторые исключения, например предложение `USE` *имя_базы_данных*, которое изменяет базу данных по умолчанию для сеанса, но не вносит никакие изменения в саму базу данных.

- Предложения, возвращающие результирующее множество, такие как `SELECT`, `SHOW`, `EXPLAIN` и `DESCRIBE`. Обычно я буду называть эту категорию в целом предложениями `SELECT`, но вы должны понимать, что имеются в виду все предложения, возвращающие строки.

Первый этап в обработке запроса заключается в отправке его на сервер MySQL для исполнения. Некоторые API (например, Perl и Java) распознают две вышеназванные категории запросов и используют разные вызовы для их выполнения. Другие (такие как PHP и Python) не различают запросы и имеют один вызов для всех предложений. Однако все API обладают одним общим свойством: для указания конца запроса не требуется никакого специального символа. В признаке конца запроса нет необходимости, потому что конец строки запроса неявно указывает на конец запроса. То есть выдача запросов в API отличается от того, как вы делаете это в *mysql*, где предложения завершаются точкой с запятой или символами `\g`. (В книге я тоже обычно использую в синтаксисе предложений SQL точку с запятой, чтобы конец запроса был более очевидным.)

После того как запрос отправлен на сервер, следует проверить, успешно ли он выполнился. *Не пренебрегайте этим этапом!* Если запрос не выполнен, а вы хотите двигаться дальше, считая, что он отработал, программа не будет работать. Если запрос выполнен, следующий шаг зависит от типа этого запроса. Если запрос не возвращает результирующее множество, делать больше нечего (если только вы не захотите проверить, сколько строк было изменено запросом). Если же запрос возвращает результирующий набор строк, можно выбрать его строки, а затем закрыть.

Теперь мы готовы рассмотреть, как реализовать запросы в каждом из API. Обратите внимание: несмотря на то что сценарии осуществляют необходимый контроль ошибок, для краткости они просто выводят сообщение о наличии ошибки. Для вывода более подробных сообщений воспользуйтесь приемами, предлагаемыми в рецепте 2.2.

Не ставьте себе палки в колеса. Контроль ошибок

Похоже, что необходимость контроля ошибок не так понятна и очевидна, как хотелось бы. Множество сообщений в форумах, тематика которых связана с MySQL, содержат просьбы о помощи со стороны разработчиков, программы которых не работают по непонятным причинам. И в удивительно большом количестве случаев люди не могут разобраться со своими программами потому, что в них *нет* проверки ошибок, так что нет никакой возможности понять, где возникла проблема, и увидеть, в чем она состояла! Помогайте себе сами. Осуществляйте контроль ошибок, чтобы должным образом отреагировать на их появление.

Perl

Модуль DBI Perl реализует два базовых подхода к выполнению запроса в зависимости от того, ожидается ли возврат результирующего множества или нет. Чтобы запустить такой запрос, как INSERT или UPDATE, не возвращающий результат, используйте метод do(). Он выполняет запрос и возвращает количество обработанных строк или undef в случае ошибки. Например, если Фред (Fred) заводит кошку (cat), увеличить соответствующее значение в его столбце cats может следующий запрос:

```
my $count = $dbh->do ("UPDATE profile SET cats = cats+1 WHERE name = 'Fred'");
if ($count)     # если ошибок не возникло, выводится количество строк
{
    $count += 0;
    print "$count rows were updated\n";
}
```

Если запрос выполняется успешно, но не изменяет ни одной строки, do() возвращает специальное значение, строку "0E0" (на самом деле это значение 0 в экспоненциальном представлении). Строка "0E0" может применяться для проверки статуса запроса, так как в булевских терминах она соответствует истине (в отличие от undef). В успешно выполненных запросах ее также можно использовать для подсчета количества обработанных строк, поскольку в числовом представлении она интерпретируется как нуль. Конечно, если вывести это значение «как есть», будет выведено "0E0", и пользователи вашей программы явно будут озадачены увиденным. В предыдущем примере показано, как убедиться в том, что этого не случилось: добавление к значению нуля явно приводит его к числовой форме, так что строка выводится как 0. Для осуществления неявного преобразования в число можно применить printf с указателем формата %d:

```
my $count = $dbh->do ("UPDATE profile SET color = color WHERE name = 'Fred'");
if ($count)     # если ошибок не возникло, выводится количество строк
{
    printf "%d rows were updated\n", $count;
}
```

Если включен атрибут RaiseError, сценарий будет автоматически завершаться при возникновении связанной с DBI ошибки, и вам не придется проверять $count, чтобы узнать о неудачном выполнении do():

```
my $count = $dbh->do ("UPDATE profile SET color = color WHERE name = 'Fred'");
printf "%d rows were updated\n", $count;
```

Для обработки запросов, возвращающих результирующее множество, таких как SELECT, используется другой подход, состоящий из четырех этапов:

- Определите запрос, вызвав при помощи дескриптора базы данных prepare(). Функция prepare()возвращает дескриптор предложения, который будет использоваться во всех последующих операциях над запросом (в случае ошибки сценарий завершает работу, если включен атрибут RaiseError, иначе prepare() возвращает undef).

- Вызовите execute() для выполнения запроса и формирования результирующего множества.
- В цикле выберите строки, возвращенные запросом. DBI предоставляет несколько методов, которые могут использоваться в таком цикле (они будут кратко рассмотрены далее).
- Освободите ресурсы, выделенные результирующему множеству, вызвав finish().

Следующий пример демонстрирует описанные шаги, используя в качестве метода выборки fetchrow_array() и действуя в предположении, что включен атрибут RaiseError:

```
my $sth = $dbh->prepare ("SELECT id, name, cats FROM profile");
$sth->execute ();
my $count = 0;
while (my @val = $sth->fetchrow_array ())
{
    print "id: $val[0], name: $val[1], cats: $val[2]\n";
    ++$count;
}
$sth->finish ();
print "$count rows were returned\n";
```

За циклом выборки строк следует вызов finish(), который закрывает результирующее множество и сообщает серверу, что тот может освободить ресурсы, выделенные для этого множества. На самом деле, если вы выбираете все строки результирующего множества, то не нужно вызывать finish(), поскольку когда DBI замечает, что вы достигли последней строки, множество освобождается. То есть если в примере опустить вызов finish(), то ничего не изменится. Но важно явно вызывать finish(), если вы выбираете только часть результирующего множества.

В примере показано, что если вы хотите узнать, сколько строк содержит результирующее множество, вам придется считать их самостоятельно по мере извлечения. Не используйте для этого метод DBI rows(); документация по DBI не рекомендует так поступать (причина в том, что метод не всегда достоверен для предложений SELECT – не из-за проблем DBI, а из-за отличий в поведении разных баз данных).

DBI содержит несколько функций, которые могут использоваться для получения строк по одной в цикле выборки строк. Используемая в предыдущем примере функция fetchrow_array() возвращает массив, содержащий следующую строку, или же пустой список, если строк больше нет. К элементам массива можно обращаться как к $val[0], $val[1], ..., их порядок в массиве повторяет порядок их упоминания в предложении SELECT. Эта функция наиболее полезна для запросов, в которых явно указаны названия столбцов для выборки (если столбцы извлекаются запросом SELECT *, неизвестно, как столбцы будут расположены в массиве).

Функция fetchrow_arrayref() похожа на fetchrow_array(), за исключением того, что она возвращает ссылку на массив или undef, если строк для извлечения

больше нет. Доступ к элементам массива осуществляется как `$ref->[0]`, `$ref->[1]` и т. д. Как и в случае с `fetchrow_array()`, значения представлены в порядке упоминания в запросе:

```
my $sth = $dbh->prepare ("SELECT id, name, cats FROM profile");
$sth->execute ();
my $count = 0;
while (my $ref = $sth->fetchrow_arrayref ())
{
    print "id: $ref->[0], name: $ref->[1], cats: $ref->[2]\n";
    ++$count;
}
print "$count rows were returned\n";
```

Функция `fetchrow_hashref()` возвращает ссылку на хеш-структуру или `undef`, если строк для извлечения больше нет:

```
my $sth = $dbh->prepare ("SELECT id, name, cats FROM profile");
$sth->execute ();
my $count = 0;
while (my $ref = $sth->fetchrow_hashref ())
{
    print "id: $ref->{id}, name: $ref->{name}, cats: $ref->{cats}\n";
    ++$count;
}
print "$count rows were returned\n";
```

Для обращения к элементам хеша используются имена столбцов, выбранных запросом (`$ref->{id}`, `$ref->{name}` и т. д.). Функция `fetchrow_hashref()` особенно полезна в запросах `SELECT *`, так как можно обращаться к элементам строк, ничего не зная о порядке следования столбцов. Необходимо знать только их имена. Но и здесь есть свои минусы – создание хеша требует больших затрат, чем создание массива, поэтому `fetchrow_hashref()` медленнее, чем `fetchrow_array()` и `fetchrow_arrayref()`. Кроме того, не исключена возможность «потери» элементов строк, имеющих одинаковые имена, ведь имена столбцов должны быть уникальными. Следующий запрос выбирает два значения, но `fetchrow_hashref()` вернет хеш-структуру, содержащую лишь один элемент с именем `id`:

```
SELECT id, id FROM profile
```

Избежать подобных проблем можно при помощи псевдонимов столбцов, что обеспечит различимость похожих названий столбцов в результирующем множестве. Следующий запрос извлекает те же столбцы, что и предыдущий, но дает им различные имена, `id` и `id2`:

```
SELECT id, id AS id2 FROM profile
```

Этот запрос может показаться глупым, но если вы извлекаете столбцы из нескольких таблиц, то возникновение ситуации с совпадающими названиями столбцов результирующего множества очень даже вероятно. Подобный пример приведен в рецепте 12.3.

В дополнение к только что описанным способам выполнения запроса DBI предоставляет несколько высокоуровневых методов поиска информации, которые выдают запрос и возвращают результирующее множество в одной операции. Это методы дескриптора базы данных, которые создают дескриптор предложения и управляют им до возврата результирующего множества. Отличие методов состоит в форме возвращаемого результата. Одни методы возвращают результирующее множество целиком, другие возвращают одну или несколько строк (табл. 2.2):[1]

Таблица 2.2. Варианты возврата результата запроса

Метод	Возвращаемое значение
selectrow_array()	Первая строка результирующего множества в виде массива
selectrow_arrayref()	Первая строка результирующего множества как ссылка на массив
selectrow_hashref()	Первая строка результирующего множества как ссылка на хеш
selectcol_arrayref()	Первый столбец результирующего множества как ссылка на массив
selectall_arrayref()	Полное результирующее множество как ссылка на массив, состоящий из ссылок на массив
selectall_hashref()	Полное результирующее множество как ссылка на хеш, состоящий из ссылок на хеш

Большинство методов возвращают ссылку. Исключением является метод selectrow_array(), который выбирает первую строку результирующего множества и возвращает массив или скаляр, в зависимости от того, как вы его вызываете. В случае массива selectrow_array() возвращает целую строку как массив (или пустой список, если ни одной строки не было выбрано). Метод удобно использовать, если вы ожидаете, что запрос вернет одну строку:

```
my @val = $dbh->selectrow_array (
            "SELECT name, birth, foods FROM profile WHERE id = 3");
```

Когда selectrow_array() возвращает массив, возвращенное значение может использоваться для определения размера результирующего множества. Количество столбцов равно количеству элементов массива, а количество строк – или 1, или 0:

```
my $ncols = @val;
my $nrows = ($ncols ? 1 : 0);
```

Можно вызвать метод selectrow_array() так, чтобы он возвращал скаляр, тогда будет выдан только первый столбец строки. Это особенно удобно для запросов, возвращающих единственное значение:

```
my $buddy_count = $dbh->selectrow_array ("SELECT COUNT(*) FROM profile");
```

[1] Для selectrow_arrayref() и selectall_hashref() требуется версия DBI 1.15 или выше. Для selectrow_hashref() требуется DBI 1.20 или выше (этот метод присутствовал и в нескольких более ранних версиях, но тогда он работал иначе).

Если запрос не возвращает результат, selectrow_array() возвращает пустой массив или undef (если возвращается скаляр).

Методы selectrow_arrayref() и selectrow_hashref()выбирают первую строку результирующего множества и возвращают ссылку на нее или undef, если ни одной строки не выбрано. Чтобы получить доступ к значениям столбцов, поступите со ссылкой так же, как вы работали бы со значением, возвращенным методом fetchrow_arrayref() или fetchrow_hashref(). Вы также можете использовать ссылку для получения количества строк и столбцов:

```
my $ref = $dbh->selectrow_arrayref ($query);
my $ncols = (defined ($ref) ? @{$ref} : 0);
my $nrows = ($ncols ? 1 : 0);

my $ref = $dbh->selectrow_hashref ($query);
my $ncols = (defined ($ref) ? keys (%{$ref}) : 0);
my $nrows = ($ncols ? 1 : 0);
```

При использовании метода selectcol_arrayref() возвращается ссылка на одномерный массив, представляющий первый столбец результирующего множества. Если считать, что возвращается не undef, то для получения значения строки *i* к элементам массива можно обратиться как $ref->[i]. Количество строк – это количество элементов массива, а количество столбцов – или 1, или 0:

```
my $ref = $dbh->selectcol_arrayref ($query);
my $nrows = (defined ($ref) ? @{$ref} : 0);
my $ncols = ($nrows ? 1 : 0);
```

Метод selectall_arrayref() возвращает ссылку на массив, при этом массив содержит по одному элементу для каждой строки результата. Каждый из этих элементов представляет собой ссылку на массив. Для доступа к строке *i* результирующего множества используйте $ref->[i], чтобы получить ссылку на строку. Затем для доступа к значениям отдельных полей строки работайте с этой ссылкой так же, как со значением, возвращаемым методом fetchrow_arrayref(). Количество строк и столбцов результирующего множества можно получить следующим образом:

```
my $ref = $dbh->selectall_arrayref ($query);
my $nrows = (defined ($ref) ? @{$ref} : 0);
my $ncols = ($nrows ? @{$ref->[0]} : 0);
```

Метод selectall_hashref() подобен методу selectall_arrayref(), но возвращает ссылку на хеш, каждый элемент которого является хеш-ссылкой на строку результирующего множества. При его вызове необходимо указать аргумент – столбец, который будет использоваться как ключ хеша. Например, если вы извлекаете строки из таблицы profile, первичным ключом будет столбец id:

```
my $ref = $dbh->selectall_hashref ("SELECT * FROM profile", "id");
```

Затем обращайтесь к строкам, используя ключи хеша. Например, если для одной из строк значение ключевого столбца равно 12, хеш-ссылка на эту строку доступна как $ref->{12}. Эти значения связаны с именами столбцов, с помощью которых можно обратиться к их отдельным элементам (напри-

мер, $ref->{12}->{name}). Количество строк и столбцов результирующего множества может быть получено следующим образом:

```
my @keys = (defined ($ref) ? keys (%{$ref}) : ());
my $nrows = scalar (@keys);
my $ncols = ($nrows ? keys (%{$ref->{$keys[0]}}) : 0);
```

Методы selectall_*XXX*() удобны, когда необходима многократная обработка результирующего множества, так как «прокрутка» результата в DBI не поддерживается. Присвоив все результирующее множество переменной, вы можете как угодно часто перемещаться по его элементам.

Будьте внимательны при работе с высокоуровневыми методами при отключенном атрибуте RaiseError. В случае отключения RaiseError значение, возвращенное методом, не всегда позволяет отличить пустое результирующее множество от ошибки. Например, если вы вызываете selectrow_array() для поиска отдельного значения (возвращается скаляр), то возвращенное значение undef может быть любым из трех: ошибкой, пустым результирующим множеством или результирующим множеством, состоящим из одного значения NULL. При тестировании на ошибки можно проверять значение $DBI::errstr или $DBI::err.

PHP

В PHP нет отдельных функций для выдачи запросов, возвращающих и не возвращающих результирующее множество. Есть одна функция для всех запросов – mysql_query(), которая принимает в качестве аргументов строку запроса и необязательный идентификатор соединения и возвращает идентификатор результата. Если опустить аргумент идентификатора соединения, mysql_query() по умолчанию использует последнее открытое соединение. Ниже приведены примеры, в первом из которых идентификатор указан явно, а во втором используется соединение по умолчанию:

```
$result_id = mysql_query ($query, $conn_id);
$result_id = mysql_query ($query);
```

Если не удается выполнить запрос, $result_id возвращает FALSE. Это значит, что ошибка возникла из-за проблем с запросом: в нем была синтаксическая ошибка, у вас не было прав на доступ к названной в запросе таблице или же случилось что-то еще, что помешало выполнению запроса. Возвращенное значение FALSE *не* означает, что запрос обработал 0 строк (для DELETE, INSERT или UPDATE) или вернул 0 строк (для SELECT).

Если $result_id не возвращает FALSE, запрос выполнен успешно. Дальнейшие действия зависят от типа запроса. Для запросов, не возвращающих результирующее множество, значение $result_id равно TRUE, и выполнение запроса завершено. При желании можно вызвать метод mysql_affected_rows(), чтобы посмотреть, сколько строк изменилось:

```
$result_id = mysql_query ("DELETE FROM profile WHERE cats = 0", $conn_id);
if (!$result_id)
```

```
    die ("Oops, the query failed");
print (mysql_affected_rows ($conn_id) . " rows were deleted\n");
```

Метод `mysql_affected_rows()` принимает в качестве аргумента идентификатор соединения. Если не указать его, используется текущее соединение.

Для запросов, возвращающих результирующее множество, `mysql_query()` возвращает ненулевой идентификатор результата. Обычно этот идентификатор используется для вызова функции выбора строк в цикле, затем вызывается `mysql_free_result()` для освобождения результирующего множества. Фактически идентификатор результата – это просто число, указывающее PHP на то, какое результирующее множество используется. Идентификатор *не* является счетчиком выбранных строк и не включает в себя содержимое какой бы то ни было строки. Многие начинающие PHP-программисты ошибочно полагают, что `mysql_query()` возвращает количество строк или результирующее множество, но на самом деле это не так. Прояснив для себя данный вопрос, вы избавитесь от возможных проблем.

Рассмотрим пример, показывающий, как запустить запрос `SELECT` и использовать идентификатор результата для выборки строк:

```
$result_id = mysql_query ("SELECT id, name, cats FROM profile", $conn_id);
if (!$result_id)
    die ("Oops, the query failed");
while ($row = mysql_fetch_row ($result_id))
    print ("id: $row[0], name: $row[1], cats: $row[2]\n");
print (mysql_num_rows ($result_id) . " rows were returned\n");
mysql_free_result ($result_id);
```

В примере показано, что для получения строк результирующего множества выполняется цикл, в котором идентификатор результата передается в одну из функций поиска строк PHP. Чтобы подсчитать количество строк результирующего множества, передайте идентификатор результата функции `mysql_num_rows()`. Когда строк больше нет, передайте идентификатор функции `mysql_free_result()` для закрытия результирующего множества. (Не пытайтесь выбирать строки или вычислять количество строк после вызова `mysql_free_result()`, так как с этого момента `$result_id` уже не действителен.)

Любая PHP-функция извлечения строк возвращает следующую строку результирующего множества, на которое указывает `$result_id`, или `FALSE`, если строк больше нет. Функции отличаются друг от друга типом данных возвращаемого значения. Функция из предыдущего примера, `mysql_fetch_row()`, возвращает массив, элементы которого соответствуют столбцам, выбранным запросом, доступ к которым осуществляется с помощью числового индекса. Функция `mysql_fetch_array()` похожа на `mysql_fetch_row()`, возвращаемый ею массив также содержит элементы, обращаться к которым можно по названиям выбранных столбцов. Иначе говоря, обратиться к любому столбцу можно по его числовому индексу или по имени:

```
$result_id = mysql_query ("SELECT id, name, cats FROM profile", $conn_id);
if (!$result_id)
    die ("Oops, the query failed");
```

```
while ($row = mysql_fetch_array ($result_id))
{
    print ("id: $row[0], name: $row[1], cats: $row[2]\n");
    print ("id: $row[id], name: $row[name], cats: $row[cats]\n");
}
print (mysql_num_rows ($result_id) . " rows were returned\n");
mysql_free_result ($result_id);
```

Можно подумать, что функция `mysql_fetch_array()` должна работать заметно медленнее, чем `mysql_fetch_row()`, но это не так, несмотря на то, что возвращаемый ею массив содержит больше информации.

В рассмотренном примере названия элементов не заключались в кавычки, так как они находились внутри строки символов, взятой в кавычки (string). Если же необходимо сослаться на элементы вне строки символов, их имена должны быть заключены в кавычки:

```
printf ("id: %s, name: %s, cats: %s\n",
        $row["id"], $row["name"], $row["cats"]);
```

Функция `mysql_fetch_object()` возвращает объект, члены которого соответствуют столбцам, выбранным запросом, обращение к которым осуществляется по именам столбцов:

```
$result_id = mysql_query ("SELECT id, name, cats FROM profile", $conn_id);
if (!$result_id)
```

Не используйте в PHP 3 функцию count() для подсчета количества столбцов

Иногда PHP-программисты выбирают строку результирующего множества, а затем применяют функцию `count($row)` для подсчета входящих в нее элементов. Но, как видно из предлагаемого ниже фрагмента кода, предпочтительнее использовать `mysql_num_fields()`:

```
if (!($result_id = mysql_query ("SELECT 1, 0, NULL", $conn_id)))
    die ("Cannot issue query\n");
$count = mysql_num_fields ($result_id);
print ("The row contains $count columns\n");
if (!($row = mysql_fetch_row ($result_id)))
    die ("Cannot fetch row\n");
$count = count ($row);
print ("The row contains $count columns\n");
```

Если выполнить этот код в PHP 3, то `count()` возвращает 2. А в PHP 4 `count()` возвращает 3. Разница в результатах объясняется тем, что `count()` считает значения массива, соответствующие значениям `NULL` в PHP 4, но не в PHP 3. Напротив, `mysql_field_count()` возвращает 3 в обеих версиях PHP. То есть имейте в виду, что `count()` не всегда выдает корректное значение. Если вы хотите получить достоверную информацию о количестве столбцов, используйте `mysql_field_count()`.

```
    die ("Oops, the query failed");
while ($row = mysql_fetch_object ($result_id))
    print ("id: $row->id, name: $row->name, cats: $row->cats\n");
print (mysql_num_rows ($result_id) . " rows were returned\n");
mysql_free_result ($result_id);
```

PHP 4.0.3 содержит четвертую дополнительную функцию выборки строк, `mysql_fetch_assoc()`, которая возвращает массив элементов, доступных по именам. Другими словами, она похожа на `mysql_fetch_array()`, за тем исключением, что строка не содержит значений, доступных по числовому индексу.

Python

Интерфейс Python DB-API не предлагает различных вызовов для запросов, возвращающих и не возвращающих результирующее множество. Для обработки запроса в Python используйте объект соединения с базой данных, получив с его помощью объект курсора.[1] Затем используйте метод курсора `execute()` для отправки запроса на сервер. Если запрос не возвращает результат, он завершен, и можно использовать атрибут курсора `rowcount` для определения количества измененных строк:[2]

```
try:
    cursor = conn.cursor ()
    cursor.execute ("UPDATE profile SET cats = cats+1 WHERE name = 'Fred'")
    print "%d rows were updated" % cursor.rowcount
except MySQLdb.Error, e:
    print "Oops, the query failed"
    print e
```

Если запрос возвращает результирующее множество, выберите его строки и закройте множество. DB-API содержит пару методов извлечения строк. Метод `fetchone()` возвращает следующую строку в виде объекта-последовательности (sequence) или `None`, если строк для выборки больше нет:

```
try:
    cursor = conn.cursor ()
    cursor.execute ("SELECT id, name, cats FROM profile")
    while 1:
        row = cursor.fetchone ()
        if row == None:
            break
        print "id: %s, name: %s, cats: %s" % (row[0], row[1], row[2])
    print "%d rows were returned" % cursor.rowcount
    cursor.close ()
```

---

[1] Если вам знакомо понятие «курсор» таким, как оно присутствует с серверной стороны некоторых баз данных, знайте, что в MySQL курсоры – это не совсем то же самое. Модуль MySQLdb при выполнении запроса эмулирует курсоры с клиентской стороны.

[2] Обратите внимание, что `rowcount` – это атрибут, а не функция. Чтобы не инициировать исключение, указывайте `rowcount`, а не `rowcount()`.

```
except MySQLdb.Error, e:
    print "Oops, the query failed"
    print e
```

Как видите, атрибут `rowcount` полезен и для запросов `SELECT` – он показывает количество строк результирующего множества.

Второй метод извлечения строк, `fetchall()`, возвращает все результирующее множество целиком как последовательность последовательностей. Вы можете перемещаться по последовательности, чтобы обращаться к строкам:

```
try:
    cursor = conn.cursor ()
    cursor.execute ("SELECT id, name, cats FROM profile")
    rows = cursor.fetchall ()
    for row in rows:
        print "id: %s, name: %s, cats: %s" % (row[0], row[1], row[2])
    print "%d rows were returned" % cursor.rowcount
    cursor.close ()
except MySQLdb.Error, e:
    print "Oops, the query failed"
    print e
```

Как и DBI, DB-API не поддерживает прокрутку результирующего множества, так что метод `fetchall()` может быть полезен, когда необходимо несколько раз пройтись по результирующему множеству или обратиться непосредственно к конкретным значениям. Например, если `rows` хранит результирующее множество, вы можете получить доступ к значению третьего столбца второй строки следующим образом: `rows[1][2]` (индексы начинаются с 0, а не с 1).

Чтобы обратиться к значениям строк по имени столбца, укажите для курсора при создании тип `DictCursor`. В этом случае строки будут возвращаться как словарные объекты Python с именованными элементами:

```
try:
    cursor = conn.cursor (MySQLdb.cursors.DictCursor)
    cursor.execute ("SELECT id, name, cats FROM profile")
    for row in cursor.fetchall ():
        print "id: %s, name: %s, cats: %s" \
                % (row["id"], row["name"], row["cats"])
    print "%d rows were returned" % cursor.rowcount
    cursor.close ()
except MySQLdb.Error, e:
    print "Oops, the query failed"
    print e
```

Java

Интерфейс JDBC предоставляет специальные типы объектов для разных этапов обработки запроса. В JDBC запросы выдаются путем передачи SQL-строк Java-объектам одного типа, а результаты, если они есть, возвращаются как объекты другого типа. При возникновении проблем с доступом к базе данных порождаются исключения.

Для выдачи запроса в первую очередь необходимо вызвать метод `createStatement()` вашего объекта `Connection` для получения объекта `Statement`:

```
Statement s = conn.createStatement ();
```

Затем используйте объект `Statement` для отправки запроса на сервер. Для этого в JDBC есть несколько методов – выберите соответствующий тому типу запроса, который планируется выполнить: `executeUpdate()` – для запросов, не возвращающих результирующее множество, `executeQuery()` – для запросов, возвращающих результат, и `execute()` – если вы не знаете, будет ли результат.

Метод `executeUpdate()` отправляет на сервер запрос, не порождающий результирующее множество, и возвращает счетчик – количество обработанных строк. По окончании работы с объектом предложения необходимо закрыть его. Рассмотрим описанную последовательность действий на примере:

```
try
{
    Statement s = conn.createStatement ();
    int count = s.executeUpdate ("DELETE FROM profile WHERE cats = 0");
    s.close ();                     // close statement
    System.out.println (count + " rows were deleted");
}
catch (Exception e)
{
    Cookbook.printErrorMessage (e);
}
```

Для запросов, возвращающих результат, выполните `executeQuery()`. Получите объект результирующего множества и используйте его для извлечения значений строк. По завершении работы закройте два объекта: результирующее множество и предложение:

```
try
{
    Statement s = conn.createStatement ();
    s.executeQuery ("SELECT id, name, cats FROM profile");
    ResultSet rs = s.getResultSet ();
    int count = 0;
    while (rs.next ())              // цикл по строкам результирующего множества
    {
        int id = rs.getInt (1);     // извлечь столбцы 1, 2 и 3
        String name = rs.getString (2);
        int cats = rs.getInt (3);
        System.out.println ("id: " + id
                            + ", name: " + name
                            + ", cats: " + cats);
        ++count;
    }
    rs.close ();                    // закрыть результирующее множество
    s.close ();                     // закрыть предложение
    System.out.println (count + " rows were returned");
}
```

```
catch (Exception e)
{
    Cookbook.printErrorMessage (e);
}
```

У объекта `ResultSet`, возвращаемого методом `getResultSet()` вашего объекта `Statement`, есть несколько собственных методов, таких как `next()` для выборки строк и различные методы `get`*XXX*`()` для доступа к столбцам текущей строки. Первоначально результирующее множество позиционируется непосредственно перед первой строкой множества. Вызывайте `next()` для последовательной выборки всех строк до тех пор, пока не будет возвращено значение `False`, показывающее, что строк больше нет. Если вас интересует количество строк результирующего множества, подсчитайте его самостоятельно, как показано в предыдущем примере.

Извлечение значений столбцов осуществляется при помощи таких методов, как `getInt()`, `getString()`, `getFloat()` и `getDate()`. Чтобы получить значение столбца как некоторый общий объект, используйте метод `getObject()`. При вызове метода `get`*XXX*`()` можно указывать аргумент: номер столбца (начиная с 1, а не с 0) или имя столбца. Ранее показывалось, как извлечь столбцы `id`, `name` и `cats`, зная их местоположение. Чтобы сослаться на столбцы по имени, перепишем цикл выборки строк следующим образом:

```
while (rs.next ())  // цикл по строкам результирующего множества
{
    int id = rs.getInt ("id");
    String name = rs.getString ("name");
    int cats = rs.getInt ("cats");
    System.out.println ("id: " + id
                        + ", name: " + name
                        + ", cats: " + cats);
    ++count;
}
```

Можно извлекать значение столбца, используя вызов любого из методов `get`*XXX*`()`, подходящего для столбца такого типа. Например, можно извлечь значение любого столбца в виде строки с помощью метода `getString()`:

```
String id = rs.getString ("id");
String name = rs.getString ("name");
String cats = rs.getString ("cats");
System.out.println ("id: " + id
                    + ", name: " + name
                    + ", cats: " + cats);
```

А можно вызвать `getObject()`, чтобы извлечь значения в виде объектов, а затем преобразовать их, если нужно. Например, преобразуем объектные значения в форму, пригодную для печати, с помощью метода `toString()`:

```
Object id = rs.getObject ("id");
Object name = rs.getObject ("name");
Object cats = rs.getObject ("cats");
```

```
System.out.println ("id: " + id.toString ()
                    + ", name: " + name.toString ()
                    + ", cats: " + cats.toString ());
```

Чтобы узнать, сколько столбцов содержится в каждой строке, необходимо обратиться к метаданным результирующего множества. Приведенный ниже код выводит список значений, разделенных запятыми, – количество столбцов в каждой строке:

```
try
{
    Statement s = conn.createStatement ();
    s.executeQuery ("SELECT * FROM profile");
    ResultSet rs = s.getResultSet ();
    ResultSetMetaData md = rs.getMetaData ();   // получить метаданные
                                                // результирующего множества
    int ncols = md.getColumnCount ();           // получить количество
                                                // столбцов из метаданных
    int count = 0;
    while (rs.next ())  // цикл по строкам результирующего множества
    {
        for (int i = 0; i < ncols; i++)         // цикл по столбцам
        {
            String val = rs.getString (i+1);
            if (i > 0)
                System.out.print (", ");
            System.out.print (val);
        }
        System.out.println ();
        ++count;
    }
    rs.close ();    // закрыть результирующее множество
    s.close ();     // закрыть предложение
    System.out.println (count + " rows were returned");
}
catch (Exception e)
{
    Cookbook.printErrorMessage (e);
}
```

Третий JDBC-метод выполнения запросов, `execute()`, работает с любым типом запросов. Метод удобно использовать, когда строка запроса получается из внешнего источника и заранее неизвестно, будет ли сформировано результирующее множество. Значение, возвращаемое методом `execute()`, указывает на тип запроса, так что вы можете обрабатывать его надлежащим образом: если `execute()` возвращает значение «истина», запрос порождает результирующее множество, иначе – нет. Обычно метод применяется как-то так (`queryStr` представляет собой произвольное предложение SQL):

```
try
{
    Statement s = conn.createStatement ();
```

```
    if (s.execute (queryStr))
    {
        // есть результирующее множество
        ResultSet rs = s.getResultSet ();

        // ... здесь обрабатывается результирующее множество ...

        rs.close ();    // закрыть результирующее множество
    }
    else
    {
        // результирующего множества нет, просто вывести количество строк
        System.out.println (s.getUpdateCount ()
                            + " rows were affected");
    }
    s.close ();     // закрыть предложение
}
catch (Exception e)
{
    Cookbook.printErrorMessage (e);
}
```

Закрытие предложений JDBC и объектов результирующих множеств

Рассмотренные в разделе примеры запуска запросов JDBC явно закрывают объекты предложения и результирующего множества при завершении работы с этими объектами. Некоторые продукты Java автоматически закрывают их при закрытии соединения. Однако лучше на это не надеяться и во избежание проблем закрывать объекты самостоятельно.

2.5. Перемещение по результирующему множеству

Задача

Требуется перейти к произвольной строке результирующего множества или обойти его несколько раз.

Решение

Если ваш API поддерживает такие функции, используйте их. В противном случае извлеките результирующее множество в структуру данных, чтобы обращаться к строкам когда и как угодно.

Обсуждение

Некоторые API разрешают «перемотку» («rewind») результирующего множества, так что вы можете повторно обращаться к его строкам. Некоторые,

кроме того, позволяют переходить к произвольной строке результирующего множества, что, в сущности, означает возможность произвольной выборки строк. Рассматриваемые нами API соотносятся с такой функциональностью следующим образом:

- Perl DBI и Python DB-API не допускают непосредственного позиционирования внутри результирующего множества.
- PHP поддерживает позиционирование строк посредством функции `mysql_data_seek()`. Передайте ей идентификатор результирующего множества и номер строки (в диапазоне от 0 до `mysql_num_rows()`−1). Последующие вызовы функций извлечения строк возвращают строки друг за другом, начиная с указанной. PHP также содержит функцию `mysql_result()`, которая принимает индексы строки и столбца для произвольного доступа к отдельным значениям внутри результирующего множества. Однако `mysql_result()` работает очень медленно и обычно не используется.
- JDBC 2 вводит понятие «прокручиваемого» («scrollable») результирующего множества, а также предоставляет методы перемещения по строкам вперед и назад. В более ранних версиях JDBC таких возможностей нет, хотя драйвер MySQL Connector/J поддерживал методы `next()` и `previous()` даже для JDBC 1.12.

Вне зависимости от того, разрешает ли конкретный API перемотку и позиционирование, ваши программы могут обеспечить доступ к отдельным значениям результирующего множества за счет извлечения всех строк и сохранения их в структуре данных. Например, можно использовать двумерный массив для хранения строк и столбцов результата в виде элементов матрицы. После того как строки сохранены, вы можете сколько угодно раз проходить по результирующему множеству или обращаться к его произвольным элементам. Если ваш API содержит вызов, возвращающий в одной операции все результирующее множество, сформировать такую матрицу достаточно просто. (Perl и Python могут это сделать.) Иначе необходимо создать цикл по извлечению строк и самостоятельно сохранять эти строки.

2.6. Использование в запросах подготовленных предложений и заполнителей

Задача

Необходимо написать обобщенные запросы, не ссылающиеся на конкретные значения, с целью их многократного использования.

Решение

Если ваш API поддерживает механизм заполнителей (placeholder), используйте его.

Обсуждение

Для того чтобы сформировать в программе предложение SQL, можно поместить значения данных непосредственно в строку запроса:

```
SELECT * FROM profile WHERE age > 40 AND color = 'green'

INSERT INTO profile (name,color) VALUES('Gary','blue')
```

Некоторые API предлагают альтернативу – создание строк запроса, не содержащих литералов. Используя такой подход, можно писать предложения при помощи заполнителей – специальных символов, указывающих, где должны находиться значения. Общепринятым заполнителем является символ ?; подставим его в только что приведенные запросы:

```
SELECT * FROM profile WHERE age > ? AND color = ?

INSERT INTO profile (name,color) VALUES(?,?)
```

При работе в API, поддерживающих этот механизм, вы передаете строку в базу данных для подготовки плана запроса. Затем вы передаете значения данных и присваиваете их заполнителям при выполнении запроса. Запрос может выполняться несколько раз, при этом вводимые значения каждый раз могут быть новыми.

Одним из преимуществ использования подготовленных предложений и заполнителей является то, что операции связывания параметров автоматически обрабатывают такие специальные символы, как кавычки и обратный слэш, о которых вам приходится заботиться самим, если вы помещаете данные непосредственно в запрос. Это особенно полезно в тех случаях, когда, например, вы помещаете двоичные данные (изображения) в базу данных или используете данные с неизвестным содержимым, например данные, введенные удаленным пользователем с клавиатуры в форму на веб-странице.

Еще один положительный момент – предложения можно использовать многократно. Предложения становятся более общими, так как содержат не конкретные значения данных, а заполнители. Если вы выполняете какую-то операцию снова и снова, вероятно, вы можете использовать подготовленное предложение и при каждом выполнении просто передавать ему новые значения. Увеличивается производительность, по крайней мере, для тех баз данных, которые составляют планы запросов. Например, если программа в процессе работы несколько раз выдает определенный тип предложения SELECT, такая база данных может один раз составить план запроса, а затем использовать его при каждом исполнении вместо того, чтобы каждый раз пересоздавать план. MySQL не формирует план запроса, так что выигрыша в производительности от использования подготовленных предложений вы не получите. Но если перенести программу на базу данных, работающую с планами запросов, вы автоматически улучшите ее производительность за счет использования подготовленных предложений.

Третий плюс в том, что код, использующий запросы с заполнителями, становится более удобочитаемым, хотя это мнение можно и оспорить. Сравните

сами запросы данного раздела с запросами из предыдущего, где не использовались заполнители, и сделайте свой выбор.

Perl

Чтобы использовать заполнители в сценариях DBI, поместите ? во все те места запроса, где предполагаются значения данных, затем свяжите значения с запросом. Связывание значений осуществляется путем передачи их в методы do() или execute() или за счет вызова метода DBI, специально предназначенного для замещения заполнителей.

Если вы используете do(), передавайте в одном вызове и строку запроса, и значения данных:

```
my $count = $dbh->do ("UPDATE profile SET color = ? WHERE name = ?",
                      undef,
                      "green", "Mara");
```

В списке аргументов за строкой запроса следует undef, затем приводится список значений – по одному для каждого заполнителя. (Аргумент undef является историческим артефактом, но он необходим.)

При использовании метода prepare() вместе с execute(), передайте строку запроса prepare() для получения дескриптора предложения. Затем с помощью этого дескриптора передайте значения данных через execute():

```
my $sth = $dbh->prepare ("UPDATE profile SET color = ? WHERE name = ?");
my $count = $sth->execute ("green", "Mara");
```

Можно использовать заполнители и в предложениях SELECT. Следующий запрос ищет записи, значение name которых начинается с «M»:

```
my $sth = $dbh->prepare ("SELECT * FROM profile WHERE name LIKE ?");
$sth->execute ("M%");
while (my $ref = $sth->fetchrow_hashref ())
{
    print "id: $ref->{id}, name: $ref->{name}, cats: $ref->{cats}\n";
}
$sth->finish ();
```

Есть и третий способ связывания значений с заполнителями – вызов метода bind_param(). Метод принимает два аргумента: положение заполнителя и значение, которое будет подставлено вместо данного заполнителя (нумерация позиций заполнителей начинается с 1, а не с 0). Перепишем два предыдущих примера, используя bind_param():

```
my $sth = $dbh->prepare ("UPDATE profile SET color = ? WHERE name = ?");
$sth->bind_param (1, "green");
$sth->bind_param (2, "Mara");
my $count = $sth->execute ();
```

```
my $sth = $dbh->prepare ("SELECT * FROM profile WHERE name LIKE ?");
$sth->bind_param (1, "M%");
$sth->execute ();
```

```
while (my $ref = $sth->fetchrow_hashref ())
{
    print "id: $ref->{id}, name: $ref->{name}, cats: $ref->{cats}\n";
}
$sth->finish ();
```

Какой бы метод вы ни использовали при работе с заполнителями, никогда не заключайте символ ? в кавычки, даже если заполнители заменяют строки. DBI самостоятельно добавит кавычки в случае необходимости. Если же вы заключите заполнитель в кавычки, DBI воспримет его как буквальную строковую константу "?", а не как заполнитель.

Формирование списка заполнителей

Если вы хотите использовать заполнители для множества значений данных переменного размера, необходимо построить список символов-заполнителей. Например, в Perl следующее предложение создает строку, состоящую из *n* символов-заполнителей, разделенных запятыми:

```
$str = join (",", ("?") x n);
```

Оператор повторения x, будучи применен к списку, создает *n* копий списка, так что вызов join() соединяет эти списки, формируя единую строку, содержащую *n* экземпляров символа ?, разделенных запятыми. Это удобно, когда вы хотите связать массив значений со списком заполнителей из строки запроса – размер массива показывает, сколько требуется символов-заменителей:

```
$str = join (",", ("?") x @values);
```

Есть и другой способ формирования списка заполнителей, который, может быть, выглядит более понятно:

```
$str = "?" if @values;
$str .= ",?" for 1 .. @values-1;
Есть и третий:
$str = "?" if @values;
for (my $i = 1; $i < @values; $i++)
{
    $str .= ",?";
}
```

Синтаксис последнего метода наименее зависит от специфики Perl, и поэтому его легче перевести на другой язык. Например, аналогичный метод в Python выглядит так:

```
str = ""
if len (values) > 0:
    str = "?"
for i in range (1, len (values)):
    str = str + ",?"
```

Заполнители можно использовать и с такими высокоуровневыми методами выборки, как `selectrow_array()` и `selectall_arrayref()`. Как и для метода `do()`, аргументами являются строка запроса и `undef`; за ними следуют значения данных, которые должны быть подставлены вместо заполнителей из строки запроса. Например:

```
my $ref = $dbh->selectall_arrayref (
                        "SELECT name, birth, foods FROM profile
                        WHERE id > ? AND color = ?",
                        undef, 3, "green");
```

PHP

PHP не поддерживает механизм заполнителей. Информацию о построении запросов, ссылающихся на значения данных, содержащих специальные символы, можно найти в рецепте 2.8. См. также рецепт 2.9, в котором создается основанный на классах интерфейс для PHP, эмулирующий заполнители.

Python

Модуль MySQLdb Python реализует заполнители посредством использования в строке запроса спецификации формата. Если вы хотите работать с заполнителями, вызовите метод `execute()` с двумя аргументами: строка запроса, содержащая спецификацию формата, и последовательность, содержащая значения, которые должны быть связаны со строкой запроса. Следующий запрос использует заполнители для поиска записей, в которых любимым цветом является зеленый (green), а количество кошек не превышает 2:

```
try:
    cursor = conn.cursor ()
    cursor.execute ("SELECT * FROM profile WHERE cats < %s AND color = %s", \
                                                        (2, "green"))
    for row in cursor.fetchall ():
        print row
    print "%d rows were returned" % cursor.rowcount
    cursor.close ()
except MySQLdb.Error, e:
    print "Oops, the query failed"
    print e
```

Если с заполнителем должно быть связано единственное значение, *val*, его можно записать как последовательность (*val*,). Рассмотрим для примера предложение `UPDATE`:

```
try:
    cursor = conn.cursor ()
    cursor.execute ("UPDATE profile SET cats = cats+1 WHERE name = %s", \
                                                        ("Fred",))
    print "%d rows were updated" % cursor.rowcount
except MySQLdb.Error, e:
    print "Oops, the query failed"
    print e
```

Некоторые модули драйвера Python DB-API поддерживают несколько спецификаций формата (например, `%d` для целых чисел и `%f` для чисел с плавающей точкой). В MySQLdb необходимо использовать заполнитель `%s` для форматирования всех значений данных в строки. MySQL выполнит надлежащие преобразования типов. Если вы хотите поместить в запрос литеральный символ `%`, используйте в строке запроса `%%`.

При необходимости механизм заполнителей Python при связывании значений данных со строкой запроса заключает их в кавычки, так что вам не нужно добавлять кавычки самостоятельно.

Java

JDBC обеспечивает поддержку заполнителей при использовании подготовленных, а не обычных предложений. Как вы помните, для запуска обычного запроса следует создать объект `Statement` и передать строку запроса одной из функций запуска запросов – `executeUpdate()`, `executeQuery()` или `execute()`. Чтобы использовать подготовленный запрос, создайте объект `PreparedStatement`, передав строку запроса, содержащую символы-заполнители `?`, методу `prepareStatement()` вашего объекта соединения. Затем свяжите с предложением значения данных, используя методы `setXXX()`. Наконец, выполните предложение, вызвав `executeUpdate()`, `executeQuery()` или `execute()` с пустым списком аргументов. Приведем пример использования `executeUpdate()` для выполнения запроса `DELETE`:

```
PreparedStatement s;
int count;
s = conn.prepareStatement ("DELETE FROM profile WHERE cats = ?");
s.setInt (1, 2);    // связать значение 2 с первым заполнителем
count = s.executeUpdate ();
s.close ();     // закрыть предложение
System.out.println (count + " rows were deleted");
```

Для запросов, возвращающих результирующее множество, проводятся аналогичные действия, только вызывается метод `executeQuery()`:

```
PreparedStatement s;
s = conn.prepareStatement ("SELECT id, name, cats FROM profile"
                    + " WHERE cats < ? AND color = ?");
s.setInt (1, 2);    // связать значения 2 и "green"
                    // с первым и вторым заполнителями
s.setString (2, "green");
s.executeQuery ();
// ... здесь обработка результирующего множества данных ...
s.close ();     // закрыть предложение
```

Методы `setXXX()`, связывающие значения с предложениями, принимают два аргумента: позицию заполнителя (начиная с 1, а не с 0) и значение, которое должно быть подставлено вместо данного заполнителя. Тип значения должен соответствовать типу имени метода `setXXX()`. Например, в `setInt()` нужно передавать целое число, а не строку (string).

Нет необходимости заключать заполнители в строке запроса в кавычки. JDBC расставляет необходимые кавычки при связывании значений с заполнителями.

2.7. Использование в запросах специальных символов и значений NULL

Задача

У вас возникли проблемы с построением запросов, содержащих значения, включающие в себя специальные символы (кавычки, обратный слэш) или специальные значения (такие как NULL).

Решение

Используйте механизм заполнителей вашего API или функцию заключения в кавычки (quoting).

Обсуждение

Пока что рассматриваемые запросы содержали только «безопасные» данные, не требующие специальной обработки. В этом разделе рассказано о том, как строить запросы, в которых используются значения, содержащие специальные символы: кавычки, обратный слэш, двоичные данные или значения NULL. Рассмотрим возможные трудности на примере предложения INSERT:

```
INSERT INTO profile (name,birth,color,foods,cats)
VALUES('Alison','1973-01-12','blue','eggroll',4);
```

Ничего необычного. Но если заменить значение столбца name на нечто вроде De'Mont, содержащее одинарную кавычку, запрос станет синтаксически неверным:

```
INSERT INTO profile (name,birth,color,foods,cats)
VALUES('De'Mont','1973-01-12','blue','eggroll',4);
```

Проблема в наличии одинарной кавычки внутри строки, заключенной в одинарные кавычки. Чтобы сделать запрос корректным, необходимо экранировать кавычку, предварив ее одинарной кавычкой или символом обратного слэша:

```
INSERT INTO profile (name,birth,color,foods,cats)
VALUES('De''Mont','1973-01-12','blue','eggroll',4);

INSERT INTO profile (name,birth,color,foods,cats)
VALUES('De\'Mont','1973-01-12','blue','eggroll',4);
```

Есть и другой вариант – заключить значение name не в одинарные, а в двойные кавычки:

```
INSERT INTO profile (name,birth,color,foods,cats)
VALUES("De'Mont",'1973-01-12','blue','eggroll',4);
```

Конечно, если вы буквально вводите запрос в программе, то можете вручную заключить значение name в кавычки или поставить перед ним управляющий символ, так как знаете, что это за значение. Но если вы используете для получения значения name переменную, то можете и не знать, каким будет значение. И что еще хуже, одинарная кавычка – это не единственный символ, к неприятностям от которого вам следует подготовиться; двойные кавычки и символы обратного слэша тоже создают проблемы. А если вы хотите хранить в базе данных двоичные данные (изображения или звуковые клипы), такие значения могут содержать вообще все, что угодно – например значения NULL (нулевые байты). Проблема корректной обработки специальных символов особенно остра в веб-среде, где запросы строятся по введенному в форме образцу (например, если вы ищете записи, которые отвечают условиям поиска, заданным удаленным пользователем). Вы должны уметь обрабатывать в общем виде любую разновидность запросов, так как невозможно заранее предугадать, какую информацию введет пользователь. Надо сказать, что нередко встречаются и злоумышленники, специально вводящие значения с проблемными символами для разрушения ваших сценариев.

Значение NULL не относится к специальным символам, но тоже требует специальной обработки. В SQL NULL соответствует отсутствию значения. В зависимости от контекста это может означать «неизвестно», «отсутствует», «вне диапазона» и т. п. Ранее в наших запросах не использовались значения NULL, так как нам не хотелось заниматься создаваемыми ими трудностями, но теперь время пришло. Например, если вы не знаете любимый цвет человека с фамилией De'Mont, то можете установить столбец color в NULL – но не путем написания такого запроса:

```
INSERT INTO profile (name,birth,color,foods,cats)
VALUES('De''Mont','1973-01-12','NULL','eggroll',4);
```

Значения NULL не нужно заключать в кавычки:

```
INSERT INTO profile (name,birth,color,foods,cats)
VALUES('De''Mont','1973-01-12',NULL,'eggroll',4);
```

Если вы вводите значения столбцов запроса сами, просто напишите слово NULL без кавычек. Если же значение для color получается из переменной, уже не так очевидно, как поступить. Необходимо иметь какую-то информацию о значении переменной, чтобы определить, следует ли заключать его в кавычки при формировании запроса.

В вашем распоряжении есть два основных средства обработки специальных символов (кавычек, обратных слэшей) и специальных значений (NULL):

- Если ваш API поддерживает заполнители, используйте их. Этот метод предпочтительнее, так как всю или почти всю работу выполняет сам API, заключая значения в кавычки, расставляя специальные или экранирующие символы там, где это необходимо и, возможно, интерпретируя специальное значение для отображения на NULL без кавычек. Базовая информация о заполнителях приведена в рецепте 2.6; обратитесь к нему, если вы еще этого не сделали.

- Если ваш API содержит функцию заключения в кавычки, используйте ее для преобразования значений данных к безопасной надежной форме, которая подходит для строк запросов.

В оставшейся части раздела описана обработка специальных символов для каждого API. В примерах показано, как вставить в таблицу `profile` запись, значение `name` у которой – `De'Mont`, а значение для столбца `color` – `NULL`. Рассматриваемые приемы обеспечивают обработку любых специальных символов, в том числе и обнаруженных в двоичных данных. (Применяться эти примеры могут не только к предложениям `INSERT`, но и к другим типам запросов, таким как `SELECT`.) Примеры, иллюстрирующие работу с разновидностью двоичных данных – рисунками, представлены в главе 17.

Родственная, но не рассматриваемая в разделе тема – это обратная операция превращения специальных символов в значениях, возвращенных базой данных, для отображения в различных средах. Например, если вы формируете HTML-страницы, содержащие значения, полученные из базы данных, вам необходимо преобразовать символы < и > в такие конструкции HTML, как `<` и `>`, чтобы они корректно отображались. Подробные сведения будут приведены в главе 16.

Perl

DBI поддерживает механизм заполнителей для связывания данных с запросами (см. рецепт 2.6). Используя этот механизм, вы можете добавить в `profile` запись для `De'Mont` при помощи `do()`:

```
my $count = $dbh->do ("INSERT INTO profile (name,birth,color,foods,cats)
                       VALUES(?,?,?,?,?)",
                      undef,
                      "De'Mont", "1973-01-12", undef, "eggroll", 4);
```

Или можно было выбрать `prepare()` плюс `execute()`:

```
my $sth = $dbh->prepare ("INSERT INTO profile (name,birth,color,foods,cats)
                          VALUES(?,?,?,?,?)");
my $count = $sth->execute ("De'Mont", "1973-01-12", undef, "eggroll", 4);
```

В любом случае результирующее множество, сформированное DBI, будет таким:

```
INSERT INTO profile (name,birth,color,foods,cats)
VALUES('De\'Mont','1973-01-12',NULL,'eggroll','4')
```

Обратите внимание на то, что DBI заключает значения данных в кавычки несмотря на то, что их не было вокруг символов-заполнителей `?` в исходной строке запроса. (Механизм заполнителей расставляет кавычки и вокруг числовых значений, но это не страшно, так как сервер MySQL выполняет необходимое преобразование типа для превращения строковых значений в числа.) Отметьте также принятое в DBI соглашение о том, что если с заполнителем связывается `undef`, то DBI помещает в запрос `NULL` и корректно воздерживается от кавычек.

В качестве альтернативы заполнителям DBI предлагает метод quote(). Это метод дескриптора базы данных, поэтому для его использования необходимо наличие открытого соединения с сервером. (Дело в том, что соответствующие правила расстановки кавычек могут быть выбраны только после того, как драйвер будет известен, поскольку разные базы данных могут поддерживать разные правила.) Используем quote() для создания строки запроса, добавляющего новую запись в таблицу profile:

```
my $stmt = sprintf (
            "INSERT INTO profile (name,birth,color,foods,cats)
            VALUES(%s,%s,%s,%s,%s)",
                $dbh->quote ("De'Mont"),
                $dbh->quote ("1973-01-12"),
                $dbh->quote (undef),
                $dbh->quote ("eggroll"),
                $dbh->quote (4));
my $count = $dbh->do ($stmt);
```

Формируемая этим кодом строка запроса идентична построенной при помощи заполнителей. Спецификация формата %s записывается без кавычек, так как при необходимости quote() подставляет их автоматически: значения undef вставляются как NULL и без кавычек, а не-undef добавляются заключенными в кавычки.

PHP

PHP не поддерживает заполнители, но содержит функцию addslashes(), которую можно использовать для преобразования значений в не представляющие опасности для строки запроса. Функция addslashes() экранирует специальные символы, такие как кавычки и обратный слэш, но не заключает значения в кавычки, эту операцию вы должны проделать сами. Кроме того, необходимо как-то обозначить значения NULL; давайте попробуем применить unset() для того, чтобы заставить переменную иметь «отсутствие значения» (что-то типа значения undef в Perl). Вот PHP-код для вставки в таблицу profile записи о De'Mont:

```
unset ($null);  # создать значение "null"
$stmt = sprintf ("
            INSERT INTO profile (name,birth,color,foods,cats)
            VALUES('%s','%s','%s','%s','%s')",
                addslashes ("De'Mont"),
                addslashes ("1973-01-12"),
                addslashes ($null),
                addslashes ("eggroll"),
                addslashes (4));
$result_id = mysql_query ($stmt, $conn_id);
```

В примере спецификация формата %s в строке запроса приведена заключенной в кавычки, поскольку addslashes() не добавляет их. К сожалению, результирующее множество выглядит не так, как хотелось бы:

```
INSERT INTO profile (name,birth,color,foods,cats)
VALUES('De\'Mont','1973-01-12','','eggroll','4')
```

Кавычка в поле `name` корректно экранирована, а вот переданное для столбца `color` значение «null» (неустановленное) превратилось в пустую строку, а не в `NULL`. Подправим положение, написав вспомогательную функцию `sql_quote()`, которую и будем использовать вместо `addslashes()`. Функция `sql_quote()` похожа на `addslashes()`, но возвращает `NULL` (без кавычек) для неустановленных значений, а также добавляет кавычки вокруг остальных значений. Вот как она выглядит:

```
function sql_quote ($str)
{
    return (isset ($str) ? "'" . addslashes ($str) . "'" : "NULL");
}
```

Поскольку функция `sql_quote()` сама расставляет кавычки вокруг значений данных (если они там нужны), можно убрать кавычки, в которые заключена спецификация формата в строке запроса, и переписать предложение `INSERT` так:

```
unset ($null);  # создать значение "null"
$stmt = sprintf ("
            INSERT INTO profile (name,birth,color,foods,cats)
            VALUES(%s,%s,%s,%s,%s)",
                sql_quote ("De'Mont"),
                sql_quote ("1973-01-12"),
                sql_quote ($null),
                sql_quote ("eggroll"),
                sql_quote (4));
$result_id = mysql_query ($stmt, $conn_id);
```

После того как выполнены описанные изменения, значение `$stmt` содержит значение `NULL`, корректно не заключенное в кавычки:

```
INSERT INTO profile (name,birth,color,foods,cats)
VALUES('De\'Mont','1973-01-12',NULL,'eggroll','4')
```

Если вы работаете в PHP 4, то можете использовать дополнительные возможности обработки значений `NULL` и специальных символов. Во-первых, PHP 4 имеет специальное значение `NULL`, совпадающее с неустановленным значением, поэтому вы можете использовать его в предыдущем примере, формирующем предложение `INSERT` вместо `$null`. (Однако если вы хотите написать код, который работал бы и в PHP 3, и в PHP 4, используйте неустановленную переменную `$null`.) Во-вторых, начиная с версии PHP 4.0.3 можно использовать в качестве альтернативы `addslashes()` функцию `mysql_escape_string()`, созданную на основе одноименной функции API языка C для MySQL. Например, можно переписать `sql_quote()` так:

```
function sql_quote ($str)
{
    return (isset ($str) ? "'" . mysql_escape_string ($str) . "'" : "NULL");
}
```

Если же вы хотите написать код, который использовал бы функцию mysql_escape_string() при ее наличии, а противном случае возвращался бы к addslashes(), поступите так:

```
function sql_quote ($str)
{
    if (!isset ($str))
        return ("NULL");
    $func = function_exists ("mysql_escape_string")
            ? "mysql_escape_string"
            : "addslashes";
    return ("'" . $func ($str) . "'");
}
```

Какую бы версию sql_quote() вы ни выбрали, эта процедура достойна включения в библиотечный файл. В последующих рецептах предполагается, что PHP-сценарии имеют доступ к такой функции. Она входит в файл *Cookbook_Utils.php*, расположенный в каталоге *lib* дистрибутива recipes. Чтобы работать с этим файлом, поместите его в каталог, в котором находится файл *Cookbook.php*, и ссылайтесь на него из сценария так:

```
include "Cookbook_Utils.php";
```

Python

Как говорилось в рецепте 2.6, Python поддерживает механизм заполнителей, который можно применять для обработки специальных символов в значениях данных. Чтобы добавить в таблицу profile запись о De'Mont, напишем такой код:

```
try:
    cursor = conn.cursor ()
    cursor.execute ("""
            INSERT INTO profile (name,birth,color,foods,cats)
            VALUES(%s,%s,%s,%s,%s)
            """, ("De'Mont", "1973-01-12", None, "eggroll", 4))
    print "%d row was inserted" % cursor.rowcount
except:
    print "Oops, the query failed"
```

Механизм связывания параметров добавляет кавычки вокруг значений там, где это необходимо. DB-API рассматривает None как логический эквивалент SQL-значения NULL, так что вы можете связать заполнитель с None, и получить NULL в строке запроса. Запрос, отправляемый на сервер предыдущим вызовом execute(), выглядит так:

```
INSERT INTO profile (name,birth,color,foods,cats)
VALUES('De\'Mont','1973-01-12',NULL,'eggroll',4)
```

В MySQLdb версии 0.9.1 или выше есть еще один способ обработки специальных символов – использование метода literal(). Создадим предложение INSERT для De'Mont с помощью literal():

```
try:
    cursor = conn.cursor ()
    str = """
            INSERT INTO profile (name,birth,color,foods,cats)
            VALUES(%s,%s,%s,%s,%s)
            """ % \
                (conn.literal ("De'Mont"), \
                conn.literal ("1973-01-12"), \
                conn.literal (None), \
                conn.literal ("eggroll"), \
                conn.literal (4))
    cursor.execute (str)
    print "%d row was inserted" % cursor.rowcount
except:
    print "Oops, the query failed"
```

Java

Java предоставляет механизм заполнителей, с помощью которого можно обрабатывать специальные символы в значениях данных (см. рецепт 2.6). Чтобы добавить в таблицу `profile` запись о De'Mont, создайте подготовленное предложение, свяжите с ним значения данных и выполните предложение:

```
PreparedStatement s;
int count;
s = conn.prepareStatement (
        "INSERT INTO profile (name,birth,color,foods,cats)"
        + " VALUES(?,?,?,?,?)");
s.setString (1, "De'Mont");
s.setString (2, "1973-01-12");
s.setNull (3, java.sql.Types.CHAR);
s.setString (4, "eggroll");
s.setInt (5, 4);
count = s.executeUpdate ();
s.close ();     // закрыть предложение
```

Каждый вызов связывания значения (value-binding) выбирается в соответствии с типом данных столбца, с которым связывается значение: `setString()` — для связывания строки символов со столбцом `name`, `setInt()` — для связывания целого числа со столбцом `cats` и т. д. (На самом деле я немного сжульничал, использовав `setString()` для того, чтобы рассматривать значение данных `birth` как строку символов.) Вызовы `set`*`XXX`*`()` добавляют кавычки вокруг тех значений, которым это необходимо, поэтому кавычки вокруг символов-заполнителей `?` в строке запроса не нужны. Отличие JDBC от других API заключается в том, что вы не указываете какое-либо специальное значение для связывания с заполнителем значения `NULL` (подобного значению `undef` в Perl или `None` в Python). Вместо этого вызывается специальный метод `setNull()`, второй аргумент которого определяет тип столбца (`java.sql.Types.CHAR` для строки символов, `java.sql.Types.INTEGER` для целых чисел и т. д.).

Чтобы обеспечить единообразие вызовов связывания значений, определим вспомогательную функцию `bindParam()`, принимающую объект `Statement`, позицию заполнителя и значение данных. Тогда одна функция будет использоваться для связывания любых значений. Можно даже заключить следующее соглашение: передача Java-значения `null` связывает с запросом SQL-значение `NULL`. Перепишем предыдущий пример, используя `bindParam()`:

```
PreparedStatement s;
int count;
s = conn.prepareStatement (
        "INSERT INTO profile (name,birth,color,foods,cats)"
        + " VALUES(?,?,?,?,?)");
bindParam (s, 1, "De'Mont");
bindParam (s, 2, "1973-01-12");
bindParam (s, 3, null);
bindParam (s, 4, "eggroll");
bindParam (s, 5, 4);
count = s.executeUpdate ();
s.close ();     // закрыть предложение
```

Специальные символы в именах баз данных, таблиц и столбцов

В версиях MySQL 3.23.6 и выше можно заключать имена баз данных, таблиц и столбцов в обратные одиночные кавычки (backquotes). Благодаря этой возможности можно включать в такие имена символы, которые обычно запрещено там использовать. Например, по умолчанию пробелы в именах запрещены:

```
mysql> CREATE TABLE my table (i INT);
ERROR 1064 at line 1: You have an error in your SQL syntax
near 'table (i INT)' at line 1
```

Чтобы включить пробел в название, защитите его обратными одиночными кавычками:

```
mysql> CREATE TABLE `my table` (i INT);
Query OK, 0 rows affected (0.04 sec)
```

Вы получаете больше свободы в выборе имен, но корректно писать программы становится труднее. (Если вы используете имя, заключенное в кавычки, необходимо использовать обратные кавычки при каждом вхождении данного имени в код.) Это немного усложняет жизнь, поэтому я лично не пользуюсь такими именами и вам не советую. Тем же, кто не захочет прислушаться к совету, предложу следующее: определить переменную, хранящую имя, включая обратные кавычки, затем использовать эту переменную для ссылок на имя. Например, в Perl можно написать так:

```
$tbl_name = "`my table`";
$dbh->do ("DELETE FROM $tbl_name");
```

Для исполнения bindParam() требуется множество функций, поскольку ее третий аргумент может иметь произвольный тип, и нам потребуется по одной функции для каждого типа. В следующем коде приведены версии для обработки целых и строковых значений (строковая версия обрабатывает null и связывает его с NULL):

```
public static void bindParam (PreparedStatement s, int pos, int val)
{
    try
    {
        s.setInt (pos, val);
    }
    catch (Exception e) { /* перехватывать и игнорировать */ }
}

public static void bindParam (PreparedStatement s, int pos, String val)
{
    try
    {
        if (val == null)
            s.setNull (pos, java.sql.Types.CHAR);
        else
            s.setString (pos, val);
    }
    catch (Exception e) { /* перехватывать и игнорировать */ }
}
```

Для обработки других типов данных напишите версии bindParam(), принимающие аргументы соответствующего типа.

2.8. Обработка значений NULL в результирующих множествах

Задача

Результирующее множество содержит значения NULL, но вы не знаете, как выяснить, где именно они находятся.

Решение

Вероятно, ваш API имеет некое значение, условно представляющее NULL. Узнайте, что это за значение и как проводить проверку на него.

Обсуждение

В рецепте 2.7 рассказывалось о том, как ссылаться на значения NULL при отправке запросов *на* сервер. В этом разделе мы поговорим о том, как распознавать и обрабатывать значения NULL, полученные *от* базы данных. Вообще говоря, необходимо знать, какие специальные значения API сопоставляет значениям NULL или какие функции вызываются (табл. 2.3):

Таблица 2.3. Выявление значений NULL

Язык	Значение (функция), определяющее NULL
Perl	undef
PHP	Неустановленное значение
Python	None
Java	wasNull()

В следующих разделах будет представлено чрезвычайно простое приложение, определяющее значения NULL. В примере извлекается результирующее множество и выводятся все его значения, при этом значения NULL отображаются в виде печатаемой строки "NULL".

Чтобы убедиться в том, что в таблице profile есть строка, которая содержит значения NULL, выполните предложение INSERT в *mysql*. Затем выполните SELECT, чтобы посмотреть, присутствуют ли в возвращенных строках ожидаемые значения:

```
mysql> INSERT INTO profile (name) VALUES('Juan');
mysql> SELECT * FROM profile WHERE name = 'Juan';
+----+------+-------+-------+-------+------+
| id | name | birth | color | foods | cats |
+----+------+-------+-------+-------+------+
| 11 | Juan | NULL  | NULL  | NULL  | NULL |
+----+------+-------+-------+-------+------+
```

Столбец id может содержать другое число, но все остальные значения должны быть именно такими.

Perl

В сценариях Perl DBI значению NULL соответствует undef. Такие значения легко выявить при помощи функции defined(), и это необходимо сделать, если вы используете опцию *-w* в строке #!, открывающей ваш сценарий. Иначе при обращении к значениям undef Perl выдаст следующее сообщение:

```
Use of uninitialized value
```

Чтобы предотвратить появление такого сообщения, перед обработкой значений столбцов, которые могут содержать undef, проверьте их с помощью defined(). Приведенный ниже код выбирает несколько столбцов из profile и выводит "NULL" для любых неопределенных значений каждой строки. В этом случае значения NULL явно присутствуют в выводе, и никакие предупреждения не появляются:

```
my $sth = $dbh->prepare ("SELECT name, birth, foods FROM profile");
$sth->execute ();
while (my $ref = $sth->fetchrow_hashref ())
{
    printf "name: %s, birth: %s, foods: %s\n",
                defined ($ref->{name}) ? $ref->{name} : "NULL",
```

```
                defined ($ref->{birth}) ? $ref->{birth} : "NULL",
                defined ($ref->{foods}) ? $ref->{foods} : "NULL";
}
```

К сожалению, проведенная проверка очень тяжеловесна, и чем больше столбцов, тем более громоздкой она будет. Вместо этого перед выводом можно проверять и устанавливать неопределенные значения в цикле. Тогда объем кода для выполнения проверок будет оставаться неизменным и не будет зависеть от количества исследуемых столбцов. В цикле нет ссылок на имена конкретных столбцов, поэтому его можно копировать и вставлять в другие программы или использовать в качестве основы для библиотечной функции:

```
my $sth = $dbh->prepare ("SELECT name, birth, foods FROM profile");
$sth->execute ();
while (my $ref = $sth->fetchrow_hashref ())
{
    foreach my $key (keys (%{$ref}))
    {
        $ref->{$key} = "NULL" unless defined ($ref->{$key});
    }
    printf "name: %s, birth: %s, foods: %s\n",
                $ref->{name}, $ref->{birth}, $ref->{foods};
}
```

Если вы выбираете строки в массив, а не в хеш, то можете использовать для преобразования любых значений `undef` функцию `map()`:

```
my $sth = $dbh->prepare ("SELECT name, birth, foods FROM profile");
$sth->execute ();
while (my @val = $sth->fetchrow_array ())
{
    @val = map { defined ($_) ? $_ : "NULL" } @val;
    printf "name: %s, birth: %s, foods: %s\n",
                $val[0], $val[1], $val[2];
}
```

PHP

PHP представляет `NULL` в результирующих множествах как неустановленные значения, поэтому для распознавания `NULL` в результирующем множестве можно использовать функцию `isset()`. Покажем, как это сделать:

```
$result_id = mysql_query ("SELECT name, birth, foods FROM profile", $conn_id);
if (!$result_id)
    die ("Oops, the query failed\n");
while ($row = mysql_fetch_row ($result_id))
{
    while (list ($key, $value) = each ($row))
    {
        if (!isset ($row[$key]))    # проверка на неустановленные значения
            $row[$key] = "NULL";
    }
    print ("name: $row[0], birth: $row[1], foods: $row[2]\n");
```

```
}
mysql_free_result ($result_id);
```

В PHP 4 есть специальное значение NULL, подобное неустановленному значению. Если ваши сценарии предназначены для работы в PHP 4, то можно проводить проверку на NULL так:

```
if ($row[$key] === NULL)    # проверка на PHP-значения NULL
    $row[$key] = "NULL";
```

Заметим, что тут использован оператор «тройного равенства» ===, который в PHP 4 означает «в точности равно». Обычный оператор сравнения == тут не подходит, так как воспринимает значение NULL, пустую строку и 0 как равные.

Python

Программы DB-API Python для представления значений NULL в результирующих множествах используют значение None. Рассмотрим пример выявления NULL:

```
try:
    cursor = conn.cursor ()
    cursor.execute ("SELECT name, birth, foods FROM profile")
    for row in cursor.fetchall ():
        row = list (row)    # преобразовать неизменяемый кортеж в изменяемый список
        for i in range (0, len (row)):
            if row[i] == None:  # значение столбца равно NULL?
                row[i] = "NULL"
        print "name: %s, birth: %s, foods: %s" % (row[0], row[1], row[2])
    cursor.close ()
except:
    print "Oops, the query failed"
```

Внутренний цикл проверяет на NULL значения столбцов, отыскивая None и выполняя преобразование в строку "NULL". Обратите внимание на то, что row конвертируется в изменяемый объект перед началом цикла: дело в том, что fetchall() возвращает строки как значения последовательности, которые не изменяемы (доступны только для чтения).

Java

В программах JDBC, если есть шанс, что в столбце результирующего множества может появиться значение NULL, лучше проводить явную проверку на такие значения. Для этого выберите значение и вызовите функцию wasNull(), которая возвращает значение «истина», если столбец содержит NULL, и «ложь» в противном случае. Например:

```
Object obj = rs.getObject (index);
if (rs.wasNull ())
{ /* проверка на NULL */ }
```

В данном случае использован вызов getObject(), все вышесказанное верно для любого вызова get*XXX*().

Рассмотрим пример, выводящий каждую строку результирующего множества в виде списка значений, разделенных запятыми, при этом для значений NULL будет выводиться строка "NULL":

```
Statement s = conn.createStatement ();
s.executeQuery ("SELECT name, birth, foods FROM profile");
ResultSet rs = s.getResultSet ();
ResultSetMetaData md = rs.getMetaData ();
int ncols = md.getColumnCount ();
while (rs.next ())  // цикл по строкам результирующего множества
{
    for (int i = 0; i < ncols; i++) // цикл по столбцам
    {
        String val = rs.getString (i+1);
        if (i > 0)
            System.out.print (", ");
        if (rs.wasNull ())
            System.out.print ("NULL");
        else
            System.out.print (val);
    }
    System.out.println ();
}
rs.close ();    // закрыть результирующее множество
s.close ();     // закрыть предложение
```

2.9. Создание объектно-ориентированного интерфейса MySQL для PHP

Задача

Необходим способ создания сценариев PHP, который бы не так сильно зависел от собственных функций PHP, связанных с MySQL.

Решение

Используйте один из доступных абстрактных интерфейсов или напишите собственный.

Обсуждение

Может быть, вы обратили внимание на то, что операции Perl, Python и Java, осуществляющие соединение с сервером MySQL, возвращают значение, которое позволяет обрабатывать запросы, применяя объектно-ориентированный подход. Perl предоставляет дескрипторы базы данных и предложения, у Python есть объекты курсора и соединения, у Java имеются объекты для всего: для соединений, предложений, результирующих множеств и метаданных. Все эти объектно-ориентированные интерфейсы используют двухуровневую архитектуру.

Верхний уровень обеспечивает независящие от базы данных методы, реализующие доступ к БД. Эти методы являются переносимыми, то есть не имеет значения, с какой системой управления базами данных вы работаете: это может быть MySQL, PostgreSQL, Oracle или что-то еще. Нижний уровень состоит из набора драйверов, каждый из которых реализует особенности конкретной СУБД. Двухуровневая архитектура позволяет приложению использовать абстрактный интерфейс, не связанный со спецификой доступа к определенному серверу базы данных. Улучшается переносимость программ – для работы с какой-то другой базой данных вы просто выбираете другой драйвер нижнего уровня. По крайней мере, так это выглядит в теории. На практике же идеальная переносимость может оказаться чем-то недостижимым:

- Методы интерфейса, предоставляемые на верхнем уровне архитектуры, остаются неизменными вне зависимости от того, какой драйвер используется, но существует возможность создавать предложения SQL, которые содержат конструкции, поддерживаемые только каким-то определенным сервером. В случае с MySQL хорошим примером может быть предложение `SHOW`, предоставляющее информацию о структуре базы данных и таблицы. Если отправить это предложение на не-MySQL-сервер, вероятно, возникнет ошибка.
- Драйверы нижнего уровня часто расширяют абстрактный интерфейс для того, чтобы добраться до специальных возможностей конкретной базы данных. Например, MySQL-драйвер для DBI делает доступным (в виде атрибута дескриптора базы данных) последнее значение `AUTO_INCREMENT`, так что вы можете обратиться к нему с помощью `$dbh->{mysql_insertid}`. С одной стороны, упрощается создание программы на начальном этапе, с другой – ухудшается переносимость, и если программу нужно будет использовать с другой СУБД, придется вносить в нее изменения.

Несмотря на эти факторы, несколько ухудшающие переносимость, преимущество двухуровневой архитектуры для программистов Perl, Python, и Java весьма значительно. Было бы замечательно, если бы аналогичный подход можно было использовать при написании сценариев PHP, но сам PHP не поддерживает его. Интерфейс PHP для MySQL состоит из набора функций, которые по сути своей непереносимы, так как все они называются `mysql_`*`xxx`*`()`. Чтобы обойти данное ограничение, вы можете создать собственный механизм абстракции базы данных.

Этим мы и займемся. В разделе будет показано, как написать объектно-ориентированный интерфейс PHP, скрывающий большинство особенностей MySQL и относительно независимый от выбора базы данных, по крайней мере, более независимый, чем интерфейс, состоящий из набора функций. Интерфейс будет создаваться специально для MySQL, но если вы захотите адаптировать его для работы с другими базами данных, то сможете сделать это, предоставив другой набор методов базового класса.

Если же вы хотите писать не зависящие от выбора базы данных сценарии PHP, но не хотите разрабатывать собственный интерфейс, используйте интерфейс третьих фирм. Один такой класс доступа к базе данных входит

в стандартную библиотеку классов PHP – PEAR (PHP Extension and Add-on Repository), включенную в текущие редакции PHP 4.

Далее будет рассказано, как написать класс `MySQL_Access`, реализующий объектно-ориентированный интерфейс для MySQL, и класс `Cookbook_DB_Access`, который надстраивается над `MySQL_Access` и автоматически поставляет значения по умолчанию для соединения с базой данных `cookbook`. (Если вы не знакомы с классами PHP, то можете обратиться за базовой информацией по теме к главе «Classes and objects» («Классы и объекты») PHP Manual.) Основная цель этого объектно-ориентированного интерфейса – упростить работу с MySQL, сократив количество операций, которые ваши сценарии должны выполнять явно:

- При запуске запроса без предварительного открытия соединения интерфейс автоматически устанавливает соединение с сервером MySQL. Конечно, необходимо как-то задать параметры соединения, но далее вы узнаете, что и эту операцию можно сделать автоматической.
- Интерфейс обеспечивает автоматический контроль ошибок в вызовах MySQL. Это удобнее, чем самостоятельная проверка, к тому же решается одна из наиболее распространенных проблем сценариев PHP: невозможность постоянной проверки ошибок базы данных. Поведение по умолчанию таково: в случае ошибки выход с выводом сообщения, но вы можете изменить его, если хотите сами обрабатывать ошибки.
- Когда при извлечении строк результирующего множества вы доходите до конца, класс автоматически освобождает множество.

Объектно-ориентированный интерфейс содержит также метод заключения значений в кавычки для их безопасного использования в запросах и поддерживает механизм заполнителей, так что вы сами (если не хотите) можете вообще не заниматься кавычками. Все эти возможности недоступны в собственном интерфейсе PHP, основанном на функциях.

В следующем примере показано, как использование объектно-ориентированного интерфейса изменяет манеру написания сценариев PHP, обращающихся к MySQL. Сценарий, основанный на вызовах собственных функций PHP, обычно обращается к MySQL примерно так:

```
if (!($conn_id = mysql_connect ("localhost", "cbuser", "cbpass")))
    die ("Cannot connect to database\n");
if (!mysql_select_db ("cookbook", $conn_id))
    die ("Cannot select database\n");
$query = "UPDATE profile SET cats=cats+1 WHERE name = 'Fred'";
$result_id = mysql_query ($query, $conn_id);
if (!$result_id)
    die (mysql_error ($conn_id));
print (mysql_affected_rows ($conn_id) . " rows were updated\n");
$query = "SELECT id, name, cats FROM profile";
$result_id = mysql_query ($query, $conn_id);
if (!$result_id)
    die (mysql_error ($conn_id));
```

```
while ($row = mysql_fetch_row ($result_id))
    print ("id: $row[0], name: $row[1], cats: $row[2]\n");
mysql_free_result ($result_id);
```

На первом этапе работы по удалению некоторого количества приведенного кода заменим первые несколько строк на вызов функции `cookbook_connect()` из библиотечного файла PHP *Cookbook.php*, созданного в рецепте 2.3. Эта функция инкапсулирует код установления соединения с базой данных и выбора базы данных для работы:

```
include "Cookbook.php";
$conn_id = cookbook_connect ();
$query = "UPDATE profile SET cats=cats+1 WHERE name = 'Fred'";
$result_id = mysql_query ($query, $conn_id);
if (!$result_id)
    die (mysql_error ($conn_id));
print (mysql_affected_rows ($conn_id) . " rows were updated\n");
$query = "SELECT id, name, cats FROM profile";
$result_id = mysql_query ($query, $conn_id);
if (!$result_id)
    die (mysql_error ($conn_id));
while ($row = mysql_fetch_row ($result_id))
    print ("id: $row[0], name: $row[1], cats: $row[2]\n");
mysql_free_result ($result_id);
```

А объектно-ориентированный интерфейс может выполнить дальнейшую инкапсуляцию кода и дополнительно сократить объем сценария за счет устранения необходимости явного установления соединения, явной проверки ошибок и явного закрытия результирующего множества. Все эти операции могут выполняться автоматически:

```
include "Cookbook_DB_Access.php";
$conn = new Cookbook_DB_Access;
$query = "UPDATE profile SET cats=cats+1 WHERE name = 'Fred'";
$conn->issue_query ($query);
print ($conn->num_rows . " rows were updated\n");
$query = "SELECT id, name, cats FROM profile";
$conn->issue_query ($query);
while ($row = $conn->fetch_row ())
    print ("id: $row[0], name: $row[1], cats: $row[2]\n");
```

Объектный интерфейс упрощает работу с MySQL, уменьшая тот объем кода, который необходимо написать при создании нового сценария, и это не единственное его достоинство. Например, он может помочь в переводе рецепта. Предположим, что программа в одной из следующих глав приведена на Perl, а вам необходимо использовать ее в PHP, и на веб-сайте нашего «Сборника рецептов» нет версии для PHP. DBI Perl является объектно-ориентированным, поэтому вы, вероятно, обнаружите, что сценарий Perl проще преобразовывать в объектно-ориентированный сценарий PHP, чем в сценарий, основанный на функциях.

Общее представление о классе

При реализации объектного интерфейса использовались рецепты PHP и приемы, рассмотренные ранее в главе, так что вам придется их освоить. Например, объектный интерфейс должен уметь устанавливать соединение с сервером и обрабатывать запросы. Кроме того, интерфейс будет инкапсулирован во включаемый (библиотечный) файл, чтобы его можно было без труда использовать из различных сценариев PHP.

Обсуждаемый интерфейс будет работать только с PHP версии 4. Собственные же функции PHP для MySQL работают и в PHP 3, и в PHP 4. Ограничение объясняется применением конструкций, которые не были доступны или работали некорректно в PHP 3. Если точнее, интерфейс предполагает наличие предложения `include once` и PHP-значения `NULL`. Кроме того, интерфейс полагает, что `count()` корректно подсчитывает неустановленные значения в массивах, что верно только для PHP 4.

В реализацию вовлечены два класса. Первый из них – общий базовый класс `MySQL_Access`, предоставляющий переменные и методы для использования MySQL. Второй – это порожденный класс `Cookbook_DB_Access`, который имеет доступ ко всему в базовом классе, но автоматически устанавливает параметры соединения именно для базы данных `cookbook`, чтобы нам не приходилось это делать самим. В качестве альтернативы можно работать всего с одним классом и жестко зашить параметры соединения прямо в него. Но благодаря наличию общего базового класса интерфейс удобнее использовать для сценариев, обращающихся к базе данных, отличной от `cookbook`. (Для таких сценариев вы просто пишете другой порожденный класс, использующий базовый, но предоставляющий другой набор параметров соединения.)

Определение класса PHP начинается со строки `class`, указывающей имя класса, затем определяются переменные и методы, входящие в класс. Структура базового класса `MySQL_Access` выглядит так:

```
class MySQL_Access
{
    var $host_name = "";
    var $user_name = "";
    var $password = "";
    var $db_name = "";
    var $conn_id = 0;
    var $errno = 0;
    var $errstr = "";
    var $halt_on_error = 1;
    var $query_pieces = array ();
    var $result_id = 0;
    var $num_rows = 0;
    var $row = array ();

# ... определения методов ...

} # конец MySQL_Access
```

Определение класса содержит несколько переменных, используемых в дальнейшем:

- Первые несколько переменных хранят параметры соединения с сервером MySQL (`$host_name`, `$user_name`, `$password` и `$db_name`). Сначала они пусты и должны быть заданы до совершения попытки установления соединения.
- После того как соединение установлено, идентификатор соединения сохраняется в `$conn_id`. Начальное значение переменной `0` означает «отсутствие соединения». Так экземпляр класса получает возможность определить, открыто ли еще соединение с базой данных.
- Переменные `$errno` и `$errstr` хранят информацию об ошибках, класс устанавливает их после каждой операции MySQL, чтобы обозначить успешное или неуспешное выполнение операции. Начальные значения – `0` и пустая строка – означают «отсутствие ошибок». Для ошибок, возникших не в результате взаимодействия с сервером, `$errno` устанавливается в `-1` (ненулевое значение, которое никогда не используется в MySQL). Такая ошибка может возникнуть, например, если вы помещаете символы-заполнители в строку запроса, но при связывании значений данных со строкой запроса указываете неправильные позиции. Класс выявляет ошибку, не отправляя ничего на сервер.
- Переменная `$halt_on_error` определяет, следует ли завершать сценарий в случае возникновения ошибки. Действие по умолчанию – завершать. Сценарии, самостоятельно проводящие обработку ошибок, могут установить эту переменную в ноль.
- Переменная `$query_pieces` используется для хранения строки запроса для подготовленных предложений и связывания параметров. (Позже я поясню, почему эта переменная является массивом.)
- Переменные `$result_id`, `$num_rows` и `$row` используются при обработке результирующего множества для хранения его идентификатора, количества строк, возвращенных или обработанных запросом, и текущей строки результирующего множества.

Функции конструктора класса PHP

В PHP в определении класса можно указать функцию конструктора, которая будет автоматически вызываться при создании нового экземпляра класса. Для этого функции необходимо дать имя, совпадающее с именем класса. Это можно сделать, например, чтобы инициализировать переменные объекта в непостоянные значения. (В PHP 4 переменные объектов можно инициализировать только константами.) У класса `MySQL_Access` нет конструктора, так как все его переменные имеют постоянное начальное значение.

В строку «определения методов» в конце описания класса помещаются функции, которые соединяются с сервером MySQL, осуществляют проверку на ошибки, выдают запросы и т. д. Вскоре мы заполним эту часть, пока же

поговорим о том, как может использоваться класс. Можно поместить код класса во включаемый файл *MySQL_Access.php* и поместить этот файл в каталог, который PHP просматривает в поиске включаемых файлов (например, в каталог */usr/local/apache/lib/php*, обсуждаемый в рецепте 2.3). Затем можно ссылаться на этот файл при помощи предложения `include`, создавая экземпляр класса для получения объекта соединения `$conn` и задавая параметры соединения для этого объекта:

```
include "MySQL_Access.php";          # включить класс MySQL_Access
$conn = new MySQL_Access;            # создать новый объект класса
$conn->host_name = "localhost";      # инициализировать параметры соединения
$conn->db_name = "cookbook";
$conn->user_name = "cbuser";
$conn->password = "cbpass";
```

Однако такое использование класса не очень удобно для соединения с сервером, поскольку приходится писать все эти операторы присваивания для параметров соединения. Удобнее работать с порожденным классом, использующим базовый, так как порожденный класс может устанавливать параметры автоматически. Давайте создадим класс `Cookbook_DB_Access`, расширяющий `MySQL_Access` за счет предоставления параметров для соединения с базой данных `cookbook`. Теперь можно написать сценарий, открывающий доступ к базе данных `cookbook` всего в двух строках:

```
include "Cookbook_DB_Access.php";
$conn = new Cookbook_DB_Access;
```

Реализация класса `Cookbook_DB_Access` достаточно проста. Создадим файл *Cookbook_DB_Access.php*, который будет выглядеть так:

```
include_once "MySQL_Access.php";

class Cookbook_DB_Access extends MySQL_Access
{
    # override default class variable values
    var $host_name = "localhost";
    var $user_name = "cbuser";
    var $password = "cbpass";
    var $db_name = "cookbook";
}
```

В строке `class` указывается имя класса, `Cookbook_DB_Access`, а инструкция `extends` указывает на то, что базовым для него является класс `MySQL_Access`. Такое расширение класса называется созданием подкласса (subclass), или порожденного (derived) класса, из базового. Определение нового класса тривиально и содержит только присваивания переменных для параметров соединения. Они подменяют собой значения (пустые) из базового класса. В результате при создании экземпляра `Cookbook_DB_Access` вы получаете объект, который всем похож на объект `MySQL_Access`, за исключением того, что параметры соединения автоматически установлены для соединения с базой данных `cookbook`.

Теперь должно быть понятно, почему мы оставляем параметры соединения в классе `MySQL_Access` пустыми (empty), а не задаем установку соединения с базой данных `cookbook`. Оставляя параметры соединения пустыми, мы создаем более общий класс, который можно расширять для использования с любым количеством баз данных за счет создания различных порожденных классов. Одним из таких классов является `Cookbook_DB_Access`. Если вы пишете сценарии, которые будут работать с другой базой данных, создайте другой порожденный расширяющий класс, предоставляющий соответствующие параметры соединения для вашей базы данных. Затем сделайте так, чтобы сценарии использовали второй порожденный класс вместо *Cookbook_DB_Access.php*.

Кстати, поскольку файл *Cookbook_DB_Access.php* включает в себя *MySQL_Access.php*, вам не нужно включать его повторно. Если сценарии включают *Cookbook_DB_Access.php*, они «бесплатно» получают и *MySQL_Access.php*. Предложение `include_once` используется вместо `include` во избежание проблем с дублированием предложений в том случае, если сценарию все-таки придется включить и *MySQL_Access.php*.

Подключение и отключение

Пришло время написать для базового класса `MySQL_Access` методы, которые взаимодействуют с MySQL. Они попадут в исходный файл *MySQL_Access.php*. Сначала следует создать метод `connect()`, устанавливающий соединение с сервером MySQL:

```
function connect ()
{
    $this->errno = 0;           # очистить переменные ошибок
    $this->errstr = "";
    if ($this->conn_id == 0)    # установить соединение, если оно еще не установлено
    {
        $this->conn_id = @mysql_connect ($this->host_name,
                                         $this->user_name,
                                         $this->password);
        # использовать mysql_errno()/mysql_error(), если они работают для
        # ошибок соединения, иначе использовать $php_errormsg
        if (!$this->conn_id)
        {
            # mysql_errno() возвращает не ноль, если обрабатывает ошибки соединения
            if (mysql_errno ())
            {
                $this->errno = mysql_errno ();
                $this->errstr = mysql_error ();
            }
            else
            {
                $this->errno = -1;  # использовать другие значения
                $this->errstr = $php_errormsg;
            }
            $this->error ("Cannot connect to server");
            return (FALSE);
```

```
        }
        # выбрать базу данных, если она была указана
        if (isset ($this->db_name) && $this->db_name != "")
        {
            if (!@mysql_select_db ($this->db_name, $this->conn_id))
            {
                $this->errno = mysql_errno ();
                $this->errstr = mysql_error ();
                $this->error ("Cannot select database");
                return (FALSE);
            }
        }
    }
    return ($this->conn_id);
}
```

Метод `connect()` проверяет, не установлено ли уже соединение, и в случае отрицательного ответа пытается установить его. Метод `connect()` делает это так, что другие методы, которым требуется соединение с сервером, могут просто вызвать `connect()`, чтобы быть уверенными в том, что соединение установлено. В частности, можно при создании метода, выдающего запросы, вызывать `connect()` перед отправкой запроса на сервер. Получается, что сценарию остается только создать объект класса и приступить к запуску запросов, а методы класса будут автоматически заниматься установкой соединения вместо вас. Если класс создается подобным образом, то сценариям, использующим класс, уже не нужно явно устанавливать соединение.

В случае успешного установления соединения, или если оно уже было открыто, `connect()` возвращает идентификатор соединения (значение не-`FALSE`). При возникновении ошибки `connect()` вызывает `error()`, и далее возможны два варианта:

- Если переменная `$halt_on_error` установлена, то `error()` выводит сообщение и завершает работу сценария.
- В противном случае `error()` ничего не делает и возвращается в `connect()`, который возвращает `FALSE`.

Отметьте тот факт, что если соединение установить не удалось, метод `connect()` пытается использовать `mysql_errno()` и `mysql_error()`, если это версии PHP 4.0.6 и выше, которые возвращают полезную для `mysql_connect()` информацию об ошибках (см. рецепт 2.2). Иначе он устанавливает `$errno` в `-1`, а `$errstr` — в `$php_errormsg`.

Существует и метод `disconnect()` для явного закрытия соединения. (Если не закрыть соединения явно, PHP закроет его по завершении работы сценария.) Метод вызывает `mysql_close()`, если соединение открыто:

```
function disconnect ()
{
    if ($this->conn_id != 0)    # есть открытое соединение; закрыть его
    {
        mysql_close ($this->conn_id);
```

```
            $this->conn_id = 0;
        }
        return (TRUE);
    }
```

Обработка ошибок

Методы `MySQL_Access` обрабатывают ошибки, вызывая метод `error()`. Метод ведет себя по-разному в зависимости от значения переменной `$halt_on_error`. Если значение `$halt_on_error` истинно (не равно нулю), `error()` выводит сообщение и завершает работу. Так метод ведет себя по умолчанию, то есть если вы не хотите проверять ошибки, то можете этого не делать. Если отключить переменную `$halt_on_error`, установив ее в ноль, `error()` просто вернет ее вызывающему методу, а тот в свою очередь передаст код возврата ошибки своему вызывающему методу. То есть код обработки ошибок внутри `MySQL_Access` обычно выглядит так:

```
if (some error occurred)
{
    $this->error ("some error occurred");
    return (FALSE);
}
```

Если переменная `$halt_on_error` включена, то в случае ошибки вызывается `error()` и завершает сценарий. В противном случае выполняется следующее за ним предложение `return()`.

Чтобы написать код, проводящий собственную проверку ошибок, отключите `$halt_on_error`. В этом случае вы также можете захотеть воспользоваться переменными `$errno` и `$errstr`, которые хранят числовой код и текстовое описание ошибки MySQL. В следующем примере показано, как отключить `$halt_on_error` на время выполнения одной операции:

```
$conn->halt_on_error = 0;
print ("Test of error-trapping:\n");
if (!$conn->issue_query ($query_str))
    print ("Hey, error $conn->errno occurred: $conn->errstr\n");
$conn->halt_on_error = 1;
```

Когда метод `error()` выводит сообщение, он также выводит и значения переменных (если переменная `$errno` не равна `0`). Метод `error()` преобразует сообщение в соответствующим образом экранированный HTML-код в предположении, что класс будет использоваться в веб-окружении:

```
function error ($msg)
{
    if (!$this->halt_on_error)  # вернуться молча
        return;
    $msg .= "\n";
    if ($this->errno)   # если известен код ошибки, вывести информацию об ошибке
        $msg .= sprintf ("Error: %s (%d)\n", $this->errstr, $this->errno);
    die (nl2br (htmlspecialchars ($msg)));
}
```

Запуск запросов и обработка результатов

Мы подошли к самому главному – к запуску запросов. Чтобы выполнить запрос, передайте его методу issue_query():

```
function issue_query ($query_str)
{
    if (!$this->connect ())     # если необходимо, установить соединение с сервером
        return (FALSE);

    $this->num_rows = 0;
    $this->result_id = mysql_query ($query_str, $this->conn_id);
    $this->errno = mysql_errno ();
    $this->errstr = mysql_error ();
    if ($this->errno)
    {
        $this->error ("Cannot execute query: $query_str");
        return (FALSE);
    }
    # получить количество обработанных строк для не-SELECT;
    # возвращается и количество строк для SELECT
    $this->num_rows = mysql_affected_rows ($this->conn_id);
    return ($this->result_id);
}
```

Прежде чем отправить запрос на сервер, метод issue_query() вызывает connect(), чтобы убедиться в том, что соединение установлено. Затем он выполняет запрос, устанавливает переменные ошибок (если ошибки не возникнут, значениями этих переменных будут 0 и пустая строка), проверяет, успешно ли выполнился запрос. В случае неуспешного выполнения issue_query() предпринимает соответствующее действие по обработке ошибок. В противном случае этот метод устанавливает переменную $num_rows, и идентификатор результирующего множества становится возвращаемым значением. Если этот запрос – не SELECT, то $num_rows показывает количество строк, измененных запросом. Для SELECT значение переменной $num_rows показывает количество возвращенных строк. (Здесь есть маленькая хитрость. В действительности mysql_affected_rows() предназначена только для не-SELECT предложений, но по счастливой случайности возвращает количество строк результирующего множества запросов SELECT.)

Если запрос формирует результирующее множество, вы захотите извлечь его строки. PHP предоставляет для выполнения этой операции ряд функций, которые рассматривались в рецепте 2.4: mysql_fetch_row(), mysql_fetch_array() и mysql_fetch_object(). Эти функции можно использовать как основу для соответствующих методов MySQL_Access: fetch_row(),fetch_array() и fetch_object(). Каждый из методов выбирает следующую строку и возвращает ее или, если строк больше нет, освобождает результирующее множество и возвращает FALSE. Они также автоматически при каждом вызове присваивают значение переменным ошибок. Здесь приведен метод fetch_row(); методы fetch_array() и fetch_object() устроены аналогично:

```
# Извлекает следующую строку в виде массива с числовыми индексами
function fetch_row ()
{
    $this->row = mysql_fetch_row ($this->result_id);
    $this->errno = mysql_errno ();
    $this->errstr = mysql_error ();
    if ($this->errno)
    {
        $this->error ("fetch_row error");
        return (FALSE);
    }
    if (is_array ($this->row))
        return ($this->row);
    $this->free_result ();
    return (FALSE);
}
```

Метод `free_result()`, используемый методами выборки строк, освобождает результирующее множество (если оно есть):

```
function free_result ()
{
    if ($this->result_id)
        mysql_free_result ($this->result_id);
    $this->result_id = 0;
    return (TRUE);
}
```

Автоматическое освобождение результирующего множества после извлечения его последней строки – это одно из преимуществ объектного интерфейса по отношению к интерфейсу PHP, основанному на функциях. Однако любой сценарий, извлекающий только часть строк результирующего множества, должен самостоятельно вызывать `free_result()` для явного освобождения результата.

Для того чтобы определить, является ли значение, входящее в результирующее множество, значением `NULL`, сравните его с `NULL`-значением PHP при помощи оператора тройного равенства:

```
if ($val === NULL)
{
    # $val - это значение NULL
}
```

Есть и другая возможность – использовать функцию `isset()`:

```
if (!isset ($val))
{
    # $val - это значение NULL
}
```

Теперь в объектном интерфейсе достаточно механизмов для написания сценариев, выдающих запросы и обрабатывающих их результаты:

```
# создать экземпляр объекта соединения
include "Cookbook_DB_Access.php";
$conn = new Cookbook_DB_Access;

# создать запрос, не возвращающий результирующее множество
$query = "UPDATE profile SET cats=cats+1 WHERE name = 'Fred'";
$conn->issue_query ($query);
print ($conn->num_rows . " rows were updated\n");

# запуск запросов, возвращающих строки, и извлечение строк всеми методами
$query = "SELECT id, name, cats FROM profile";
$conn->issue_query ($query);
while ($row = $conn->fetch_row ())
    print ("id: $row[0], name: $row[1], cats: $row[2]\n");
$conn->issue_query ($query);
while ($row = $conn->fetch_array ())
{
    print ("id: $row[0], name: $row[1], cats: $row[2]\n");
    print ("id: $row[id], name: $row[name], cats: $row[cats]\n");
}
$conn->issue_query ($query);
while ($row = $conn->fetch_object ())
    print ("id: $row->id, name: $row->name, cats: $row->cats\n");
```

Поддержка заполнителей и обработка специальных символов

В рецепте 2.7 была написана функция PHP `sql_quote()` для обработки в PHP кавычек, экранирования и работы со значениями `NULL` (неустановленными). Благодаря этой функции в запросе может использоваться любое значение:

```
function sql_quote ($str)
{
    if (!isset ($str))
        return ("NULL");
    $func = function_exists ("mysql_escape_string")
            ? "mysql_escape_string"
            : "addslashes";
    return ("'" . $func ($str) . "'");
}
```

Если добавить функцию `sql_quote()` в класс `MySQL_Access`, она автоматически будет доступна любому экземпляру класса как метод объекта, и вы сможете сформировать строку запроса, содержащую значения, заключенные в кавычки надлежащим образом:

```
$stmt = sprintf ("INSERT INTO profile (name,birth,color,foods,cats)
                    VALUES(%s,%s,%s,%s,%s)",
                        $conn->sql_quote ("De'Mont"),
                        $conn->sql_quote ("1973-01-12"),
                        $conn->sql_quote (NULL),
                        $conn->sql_quote ("eggroll"),
```

```
                        $conn->sql_quote (4));
$conn->issue_query ($stmt);
```

Можно использовать метод `sql_quote()` как основу для механизма эмуляции заполнителей, который будет применяться так:

1. Сначала строка запроса передается методу `prepare_query()`.
2. В строке запроса символами `?` указываются заполнители.
3. Запрос выполняется, при этом поставляется массив значений, который должен быть связан с запросом. Массив содержит по одному значению для каждого заполнителя. (Для связывания с заполнителем значения `NULL` передайте PHP-значение `NULL`.)

Для осуществления связывания (binding) параметров можно выполнять для строки запроса множество сравнений с шаблоном (поиск по образцу) и подстановок каждый раз, когда встречается символ-заполнитель `?`. Гораздо более простой способ состоит в разбиении строки запроса на части в местах использования заполнителей и их обратном склеивании при выполнении запроса с подстановкой между частями правильно заключенных в кавычки значений данных. Разбиение строки запроса также является удобным и простым способом подсчета числа заполнителей (оно на единицу меньше количества частей). Зная количество заполнителей, легко определить, нужное ли количество значений данных передается для связывания.

Метод `prepare_query()` очень прост. Единственное, что он делает, – разбивает строку запроса по символам `?`, помещая результат в массив `$query_pieces`, который затем будет использован при связывании параметров:

```
function prepare_query ($query)
{
    $this->query_pieces = explode ("?", $query);
    return (TRUE);
}
```

Можно было бы придумать новые вызовы для связывания значений данных с запросом и его выполнения, мы же просто несколько изменим `issue_query()`, чтобы выполняемое им действие зависело от типа аргумента. Если аргумент – строка символов, она интерпретируется как запрос для непосредственного выполнения (именно так метод `issue_query()` вел себя раньше). Если аргумент – массив, то считается, что он содержит значения данных для связывания с заранее подготовленным предложением. Измененный метод `issue_query()` выглядит так:

```
function issue_query ($arg = "")
{
    if ($arg == "")             # если аргумента нет, то считается, что нет значений
        $arg = array ();         # для связывания с подготовленным предложением
    if (!$this->connect ())     # при необходимости установить соединение с сервером
        return (FALSE);

    if (is_string ($arg))        # $arg - это простой запрос
        $query_str = $arg;
```

```
    else if (is_array ($arg))   # $arg содержит значения данных для заполнителей
    {
        if (count ($arg) != count ($this->query_pieces) - 1)
        {
            $this->errno = -1;
            $this->errstr = "data value/placeholder count mismatch";
            $this->error ("Cannot execute query");
            return (FALSE);
        }
        # вставить значения данных в запрос вместо
        # заполнителей, соответствующим образом используя кавычки
        $query_str = $this->query_pieces[0];
        for ($i = 0; $i < count ($arg); $i++)
        {
            $query_str .= $this->sql_quote ($arg[$i])
                        . $this->query_pieces[$i+1];
        }
    }
    else                        # в $arg содержится "мусор"
    {
        $this->errno = -1;
        $this->errstr = "unknown argument type to issue_query";
        $this->error ("Cannot execute query");
        return (FALSE);
    }

    $this->num_rows = 0;
    $this->result_id = mysql_query ($query_str, $this->conn_id);
    $this->errno = mysql_errno ();
    $this->errstr = mysql_error ();
    if ($this->errno)
    {
        $this->error ("Cannot execute query: $query_str");
        return (FALSE);
    }
    # получить количество обработанных строк для не-SELECT;
    # возвращается и количество строк для SELECT
    $this->num_rows = mysql_affected_rows ($this->conn_id);
    return ($this->result_id);
}
```

Заключение в кавычки и использование заполнителей поддерживаются, и теперь класс обеспечивает три способа формирования запросов. Во-первых, можно вводить всю строку запроса буквально и самостоятельно выполнять заключение в кавычки, экранирование специальных символов и обработку значений `NULL`**:**

```
$conn->issue_query ("INSERT INTO profile (name,birth,color,foods,cats)
                     VALUES('De\'Mont','1973-01-12',NULL,'eggroll','4')");
```

Во-вторых, можно использовать метод `sql_quote()` **для вставки значений данных в строку запроса:**

```
$stmt = sprintf ("INSERT INTO profile (name,birth,color,foods,cats)
                  VALUES(%s,%s,%s,%s,%s)",
                      $conn->sql_quote ("De'Mont"),
                      $conn->sql_quote ("1973-01-12"),
                      $conn->sql_quote (NULL),
                      $conn->sql_quote ("eggroll"),
                      $conn->sql_quote (4));
$conn->issue_query ($stmt);
```

В-третьих, можно использовать заполнители и предоставить всю работу по связыванию значений объектному интерфейсу:

```
$conn->prepare_query ("INSERT INTO profile (name,birth,color,foods,cats)
                       VALUES(?,?,?,?,?)");
$conn->issue_query (array ("De'Mont", "1973-01-12", NULL, "eggroll", 4));
```

Классы `MySQL_Access` и `Cookbook_DB_Access` предлагают удобный способ создания сценариев PHP, более простой в использовании, чем собственные вызовы PHP для MySQL. Кроме того, объектный интерфейс обеспечивает поддержку механизма заполнителей, о которой в PHP речь вообще не идет.

Разработка этих классов является примером создания интерфейса, скрывающего специфику работы с MySQL. Конечно, у интерфейса есть недостатки. Например, он позволяет единовременно подготовить только одно предложение, в то время как DBI и JDBC поддерживают множество параллельно подготавливаемых предложений. Если вам необходима такая функциональность, подумайте, как изменить `MySQL_Access` так, чтобы она поддерживалась.

2.10. Способы получения параметров соединения

Задача

Необходимо передать сценарию параметры соединения, чтобы он мог подключиться к серверу MySQL.

Решение

Есть масса способов выполнения этой операции. Выбор за вами.

Обсуждение

Любая программа, подключающаяся к MySQL, должна указывать такие параметры соединения, как имя пользователя, пароль и имя хоста. В предыдущих рецептах параметры соединения помещались прямо в код, устанавливающий соединение, но существует и ряд других способов передачи параметров. В данном разделе будет кратко рассмотрено некоторое количество таких способов, затем два из них будут реализованы. Итак, можно:

Жестко «зашить» параметры в программу. Параметры могут задаваться или в основном исходном файле или в используемом программой библиотечном файле. Это удобно, так как пользователям не приходится самим

вводить значения. Обратной стороной является недостаточная гибкость – чтобы изменить параметры, приходится менять программу.

Запрашивать параметры интерактивно. В средах типа командной строки вы можете задать пользователю ряд вопросов. В веб- или GUI-среде вы можете реализовать ввод параметров как диалог или заполнение формы. Это не очень удобно для тех, кто использует программу постоянно, ведь им приходится каждый раз вводить параметры.

Получать параметры из командной строки. Этот способ может применяться для команд, запускаемых интерактивно или из сценария. Как и при интерактивном получении параметров, вы заставляете пользователей вводить параметры при каждом использовании MySQL, что может быть скучно и утомительно. (Значительно уменьшающим неудобство фактором является то, что многие оболочки позволяют вызывать команды из исторического списка для повторного выполнения.)

Получать параметры из среды исполнения. Обычно этот метод используется для установки соответствующих переменных окружения в одном из загрузочных файлов оболочки (таких как *.cshrc* для *csh*, *.tcshrc* для *tcsh* или *.profile* для *sh*, *bash* и *ksh*). Тогда программы, запускаемые в рамках вашего сеанса, могут получать параметры из переменных окружения.

Получать параметры из отдельного файла. Вы можете хранить имя пользователя и пароль в файле, который программа читает перед установлением соединения с сервером MySQL. Чтение параметров из файла, изолированного от программы, избавляет вас от необходимости ввода параметров при каждом запуске программы, при этом не требуется и жесткое кодирование значений в самой программе. Это особенно удобно для интерактивных программ, так как вам не приходится вводить параметры каждый раз, когда вы используете программу. Кроме того, хранение параметров в файле позволяет сосредоточить параметры, используемые несколькими программами, в одном месте, при этом можно позаботиться о безопасности параметров, устанавливая режимы доступа к файлу. Например, вы можете запретить другим пользователям читать файл, установив для него режим, разрешающий обращение к файлу только лично вам.

Клиентская библиотека MySQL сама обеспечивает механизм файлов опций, хотя не все API поддерживают эту функциональность. Для тех, кто этого не делает, возможны обходные пути (например, Java поддерживает файлы свойств и предоставляет утилиты для их чтения).

Комбинировать методы. Часто бывает удобно комбинировать несколько методов, чтобы предоставить пользователю возможность задать параметры различными способами. Например, клиентские программы MySQL, такие как *mysql* и *mysqladmin* сначала ищут в нескольких каталогах файлы опций и читают их, если находят. Затем проверяются аргументы командной строки. Пользователи получают возможность указать параметры соединения в файле опций или в командной строке.

Методы получения параметров соединения так или иначе связаны с проблемами обеспечения безопасности:

- При любом методе, основанном на хранении параметров соединения в файле, возможны попытки несанкционированного доступа, если только файл не защищен от чтения неавторизованных пользователей. Это верно при хранении параметров в исходном файле, файле опций или сценарии, вызывающем команду и указывающем параметры в командной строке. Веб-сценарии, доступные для чтения только веб-серверу, небезопасны, если у других пользователей есть права администратора сервера.
- Параметры, указанные в командной строке или в переменных окружения, ненадежны. При выполнении программы ее аргументы, указанные в командной строке, и переменные окружения могут быть доступны другим пользователям, если те получают сведения о состоянии процессов (например, посредством команды *ps -e*). В частности, хранить пароль в переменной окружения стоит лишь в тех случаях, когда вы являетесь единственным пользователям компьютера или действительно уверены во всех остальных пользователях.

В оставшейся части главы рассказано, как обрабатывать аргументы командной строки для получения параметров соединения и как читать параметры из файла опций.

Получение параметров из командной строки

Стандартные правила MySQL касательно аргументов командной строки (то есть правила, которых придерживаются стандартные клиентские программы MySQL, такие как *mysql*) таковы: параметры могут указываться и в длинной, и в короткой форме опций. Например, имя пользователя `cbuser` может быть указано как *-u* `cbuser` (или *-u*`cbuser`) или как *--user*=`cbuser`. Кроме того, для опций, задающих пароль (*-p* или *--password*), можно опускать значение пароля после названия опции, чтобы указать, что программа должна запрашивать пароль интерактивно.

В приведенном далее наборе примеров показано, как обрабатывать аргументы командной строки для получения имени хоста, пользователя и пароля. Стандартными флагами для выполнения таких операций являются *-h* или *--host*, *-u* или *--user* и *-p* или *--password*. Вы можете написать собственный код для перемещения по списку аргументов, но обычно удобнее использовать существующие модули обработки опций, написанные именно для этого. Представленные программы реализованы при помощи функции типа `getopt()` для каждого API, кроме PHP. Если это возможно, примеры подражают поведению стандартных клиентов MySQL. (Для PHP примеров не будет, так как мало какие сценарии PHP пишутся для использования из командной строки.)

Perl

Perl передает аргументы командной строки сценариям в массиве `@ARGV`, который обрабатывается функцией `GetOptions()` модуля *Getopt::Long*. Следующая программа показывает, как разбирать (parse) аргументы командной строки на параметры соединения. Если опция пароля задана без аргумента, сценарий запрашивает значение пароля у пользователя.

```
#! /usr/bin/perl -w
# cmdline.pl - пример разбора опций командной строки в Perl

use strict;
use DBI;

use Getopt::Long;
$Getopt::Long::ignorecase = 0; # опции чувствительны к регистру
$Getopt::Long::bundling = 1;   # разрешить объединение коротких опций

# параметры соединения - все отсутствующие (undef) по умолчанию
my ($host_name, $password, $user_name);

GetOptions (
    # =s означает, что после опции требуется строковый аргумент
    # :s означает, что строковый аргумент после опции необязателен
    "host|h=s"      => \$host_name,
    "password|p:s"  => \$password,
    "user|u=s"      => \$user_name
) or exit (1);  # сообщение об ошибке не требуется; GetOptions() выводит собственное

# запрашивать пароль, если опция указана без значения
if (defined ($password) && $password eq "")
{
    # отключить отображение, но не мешать STDIN
    open (TTY, "/dev/tty") or die "Cannot open terminal\n";
    system ("stty -echo < /dev/tty");
    print STDERR "Enter password: ";
    chomp ($password = <TTY>);
    system ("stty echo < /dev/tty");
    close (TTY);
    print STDERR "\n";
}

# сформировать имя источника данных
my $dsn = "DBI:mysql:database=cookbook";
$dsn .= ";host=$host_name" if defined ($host_name);

# соединиться с сервером
my $dbh = DBI->connect ($dsn, $user_name, $password,
                        {PrintError => 0, RaiseError => 1});
print "Connected\n";

$dbh->disconnect ();
print "Disconnected\n";
exit (0);
```

Аргументы `GetOptions()` — это пары спецификаций опций и ссылок на переменные сценария, в которые должны быть помещены значения опций. Спецификация опции приводит длинную и короткую формы опции (без начальных дефисов), за которыми следует `=s`, если опция требует ввода аргумента, или `:s`, если за опцией может следовать аргумент. Например, `"host|h=s"` разрешает формы записи *--host* и *-h*, а также указывает, что за опцией должен следовать строковый аргумент. Передавать массив `@ARGV` не следует, так как

GetOptions() использует его неявно. После выполнения GetOptions() массив @ARGV содержит все оставшиеся аргументы, следующие за последней опцией.

Переменная $bundling модуля *Getopt::Long* изменяет интерпретацию аргументов, начинающихся с одного дефиса, таких как *-u*. Обычно хотелось бы воспринимать *-u* cbuser и *-u*cbuser как одно и то же, ведь именно так поступают стандартные клиентские программы MySQL. Однако если переменная $bundling равна нулю (значение по умолчанию), то GetOptions() трактует *-u*cbuser как единую опцию с именем «ucbuser». Если установить $bundling в ненулевое значение, то GetOptions() одинаково воспринимает *-u* cbuser и *-u*cbuser. Так происходит, потому что GetOptions() понимает опцию, начинающуюся с одного дефиса, как символ, зная, что несколько односимвольных опций могут быть объединены вместе. Например, когда функция видит *-u*cbuser, она смотрит на *u* и проверяет, принимает ли опция следующий аргумент. Если нет, то следующий символ интерпретируется как следующая буква опции. В противном случае оставшаяся часть строки понимается как значение опции. Что касается *-u*cbuser, *u* принимает аргумент, поэтому GetOptions() интерпретирует cbuser как значение опции.

Проблемой GetOptions() является то, что она не поддерживает *-p* без пароля так, как это делают стандартные клиентские программы MySQL. Если за *-p* следует другая опция, GetOptions() корректно определяет, что значение пароля не указано. Но если за *-p* следует аргумент без опции, GetOptions() ошибочно воспринимает его как пароль. В результате оказываются неэквивалентными два таких вызова *cmdline.pl*:

```
% cmdline.pl -h localhost -p -u cbuser xyz
Enter password:
% cmdline.pl -h localhost -u cbuser -p xyz
DBI->connect(database=cookbook;host=localhost) failed: Access denied for
user: 'cbuser@localhost' (Using password: YES) at ./cmdline.pl line 40
```

В первой команде GetOptions() определяет, что пароль не передан, и сценарий запрашивает его. Во второй команде GetOptions() принимает xyz за значение пароля.

Второй недостаток *cmdline.pl* заключается в том, что код запрашивания пароля – это специфический код UNIX, который не работает в Windows. Можно было бы попробовать использовать стандартный модуль Perl *Term::ReadKey*, но он тоже не работает в Windows. (Если у вас есть хороший запрашиватель пароля (password prompter) для Windows, пришлите мне его, пожалуйста, чтобы я включил его в дистрибутив recipes.)

PHP

PHP не обеспечивает серьезную поддержку обработки опций из командной строки, так как он используется преимущественно в веб-среде, где не очень-то распространены аргументы командной строки. Поэтому я не буду приводить пример для PHP в стиле getopt(). Если вы хотите написать собственную программу обработки аргументов, используйте содержащий аргументы масив $argv и переменную $argc, указывающую количество аргументов.

$argv[0] — это имя программы, а элементы с $argv[1] по $argv[$argc-1] — следующие аргументы. Вот как нужно обращаться к этим переменным:

```
print ("Number of arguments: $argc\n");
print ("Program name: $argv[0]\n");
print ("Arguments following program name:\n");
if ($argc == 1)
    print ("None\n");
else
{
    for ($i = 1; $i < $argc; $i++)
        print ("$i: $argv[$i]\n");
}
```

Python

Python передает аргументы команды сценарию в виде списка в переменной sys.argv. Вы можете получить доступ к этой переменной, импортировав модуль sys, а затем обработав его содержимое с помощью getopt() (если модуль getopt также импортирован). Ниже приведена программа, показывающая, как получить параметры из аргументов команды и использовать их для установления соединения с сервером:

```
#! /usr/bin/python
# cmdline.py - пример разбора опций командной строки в Python

import sys
import getopt
import MySQLdb

try:
    opts, args = getopt.getopt (sys.argv[1:],
                                "h:p:u:",
                                [ "host=", "password=", "user=" ])
except getopt.error, e:
    # print program name and text of error message
    print "%s: %s" % (sys.argv[0], e)
    sys.exit (1)

# значения параметров соединения по умолчанию
host_name = password = user_name = ""

# проход по опциям с извлечением имеющихся значений
for opt, arg in opts:
    if opt in ("-h", "--host"):
        host_name = arg
    elif opt in ("-p", "--password"):
        password = arg
    elif opt in ("-u", "--user"):
        user_name = arg

try:
    conn = MySQLdb.connect (db = "cookbook",
                            host = host_name,
```

```
                                user = user_name,
                                passwd = password)
        print "Connected"
    except MySQLdb.Error, e:
        print "Cannot connect to server"
        print "Error:", e.args[1]
        print "Code:", e.args[0]
        sys.exit (1)

    conn.close ()
    print "Disconnected"
    sys.exit (0)
```

Функция `getopt()` принимает два или три аргумента:

- Список аргументов команды (не включая имя команды `sys.argv[0]`). Можно использовать `sys.argv[1:]` для ссылок на список аргументов, следующих за именем программы.
- Строка символов, в которой приводятся короткие записи опций (буквы). За любой из букв может следовать символ двоеточия, указывающий на то, что опция требует аргумент, указывающий значение опции.
- Необязательный список длинных имен опций. За каждым именем может следовать `=`, что означает, что опция требует аргумент.

Функция `getopt()` возвращает два значения. Первое – это список пар опция/значение, а второе – список оставшихся аргументов, следующих за последней опцией. *cmdline.py* проходит по списку опций, чтобы определить, какие опции заданы и каковы их значения. Обратите внимание на то, что хотя вы не указываете начальные дефисы в названиях опций, передаваемых `getopt()`, возвращаемые функцией имена содержат начальные дефисы.

cmdline.py не запрашивает отсутствующий пароль, так как модуль `getopt` не предоставляет никакого способа для указания того, что какой-то аргумент опции является необязательным. К сожалению, это значит, что аргументы *-p* и *--password* нельзя указывать без значения пароля.

Java

Java передает аргументы командной строки программам в массиве, имя которого указывается в объявлении `main()`. В следующем объявлении используется массив `args`:

```
public static void main (String[] args)
```

Класс `Getopt` разборки аргументов в Java доступен в Интернете по адресу *http://www.urbanophile.com/arenn/coding/download.html*. Установите где-нибудь этот класс и убедитесь в том, что имя каталога его установки включено в переменную окружения `CLASSPATH`. Теперь можно использовать `Getopt` так, как показано в следующем примере:

```
// Cmdline.java - пример разбора опций командной строки в Java

import java.io.*;
```

```
import java.sql.*;
import gnu.getopt.*;    // это нужно для класса Getopt

public class Cmdline
{
    public static void main (String[] args)
    {
        Connection conn = null;
        String url = null;
        String hostName = null;
        String password = null;
        String userName = null;
        boolean promptForPassword = false;
        LongOpt[] longOpt = new LongOpt[3];
        int c;

        longOpt[0] =
            new LongOpt ("host", LongOpt.REQUIRED_ARGUMENT, null, 'h');
        longOpt[1] =
            new LongOpt ("password", LongOpt.OPTIONAL_ARGUMENT, null, 'p');
        longOpt[2] =
            new LongOpt ("user", LongOpt.REQUIRED_ARGUMENT, null, 'u');

        // назначить значение объекту обработки опций, затем
        // выполнять цикл до тех пор, пока опций не останется
        Getopt g = new Getopt ("Cmdline", args, "h:p::u:", longOpt);
        while ((c = g.getopt ()) != -1)
        {
            switch (c)
            {
            case 'h':
                hostName = g.getOptarg ();
                break;
            case 'p':
                // если задана опция пароля без последующего
                // значения, нужен запрос на ввод пароля
                password = g.getOptarg ();
                if (password == null)
                    promptForPassword = true;
                break;
            case 'u':
                userName = g.getOptarg ();
                break;
            case ':':               // необходимый аргумент отсутствует
            case '?':               // возникла какая-то другая ошибка
                // сообщение об ошибке не требуется; getopt() выводит собственное
                System.exit (1);
            }
        }

        if (password == null && promptForPassword)
        {
            try
```

```
            {
                DataInputStream s = new DataInputStream (System.in);
                System.err.print ("Enter password: ");
                // здесь следует отключить отображение символов...
                password = s.readLine ();
            }
            catch (Exception e)
            {
                System.err.println ("Error reading password");
                System.exit (1);
            }
        }

        try
        {
            // сформировать URL, указав, было ли задано имя хоста;
            // если нет, то MySQL работает с локальным хостом
            if (hostName == null)
                hostName = "";
            url = "jdbc:mysql://" + hostName + "/cookbook";
            Class.forName ("com.mysql.jdbc.Driver").newInstance ();
            conn = DriverManager.getConnection (url, userName, password);
            System.out.println ("Connected");
        }
        catch (Exception e)
        {
            System.err.println ("Cannot connect to server");
        }
        finally
        {
            if (conn != null)
            {
                try
                {
                    conn.close ();
                    System.out.println ("Disconnected");
                }
                catch (Exception e) { }
            }
        }
    }
}
```

Из примера видно, что сначала вы проводите подготовку к разбору аргументов, создавая новый экземпляр объекта `Getopt`, которому будут передаваться аргументы программы и информация, описывающая принимаемые программой опции. Затем вызываете функцию `getopt()` в цикле до тех пор, пока она не вернет `-1`, что будет означать «опций больше нет». При каждом выполнении цикла функция `getopt()` возвращает значение, показывающее, какую опцию она видит. При необходимости можно вызвать `getOptarg()` для получения аргумента опции. (`getOptarg()` возвращает `null`, если аргумент не задан.)

При создании экземпляра класса Getopt ему передается три или четыре аргумента:

- Имя программы; используется для сообщений об ошибках.
- Массив аргументов, имя которого указано в объявлении main().
- Строка, перечисляющая буквы коротких опций (без начальных дефисов). За любой буквой может следовать двоеточие, означающее необходимость ввода аргумента, или двойное двоеточие (::), указывающее на необязательность наличия аргументов после опции.
- Необязательный массив, содержащий сведения о длинных опциях. Для того чтобы использовать длинные опции, вы должны создать массив объектов LongOpt. Каждый из них описывает одну опцию с помощью четырех параметров:
  - Имя опции в виде строки символов (без начальных дефисов).
  - Значение, указывающее, принимает ли опция аргумент. Возможные значения: LongOpt.NO_ARGUMENT, LongOpt.REQUIRED_ARGUMENT или LongOpt.OPTIONAL_ARGUMENT.
  - Объект StringBuffer или null. Функция getopt() определяет, как использовать это значение, в зависимости от четвертого параметра объекта LongOpt.
  - Значение, которое должно использоваться, когда встретится опция. Это значение становится значением, возвращаемым функцией getopt(), если объект StringBuffer, названный в третьем параметре, – это null. В противном случае (буфер – не null) getopt() помещает строковое представление четвертого параметра в буфер и возвращает ноль.

В примере значение null используется как параметр StringBuffer для всех объектов длинных опций, а буква соответствующей короткой опции – как четвертый параметр. В результате getopt() возвращает букву короткой опции и для длинной, и для короткой опции, так что их можно обрабатывать в одном предложении case.

После того как getopt() возвратила –1, показывая, что в массиве аргументов больше нет опций, getOptind() возвращает индекс первого аргумента, следующего за последней опцией. Доступ к оставшимся аргументам можно осуществить, например так:

```
for (int i = g.getOptind (); i < args.length; i++)
    System.out.println (args[i]);
```

В дополнение к описанному класс Getopt поддерживает и другой способ обработки опций. Подробная информация приведена в документации на класс.

Недостатком программы *Cmdline.java* можно назвать невозможность отключения отображения пароля при его считывании.

Получение параметров из файла опций

Если ваш API поддерживает такую возможность, вы можете задавать параметры соединения в файле опций MySQL, и API будет самостоятельно считывать их оттуда. Если API не поддерживает собственно файлы опций, можно попытаться сделать так, чтобы читались другие типы файлов, в которых хранятся параметры, или написать собственные функции, которые будут обращаться к файлам опций.

Формат файлов опций был описан в главе 1. Я буду считать, что вы читали то описание, и перейду непосредственно к использованию файлов опций программами. При работе в UNIX пользовательские опции принято указывать в файле *~/.my.cnf* (то есть в файле *.my.cnf* вашего домашнего каталога). Однако механизм файлов опций MySQL может просматривать несколько разных файлов. Стандартный порядок обхода таков: */etc/my.cnf*, файл *my.cnf* в каталоге данных сервера по умолчанию и файл *~/.my.cnf* текущего пользователя. В Windows порядок просмотра файлов следующий: файл *my.ini* в системном каталоге Windows *C:\my.cnf* и файл *my.cnf* в каталоге данных сервера по умолчанию. Если существует множество файлов опций, и один параметр задан в нескольких файлах, то приоритетным является значение, прочитанное последним. Отсутствие любого из файлов опций не является ошибкой.

Программа не будет использовать файлы опций MySQL, если не сообщить ей о том, что необходимо их использовать. Perl и Python предоставляют в API явную поддержку чтения файлов опций, так что просто укажите, что хотите использовать их при установке соединения с сервером. Можно задать необходимость чтения только какого-то определенного файла или использовать стандартный порядок просмотра для работы с несколькими файлами. PHP и Java не поддерживают файлы опций. Чтобы обойти это ограничение, напишем для PHP простую функцию разбора файла опций, а для Java выберем другой подход, использующий файлы свойств.

Хотя и существует соглашение о том, что в UNIX файл пользовательских опций – это файл *.my.cnf* домашнего каталога текущего пользователя, но нет правила, обязывающего программы использовать этот конкретный файл. Вы можете назвать файл опций как угодно и поместить его куда угодно. Например, можно создать файл */usr/local/apache/lib/cb.cnf* для веб-сценариев, обращающихся к базе данных `cookbook`. Возможны ситуации, когда вам захочется создать несколько файлов. Тогда из любого сценария вы сможете выбирать тот файл, который соответствует определенному типу прав доступа. Например, у вас может быть один файл опций *cb.cnf*, перечисляющий параметры для полного доступа к MySQL, и отдельный файл *cb-ro.cnf*, содержащий параметры соединения для пользователя, которому нужен только доступ на чтение к MySQL. Также можно создать несколько групп внутри одного файла опций и сделать так, чтобы сценарий выбирал опции из надлежащей группы.

Поддержка файлов опций в API для C

API для Perl и Python созданы на основе API для C, а поддержка файлов опций была добавлена в клиентскую библиотеку C только для версии MySQL 3.22.10. То есть даже тем, кто использует Perl и Python, необходима версия MySQL 3.22.10, чтобы они могли работать с файлами опций внутри своих программ.

Исторически сложилось так, что имя базы данных не относится к параметрам, получаемым из файла опций. (Обычно это значение содержится в самой программе или предполагается, что оно будет задано пользователем.) Начиная с MySQL 3.23.6 клиентская библиотека C поддерживает поиск в файлах опций строк вида `database=`*`db_name`*, но в примерах данного раздела этот факт не учитывается.

Perl

Сценарии DBI Perl могут использовать файлы опций, если у вас есть *DBD::mysql* версии 1.21.06 или выше. Чтобы воспользоваться такой возможностью, поместите спецификации опций в третий компонент строки имени источника данных:

- Чтобы задать группу опций, используйте `mysql_read_default_group=`*`имя_группы`*. Тем самым вы укажете MySQL, что опции следует искать в указанной группе и группе `[client]` стандартных файлов опций. Значение *`имя_группы`* должно быть записано без квадратных скобок, присутствующих в строке, начинающей группу. Например, если группа в файле опций начинается со строки `[my_prog]`, то в качестве значения *`имя_группы`* нужно ввести `my_prog`. Если вы хотите просматривать стандартные файлы опций, но заглядывать только в группу `[client]`, то укажите `client` как значение *`имя_группы`*.
- Чтобы указать определенный файл опций, используйте в DSN `mysql_read_default_file=`*`имя_файла`*. Если вы это сделаете, MySQL будет просматривать только этот файл, и только опции группы `[client]`.
- Если вы укажете и имя файла опций, и имя группы опций, MySQL будет читать только указанный файл и искать опции в названной группе и в группе `[client]`.

Укажем MySQL, что нужно использовать стандартный порядок просмотра файлов опций и искать опции в группах `[cookbook]` и `[client]`:

```
# стандартный DSN
my $dsn = "DBI:mysql:database=cookbook";
# просматривать стандартные файлы опций; использовать группы [cookbook] и [client]
$dsn .= ";mysql_read_default_group=cookbook";
my $dbh = DBI->connect ($dsn, undef, undef,
                        { PrintError => 0, RaiseError => 1 });
```

В следующем примере вы явно задаете имя файла опций, расположенного в $ENV{HOME}, домашнем каталоге пользователя, запускающего сценарий. Теперь MySQL будет просматривать только этот файл опций и использует опции из группы [client]:

```
# стандартный DSN
my $dsn = "DBI:mysql:database=cookbook";
# просматривать пользовательский файл, принадлежащий текущему пользователю
$dsn .= ";mysql_read_default_file=$ENV{HOME}/.my.cnf";
my $dbh = DBI->connect ($dsn, undef, undef,
                        { PrintError => 0, RaiseError => 1 });
```

Если вы передаете пустое значение (undef или пустую строку) для аргументов имени пользователя и пароля в вызове connect(), то connect() использует любые значения, обнаруженные в файле (файлах) опций. Аналогично имя хоста, указанное в DSN, замещает любое значение файла опций. Эту особенность можно использовать для того, чтобы дать сценариям DBI возможность получать параметры соединения и из файлов опций, и из командной строки:

1. Создайте переменные $host_name, $user_name и $password и установите для них начальное значение undef. Затем произведите разбор аргументов командной строки с тем, чтобы установить переменные в не-undef-значения, если соответствующие опции присутствуют в командной строке (посмотрите, как это делалось в сценариях Perl, приведенные ранее в этом разделе).
2. После того как аргументы командной строки разобраны, сформируйте строку DSN и вызовите connect(). Используйте mysql_read_default_group и mysql_read_default_file для указания того, какие файлы опций вы хотели бы использовать. Если $host_name не undef, добавьте в DSN host=$host_name. Вызывая connect(), передайте $user_name и $password как аргументы имени пользователя и пароля. По умолчанию переменные содержат undef, если же в них записаны значения аргументов командной строки, то они содержат не-undef-значения, которые аннулируют любые значения файлов опций.

Если сценарий придерживается изложенной выше последовательности действий, то параметры, указанные пользователем в командной строке, передаются connect() и имеют приоритет по отношению к содержимому файлов опций.

PHP

В PHP нет собственной поддержки использования файлов опций MySQL, по крайней мере, в настоящий момент. Чтобы обойти это ограничение, напишем функцию, которая будет читать файл опций – read_mysql_option_file(). Она принимает как аргументы имя файла опций и имя группы опций или массив, содержащий имена групп (имена групп должны приводиться без квадратных скобок). Функция считывает все опции, присутствующие в указанной группе (группах) файла. Если аргумент группы опций не задан, функция по умолчанию просматривает группу [client]. Возвращаемое значение представляет собой массив пар «имя опции/значение опции» или FALSE в случае ошибки. Отсутствие файла опций не рассматривается как ошибка.

```
function read_mysql_option_file ($filename, $group_list = "client")
{
    if (is_string ($group_list))            # преобразовать строку в массив
        $group_list = array ($group_list);
    if (!is_array ($group_list))            # хм ... странный аргумент?
        return (FALSE);
    $opt = array ();                        # массив пар "имя опции/значение опции"
    if (!($fp = fopen ($filename, "r")))    # если файл не существует,
        return ($opt);                      # возвращать пустой список
    $in_named_group = 0;                   # ненулевое значение, пока обрабатывается
                                            # указанная группа
    while ($s = fgets ($fp, 1024))
    {
        $s = trim ($s);
        if (ereg ("^[#;]", $s))                 # пропустить комментарии
            continue;
        if (ereg ("^\[([^]]+)]", $s, $arg))     # строка входит в группу опций?
        {
            # проверить, в нужной ли мы группе
            $in_named_group = 0;
            reset ($group_list);
            while (list ($key, $group_name) = each ($group_list))
            {
                if ($arg[1] == $group_name)
                {
                    $in_named_group = 1;        # да, мы в нужной группе
                    break;
                }
            }
            continue;
        }
        if (!$in_named_group)                   # нет, мы не в нужной группе,
            continue;                           # пропустить строку
        if (ereg ("^([^ \t=]+)[ \t]*=[ \t]*(.*)", $s, $arg))
            $opt[$arg[1]] = $arg[2];            # имя=значение
        else if (ereg ("^([^ \t]+)", $s, $arg))
            $opt[$arg[1]] = "";                 # только имя
        # иначе строка деформируется
    }
    return ($opt);
}
```

Рассмотрим пару примеров использования функции `read_mysql_option_file()`**. Первый пример читает пользовательский файл опций для получения группы параметров** `[client]`**, затем использует их для соединения с сервером. Второй пример читает файл опций системы и выводит найденные там параметры запуска сервера (то есть параметры групп** `[mysqld]` **и** `[server]`**):**

```
$opt = read_mysql_option_file ("/u/paul/.my.cnf");
$link = @mysql_connect ($opt["host"], $opt["user"], $opt["password"]);

$opt = read_mysql_option_file ("/etc/my.cnf", array ("mysqld", "server"));
```

```
while (list ($name, $value) = each ($opt))
    print ("$name => $value\n");
```

Если вы используете интерфейс `MySQL_Access`, созданный в рецепте 2.9, то можете подумать о том, как расширить класс за счет порожденного класса, который получает имя пользователя, пароль и имя хоста из файла опций. Вы также можете разрешить этому порожденному классу просматривать несколько файлов (`read_mysql_option_file()` не предоставляет эту возможность, обычно востребованную при работе с файлами опций).

Python

Модуль MySQLdb для DB-API предоставляет непосредственную поддержку работы с файлами опций MySQL. Укажите файл опций или группу опций, используя аргументы `read_default_file` или `read_default_group` метода `connect()`. Эти два аргумента действуют так же, как опции `mysql_read_default_file` и `mysql_read_default_group` метода `connect()` Perl DBI (см. предыдущий раздел, посвященный Perl). Чтобы использовать стандартный порядок просмотра файлов опций в группах `[cookbook]` и `[client]`, выполните нечто подобное:

```
try:
    conn = MySQLdb.connect (db = "cookbook", read_default_group = "cookbook")
    print "Connected"
except:
    print "Cannot connect to server"
    sys.exit (1)
```

Следующий пример показывает, как использовать файл *.my.cnf*, находящийся в домашнем каталоге текущего пользователя, для получения параметров из группы `[client]`:[1]

```
try:
    option_file = os.environ["HOME"] + "/" + ".my.cnf"
    conn = MySQLdb.connect (db = "cookbook", read_default_file = option_file)
    print "Connected"
except:
    print "Cannot connect to server"
    sys.exit (1)
```

Java

Драйвер MySQL Connector/J JDBC не поддерживает файлы опций. Однако библиотека классов Java поддерживает чтение файлов свойств, которые содержат строки в формате *имя=значение*. Это похоже на формат файлов опций MySQL, хотя есть и отличия (например, файлы опций не разрешают такие строки, как [*имя_группы*]). Посмотрим на простой файл свойств:

```
# этот файл содержит список параметров для соединения с сервером MySQL
user=cbuser
password=cbpass
host=localhost
```

[1] Для доступа к `os.environ` необходимо импортировать модуль `os`.

Напишем программу *ReadPropsFile.java*, которая иллюстрирует один из способов чтения файла свойств (в данном случае – *Cookbook.properties*) для получения параметров соединения. Файл должен находиться в каталоге, имя которого указано в переменной CLASSPATH, или вам придется указывать его полное путевое имя (мы будем считать, что файл находится в каталоге, включенном в CLASSPATH):

```
import java.sql.*;
import java.util.*;     // это нужно для поддержки файла свойств

public class ReadPropsFile
{
    public static void main (String[] args)
    {
        Connection conn = null;
        String url = null;
        String propsFile = "Cookbook.properties";
        Properties props = new Properties ();

        try
        {
            props.load (ReadPropsFile.class.getResourceAsStream (propsFile));
        }
        catch (Exception e)
        {
            System.err.println ("Cannot read properties file");
            System.exit (1);
        }
        try
        {
            // сформировать URL соединения, вставляя
            // имя пользователя и пароль как параметры в конце
            url = "jdbc:mysql://"
                        + props.getProperty ("host")
                        + "/cookbook"
                        + "?user=" + props.getProperty ("user")
                        + "&password=" + props.getProperty ("password");
            Class.forName ("com.mysql.jdbc.Driver").newInstance ();
            conn = DriverManager.getConnection (url);
            System.out.println ("Connected");
        }
        catch (Exception e)
        {
            System.err.println ("Cannot connect to server");
        }
        finally
        {
            try
            {
                if (conn != null)
                {
                    conn.close ();
```

```
                    System.out.println ("Disconnected");
                }
            }
            catch (SQLException e) { /* игнорировать ошибки закрытия */ }
        }
    }
}
```

Если вы хотите, чтобы функция `getProperty()` возвращала некоторое значение по умолчанию, когда указанное свойство не найдено, передайте это значение как второй аргумент. Например, чтобы использовать `localhost` как значение по умолчанию для `host`, вызовите `getProperty()` так:

```
String hostName = props.getProperty ("host", "localhost");
```

Библиотечный файл *Cookbook.class*, созданный в рецепте 2.3, содержит программу `propsConnect()`, реализующую только что изложенные принципы. Чтобы использовать ее, наполните содержимым файл свойств *Cookbook.properties* и скопируйте этот файл туда же, где находится *Cookbook.class*. Теперь для установления соединения из программы вы можете импортировать класс `Cookbook` и вызывать `Cookbook.propsConnect()` вместо того, чтобы вызывать `Cookbook.connect()`.

2.11. Заключение и рекомендации

В этой главе было рассказано об основных операциях, предоставляемых в каждом из наших четырех API для различных аспектов взаимодействия с сервером MySQL. Эти операции позволяют писать программы, формирующие любые запросы и извлекающие результаты. Пока что рассматривались только простые запросы, так как акцент делался на API, а не на SQL. В следующей главе более пристальное внимание уделено именно SQL, в ней мы поговорим о том, как задавать серверу базы данных более сложные вопросы.

Прежде чем переходить к новой главе, следует вернуть в исходное состояние таблицу `profile`, которая будет использоваться во многих примерах следующих глав. Чтобы увидеть те же результаты при выполнении запросов из примеров, что и я, вы должны восстановить ее начальное состояние. Для этого перейдите в каталог *tables* дистрибутива `recipes` и выполните команды:

```
% mysql cookbook < profile.sql
% mysql cookbook < profile2.sql
```

3

Выбор записей

3.0. Введение

Глава посвящена предложению SELECT, которое используется для извлечения информации из базы данных. Приводятся базовые сведения о различных способах применения SELECT с целью сообщить MySQL, что вы хотели бы видеть. Глава также будет полезна тем, кто не очень хорошо знаком с SQL или хочет разобраться в специальных расширениях синтаксиса SELECT, существующих в MySQL. Но способов создания предложений SELECT так много, что мы сможем затронуть только некоторые из них. За дополнительной информацией о синтаксисе SELECT можно обратиться к справочному руководству по MySQL, где вам также будет предложено описание функций и операторов, используемых для извлечения и обработки данных.

Используя SELECT, вы можете указывать:

- Какую таблицу следует использовать
- Какие столбцы таблицы отображать
- Как назвать столбцы
- Какие строки извлекать из таблицы
- Как упорядочить строки

Многие полезные запросы очень просты и не требуют указывать все вышеперечисленное. Например, некоторые формы SELECT даже не указывают имя таблицы (эта возможность была востребована в рецепте 1.31 для использования *mysql* в качестве калькулятора). Другие запросы, не связанные с таблицами, могут применяться, например для проверки версии сервера или имени текущей базы данных:

```
mysql> SELECT VERSION(), DATABASE();
+-------------+------------+
| VERSION()   | DATABASE() |
+-------------+------------+
| 3.23.51-log | cookbook   |
+-------------+------------+
```

Однако для ответа на более сложные вопросы обычно необходимо извлечь информацию из одной или нескольких таблиц. Во многих примерах главы используется таблица `mail`, столбцы которой хранят журнал обмена почтовыми сообщениями пользователей, работающих на нескольких хостах. Ее определение выглядит так:

```
CREATE TABLE mail
(
    t       DATETIME,   # время отправки сообщения
    srcuser CHAR(8),    # отправитель (имя и хост отправителя)
    srchost CHAR(20),
    dstuser CHAR(8),    # получатель (имя и хост получателя)
    dsthost CHAR(20),
    size    BIGINT,     # размер сообщения в байтах
    INDEX   (t)
);
```

А вот ее содержимое:

```
+---------------------+---------+---------+---------+---------+---------+
| t                   | srcuser | srchost | dstuser | dsthost | size    |
+---------------------+---------+---------+---------+---------+---------+
| 2001-05-11 10:15:08 | barb    | saturn  | tricia  | mars    |   58274 |
| 2001-05-12 12:48:13 | tricia  | mars    | gene    | venus   |  194925 |
| 2001-05-12 15:02:49 | phil    | mars    | phil    | saturn  |    1048 |
| 2001-05-13 13:59:18 | barb    | saturn  | tricia  | venus   |     271 |
| 2001-05-14 09:31:37 | gene    | venus   | barb    | mars    |    2291 |
| 2001-05-14 11:52:17 | phil    | mars    | tricia  | saturn  |    5781 |
| 2001-05-14 14:42:21 | barb    | venus   | barb    | venus   |   98151 |
| 2001-05-14 17:03:01 | tricia  | saturn  | phil    | venus   | 2394482 |
| 2001-05-15 07:17:48 | gene    | mars    | gene    | saturn  |    3824 |
| 2001-05-15 08:50:57 | phil    | venus   | phil    | venus   |     978 |
| 2001-05-15 10:25:52 | gene    | mars    | tricia  | saturn  |  998532 |
| 2001-05-15 17:35:31 | gene    | saturn  | gene    | mars    |    3856 |
| 2001-05-16 09:00:28 | gene    | venus   | barb    | mars    |     613 |
| 2001-05-16 23:04:19 | phil    | venus   | barb    | venus   |   10294 |
| 2001-05-17 12:49:23 | phil    | mars    | tricia  | saturn  |     873 |
| 2001-05-19 22:21:51 | gene    | saturn  | gene    | venus   |   23992 |
+---------------------+---------+---------+---------+---------+---------+
```

Чтобы создать таблицу `mail` и заполнить ее данными, перейдите в каталог *tables* дистрибутива `recipes` и выполните команду:

```
% mysql cookbook < mail.sql
```

Время от времени в примерах главы будут использоваться и другие таблицы. Некоторые из них уже использовались в других главах, другие появятся впервые. Если таблицу необходимо будет создать, поступите так же, как в случае с таблицей `mail`: используйте сценарии каталога *tables*. Кроме того, тексты многих сценариев и программ, использованных в данной главе, можно найти в каталоге *select*. Эти файлы упростят вам работу с примерами.

Многие из приведенных примеров можно выполнить в программе *mysql*, о которой рассказано в главе 1. Некоторые примеры выдают запросы в контексте

языка программирования – за базовыми сведениями о приемах программирования обращайтесь к главе 2.

3.1. Задание столбцов вывода

Задача

Вы хотите вывести на экран несколько (или все) столбцов таблицы.

Решение

Для отображения всех столбцов используйте групповой символ `*`. Или же явно укажите названия столбцов, которые следует вывести.

Обсуждение

Чтобы сообщить, какая информация вас интересует, укажите имя столбца или список столбцов, а также имя таблицы. Проще всего организовать вывод при помощи группового символа `*`, заменяющего имена всех столбцов таблицы:

```
mysql> SELECT * FROM mail;
+---------------------+---------+---------+---------+---------+---------+
| t                   | srcuser | srchost | dstuser | dsthost | size    |
+---------------------+---------+---------+---------+---------+---------+
| 2001-05-11 10:15:08 | barb    | saturn  | tricia  | mars    |   58274 |
| 2001-05-12 12:48:13 | tricia  | mars    | gene    | venus   |  194925 |
| 2001-05-12 15:02:49 | phil    | mars    | phil    | saturn  |    1048 |
| 2001-05-13 13:59:18 | barb    | saturn  | tricia  | venus   |     271 |
...
```

Можно и явно указать названия столбцов:

```
mysql> SELECT t, srcuser, srchost, dstuser, dsthost, size FROM mail;
+---------------------+---------+---------+---------+---------+---------+
| t                   | srcuser | srchost | dstuser | dsthost | size    |
+---------------------+---------+---------+---------+---------+---------+
| 2001-05-11 10:15:08 | barb    | saturn  | tricia  | mars    |   58274 |
| 2001-05-12 12:48:13 | tricia  | mars    | gene    | venus   |  194925 |
| 2001-05-12 15:02:49 | phil    | mars    | phil    | saturn  |    1048 |
| 2001-05-13 13:59:18 | barb    | saturn  | tricia  | venus   |     271 |
...
```

Конечно, гораздо проще использовать `*`, чем указывать имена всех столбцов, но при этом заранее неизвестно, в каком именно порядке они будут выведены. (Сервер возвращает столбцы в том порядке, в котором они приведены в определении таблицы и который может изменяться при изменении определения – см. главу 8.) Преимущество явного задания имен столбцов в том, что вы получаете возможность вывести их в любом порядке. Пусть, например, вы хотите, чтобы имена хостов выводились перед именами пользователей, а не наоборот. Изменим порядок указания столбцов:

```
mysql> SELECT t, srchost, srcuser, dsthost, dstuser, size FROM mail;
+---------------------+---------+---------+---------+---------+---------+
| t                   | srchost | srcuser | dsthost | dstuser | size    |
+---------------------+---------+---------+---------+---------+---------+
| 2001-05-11 10:15:08 | saturn  | barb    | mars    | tricia  |   58274 |
| 2001-05-12 12:48:13 | mars    | tricia  | venus   | gene    |  194925 |
| 2001-05-12 15:02:49 | mars    | phil    | saturn  | phil    |    1048 |
| 2001-05-13 13:59:18 | saturn  | barb    | venus   | tricia  |     271 |
...
```

Кроме того, если вы указываете названия столбцов явно, то можете вывести только интересующие вас столбцы, а не все, как в случае с использованием символа `*`:

```
mysql> SELECT size FROM mail;
+---------+
| size    |
+---------+
|   58274 |
|  194925 |
|    1048 |
|     271 |
...
mysql> SELECT t, srcuser, srchost, size FROM mail;
+---------------------+---------+---------+---------+
| t                   | srcuser | srchost | size    |
+---------------------+---------+---------+---------+
| 2001-05-11 10:15:08 | barb    | saturn  |   58274 |
| 2001-05-12 12:48:13 | tricia  | mars    |  194925 |
| 2001-05-12 15:02:49 | phil    | mars    |    1048 |
| 2001-05-13 13:59:18 | barb    | saturn  |     271 |
...
```

3.2. Решение проблем с неправильным порядком вывода столбцов

Задача

Вы даете в программе запрос `SELECT *`, а столбцы отображаются не в том порядке, в каком хотелось бы.

Решение

Если вы используете для выбора столбцов символ `*`, то ни в чем не можете быть уверены – нельзя сделать никакого предположения о том, в каком порядке будут возвращены столбцы. Явно указывайте имена столбцов в том порядке, который вас устраивает, либо извлекайте их в такую структуру данных, в которой их порядок не имеет значения.

Обсуждение

В примерах предыдущего раздела было показано отличие использования символа * от списка названий для указания выводимых предложением SELECT столбцов в программе *mysql*. При запуске запросов из ваших собственных программ через API разница при использовании этих двух подходов также может быть существенной. Если вы выбираете столбцы при помощи *, сервер возвращает их в том порядке, в котором они приведены в определении таблицы, то есть в порядке, который может изменяться при изменении структуры таблицы. Если строки выбираются в массив, то такая неопределенность с порядком вывода столбцов лишает вас возможности определить, какому столбцу соответствует каждый элемент массива. Если же указывать имена столбцов явно, то вы сможете быть уверены в том, что выбранные столбцы появятся в массиве в том порядке, в котором вы ввели их имена.

Но ваш API может поддерживать извлечение строк в структуру, к элементам которой можно обращаться по имени. (Например, хеш в Perl; ассоциативный массив или объект в PHP.) Тогда можно дать запрос SELECT *, а затем обращаться к элементам структуры по именам столбцов в любом удобном для вас порядке. В данном случае не будет никакой разницы между выборкой столбцов при помощи * или с указанием явного списка имен. Если в вашей программе возможен доступ к значениям по имени, то их порядок в результирующем множестве не важен. Может возникнуть соблазн пойти простым путем и использовать символ * во всех запросах SELECT, даже если вам нужны не все столбцы. Однако эффективнее указывать имена только тех столбцов, которые вам действительно нужны, чтобы сервер не отправлял вам заведомо лишнюю информацию. (Пример, подробно поясняющий, почему в некоторых случаях извлечение определенных столбцов нежелательно, рассматривается в рецепте 9.8, раздел «Выбор всех столбцов, кроме некоторых».)

3.3. Присваивание имен столбцам вывода

Задача

Вам не нравятся имена столбцов в вашем результирующем множестве.

Решение

Определите собственные имена, используя механизм псевдонимов (aliases).

Обсуждение

При извлечении результирующего множества MySQL дает имя каждому выводимому столбцу. (Именно так программа *mysql* получает имена, которые вы видите в первой строке заголовков столбцов при выводе результирующего множества.) MySQL присваивает столбцам имена по умолчанию, но если они вам не подходят, то вы можете заменить их на другие, используя псевдонимы столбцов.

В этом разделе будет рассказано о псевдонимах и показано, как пользоваться ими для именования столбцов в запросах. Если вы пишете программу, которой необходимо извлекать информацию об именах столбцов, обратитесь к рецепту 9.2.

Если столбец вывода попадает в результирующее множество непосредственно из таблицы, то MySQL называет столбец результирующего множества так, как он и назывался в таблице. Например, следующее предложение выбирает из таблицы три столбца, имена которых и становятся именами соответствующих столбцов результирующего множества:

```
mysql> SELECT t, srcuser, size FROM mail;
+---------------------+---------+---------+
| t                   | srcuser | size    |
+---------------------+---------+---------+
| 2001-05-11 10:15:08 | barb    |   58274 |
| 2001-05-12 12:48:13 | tricia  |  194925 |
| 2001-05-12 15:02:49 | phil    |    1048 |
| 2001-05-13 13:59:18 | barb    |     271 |
...
```

Если же столбец формируется в результате вычисления какого-то выражения, то именем столбца будет это выражение. Как видно из примера (где выражение используется для изменения формата столбца `t` таблицы `mail`), такие имена могут быть весьма громоздкими:

```
mysql> SELECT
    -> CONCAT(MONTHNAME(t),' ',DAYOFMONTH(t),', ',YEAR(t)),
    -> srcuser, size FROM mail;
+-------------------------------------------------+---------+---------+
| CONCAT(MONTHNAME(t),' ',DAYOFMONTH(t),', ',YEAR(t)) | srcuser | size    |
+-------------------------------------------------+---------+---------+
| May 11, 2001                                    | barb    |   58274 |
| May 12, 2001                                    | tricia  |  194925 |
| May 12, 2001                                    | phil    |    1048 |
| May 13, 2001                                    | barb    |     271 |
...
```

Предыдущий пример был придуман специально для того, чтобы показать, насколько ужасно могут выглядеть имена столбцов. Я сказал «придуман специально», так как на деле вы вряд ли напишете такой запрос, ведь тот же результат можно получить, используя функцию MySQL `DATE_FORMAT()`. Но и в этом случае имена столбцов не очень-то привлекательны:

```
mysql> SELECT
    -> DATE_FORMAT(t,'%M %e, %Y'),
    -> srcuser, size FROM mail;
+---------------------------+---------+---------+
| DATE_FORMAT(t,'%M %e, %Y') | srcuser | size    |
+---------------------------+---------+---------+
| May 11, 2001              | barb    |   58274 |
| May 12, 2001              | tricia  |  194925 |
```

```
| May 12, 2001              | phil    |    1048 |
| May 13, 2001              | barb    |     271 |
...
```

Чтобы дать столбцу результирующего множества собственное имя, укажите псевдоним столбца при помощи конструкции AS *имя*. Следующий запрос извлекает тот же результат, что и предыдущий, но переименовывает первый столбец в date_sent:

```
mysql> SELECT
    -> DATE_FORMAT(t,'%M %e, %Y') AS date_sent,
    -> srcuser, size FROM mail;
+--------------+---------+---------+
| date_sent    | srcuser | size    |
+--------------+---------+---------+
| May 11, 2001 | barb    |   58274 |
| May 12, 2001 | tricia  |  194925 |
| May 12, 2001 | phil    |    1048 |
| May 13, 2001 | barb    |     271 |
...
```

Как видите, благодаря псевдониму имя столбца стало короче, удобнее для чтения и понятнее. Если вы хотите ввести описательную фразу, знайте, что псевдоним может состоять из нескольких слов. (Псевдонимы могут быть практически любыми, хотя на них все же накладываются некоторые ограничения: псевдонимы должны заключаться в кавычки, если они совпадают с ключевыми словами SQL, содержат пробелы или другие специальные символы или состоят из одних цифр.) Рассмотрим запрос, извлекающий те же значения данных, что и предыдущий, для имен столбцов вывода которого использованы фразы:

```
mysql> SELECT
    -> DATE_FORMAT(t,'%M %e, %Y') AS 'Date of message',
    -> srcuser AS 'Message sender', size AS 'Number of bytes' FROM mail;
+-----------------+----------------+-----------------+
| Date of message | Message sender | Number of bytes |
+-----------------+----------------+-----------------+
| May 11, 2001    | barb           |           58274 |
| May 12, 2001    | tricia         |          194925 |
| May 12, 2001    | phil           |            1048 |
| May 13, 2001    | barb           |             271 |
...
```

Псевдонимы могут использоваться не только для столбцов, выбираемых из таблиц, но и для любых других столбцов результирующего множества:

```
mysql> SELECT '1+1+1' AS 'The expression', 1+1+1 AS 'The result';
+----------------+------------+
| The expression | The result |
+----------------+------------+
| 1+1+1          |          3 |
+----------------+------------+
```

В данном случае значение первого столбца – это '1+1+1' (кавычки необходимы для того, чтобы значение трактовалось как строка), а значение второго столбца – 1+1+1 (кавычек нет, и MySQL интерпретирует такую запись как выражение и вычисляет его значение). Для псевдонимов выбраны фразы, поясняющие взаимосвязь значений столбцов.

Если вы задаете псевдоним, состоящий из одного слова, а MySQL выражает недовольство, вероятнее всего, псевдоним является зарезервированным словом. Чтобы-таки использовать его, заключите псевдоним в кавычки:

```
mysql> SELECT 1 AS INTEGER;
You have an error in your SQL syntax near 'INTEGER' at line 1
mysql> SELECT 1 AS 'INTEGER';
+---------+
| INTEGER |
+---------+
|       1 |
+---------+
```

3.4. Использование псевдонимов столбцов в программах

Задача

Вы хотите в программе ссылаться на столбец по имени, но он создается вычислением выражения, и его неудобно использовать.

Решение

Используйте псевдоним, чтобы дать столбцу более простое имя.

Обсуждение

Если вы пишете программу, которая выбирает строки в массив и обращается к ним по числовому индексу, то наличие или отсутствие псевдонимов столбцов ничего не меняет, так как псевдоним не влияет на позицию столбца в результирующем множестве. Однако псевдонимы имеют огромное значение, если вы обращаетесь к столбцам вывода по имени, ведь псевдонимы изменяют имена. Используйте это свойство, чтобы работать в программе с простыми именами. Например, если ваш запрос выводит значения времени таблицы из mail, преобразованные посредством DATE_FORMAT(t,'%M %e, %Y'), то это выражение является также именем, которое следует использовать при ссылке на столбец, что не очень удобно. Если использовать конструкцию AS date_sent для создания псевдонима столбца, то далее можно будет ссылаться на столбец по имени date_sent. Вот как мог бы обрабатывать такие значения сценарий Perl DBI:

```
$sth = $dbh->prepare (
                "SELECT srcuser,
                DATE_FORMAT(t,'%M %e, %Y') AS date_sent
```

```
                FROM mail");
$sth->execute ();
while (my $ref = $sth->fetchrow_hashref ())
{
    printf "user: %s, date sent: %s\n", $ref->{srcuser}, $ref->{date_sent};
}
```

А в Java вы бы сделали что-то вроде:

```
Statement s = conn.createStatement ();
s.executeQuery ("SELECT srcuser,"
                + " DATE_FORMAT(t,'%M %e, %Y') AS date_sent"
                + " FROM mail");
ResultSet rs = s.getResultSet ();
while (rs.next ())  // loop through rows of result set
{
    String name = rs.getString ("srcuser");
    String dateSent = rs.getString ("date_sent");
    System.out.println ("user: " + name
                        + ", date sent: " + dateSent);
}
rs.close ();
s.close ();
```

В PHP используйте метод `mysql_fetch_array()` или `mysql_fetch_object()` для извлечения строк результирующего множества в структуру данных, содержащую именованные элементы. В Python используйте класс курсора, который делает так, что строки возвращаются в виде словарей, содержащих пары ключ/значение, где ключами являются имена столбцов (см. рецепт 2.4).

3.5. Объединение столбцов для формирования составных значений

Задача

Вы хотите выводить значения, составленные из нескольких столбцов таблицы.

Решение

Один из способов – применить функцию `CONCAT()`. В дополнение можно использовать псевдоним, чтобы присвоить составному столбцу вывода красивое имя.

Обсуждение

Значения столбцов можно объединять для создания составных значений. Например, это выражение формирует из значений `srcuser` и `srchost` значение в формате адреса электронной почты:

```
CONCAT(srcuser,'@',srchost)
```

Такие выражения часто порождают безобразные названия столбцов, поэтому стоит использовать псевдонимы. В следующем запросе для столбцов вывода, созданных объединением имен пользователей с именами хостов в адреса электронной почты, использованы псевдонимы `sender` и `recipient`:

```
mysql> SELECT
    -> DATE_FORMAT(t,'%M %e, %Y') AS date_sent,
    -> CONCAT(srcuser,'@',srchost) AS sender,
    -> CONCAT(dstuser,'@',dsthost) AS recipient,
    -> size FROM mail;
+--------------+---------------+---------------+---------+
| date_sent    | sender        | recipient     | size    |
+--------------+---------------+---------------+---------+
| May 11, 2001 | barb@saturn   | tricia@mars   |   58274 |
| May 12, 2001 | tricia@mars   | gene@venus    |  194925 |
| May 12, 2001 | phil@mars     | phil@saturn   |    1048 |
| May 13, 2001 | barb@saturn   | tricia@venus  |     271 |
...
```

3.6. Задание выбираемых строк

Задача

Вы хотите видеть не все строки таблицы, а только некоторые.

Решение

Добавьте в запрос инструкцию `WHERE`, которая будет указывать серверу, какие строки следует возвращать.

Обсуждение

Если как-то не уточнить или не ограничить предложение `SELECT`, оно будет извлекать все строки таблицы, а вам зачастую не нужен такой объем информации. Чтобы уточнить состав выбираемых строк, используйте инструкцию `WHERE`, в которой можно указать одно или несколько условий, которым должны соответствовать извлекаемые строки.

Можно производить проверки на равенство, неравенство или относительный порядок. Для некоторых типов столбцов, таких как строки (string), возможен поиск по шаблону (сравнение с образцом). Рассмотрим запросы, которые выбирают столбцы из записей, значения `srchost` которых точно равны строке `'venus'`, лексически меньше строки `'pluto'` или начинаются с буквы `'s'`:

```
mysql> SELECT t, srcuser, srchost  FROM mail WHERE srchost = 'venus';
+---------------------+---------+---------+
| t                   | srcuser | srchost |
+---------------------+---------+---------+
| 2001-05-14 09:31:37 | gene    | venus   |
| 2001-05-14 14:42:21 | barb    | venus   |
```

```
| 2001-05-15 08:50:57 | phil    | venus   |
| 2001-05-16 09:00:28 | gene    | venus   |
| 2001-05-16 23:04:19 | phil    | venus   |
+---------------------+---------+---------+
mysql> SELECT t, srcuser, srchost FROM mail WHERE srchost < 'pluto';
+---------------------+---------+---------+
| t                   | srcuser | srchost |
+---------------------+---------+---------+
| 2001-05-12 12:48:13 | tricia  | mars    |
| 2001-05-12 15:02:49 | phil    | mars    |
| 2001-05-14 11:52:17 | phil    | mars    |
| 2001-05-15 07:17:48 | gene    | mars    |
| 2001-05-15 10:25:52 | gene    | mars    |
| 2001-05-17 12:49:23 | phil    | mars    |
+---------------------+---------+---------+
mysql> SELECT t, srcuser, srchost FROM mail WHERE srchost LIKE 's%';
+---------------------+---------+---------+
| t                   | srcuser | srchost |
+---------------------+---------+---------+
| 2001-05-11 10:15:08 | barb    | saturn  |
| 2001-05-13 13:59:18 | barb    | saturn  |
| 2001-05-14 17:03:01 | tricia  | saturn  |
| 2001-05-15 17:35:31 | gene    | saturn  |
| 2001-05-19 22:21:51 | gene    | saturn  |
+---------------------+---------+---------+
```

Инструкция `WHERE` может проверять несколько условий. Следующее предложение ищет строки, в которых столбец `srcuser` имеет одно из трех указанных значений (задается вопрос о том, когда отправляли почту `gene`, `barb` или `phil`):

```
mysql> SELECT t, srcuser, dstuser FROM mail
    -> WHERE srcuser = 'gene' OR srcuser = 'barb' OR srcuser = 'phil';
+---------------------+---------+---------+
| t                   | srcuser | dstuser |
+---------------------+---------+---------+
| 2001-05-11 10:15:08 | barb    | tricia  |
| 2001-05-12 15:02:49 | phil    | phil    |
| 2001-05-13 13:59:18 | barb    | tricia  |
| 2001-05-14 09:31:37 | gene    | barb    |
...
```

Запросы, подобные только что приведенному, где определенный столбец проверяется на наличие одного из нескольких значений, часто можно упростить при помощи оператора `IN()`. Результат `IN()` истинен, если значение столбца равно одному из значений списка аргументов:

```
mysql> SELECT t, srcuser, dstuser FROM mail
    -> WHERE srcuser IN ('gene','barb','phil');
+---------------------+---------+---------+
| t                   | srcuser | dstuser |
+---------------------+---------+---------+
| 2001-05-11 10:15:08 | barb    | tricia  |
```

```
| 2001-05-12 15:02:49 | phil    | phil    |
| 2001-05-13 13:59:18 | barb    | tricia  |
| 2001-05-14 09:31:37 | gene    | barb    |
...
```

Можно указать несколько условий для разных столбцов. Например, найдем сообщения отправителя `barb`, которые были отправлены получателю `tricia`:

```
mysql> SELECT * FROM mail WHERE srcuser = 'barb' AND dstuser = 'tricia';
+---------------------+---------+---------+---------+---------+-------+
| t                   | srcuser | srchost | dstuser | dsthost | size  |
+---------------------+---------+---------+---------+---------+-------+
| 2001-05-11 10:15:08 | barb    | saturn  | tricia  | mars    | 58274 |
| 2001-05-13 13:59:18 | barb    | saturn  | tricia  | venus   |   271 |
+---------------------+---------+---------+---------+---------+-------+
```

Единственное требование, предъявляемое к сравнениям, – это синтаксическая (но не семантическая!) корректность. Например, сложно представить себе смысл такого сравнения, но, тем не менее, оно будет успешно выполнено MySQL:[1]

```
SELECT * FROM mail WHERE srcuser + dsthost < size
```

Запрос, не возвративший строк, – это невыполненный запрос?

Если сформированное вами предложение `SELECT` не возвращает строк, означает ли это, что запрос не выполнен? Возможны варианты. Если отсутствие результирующего множества вызвано проблемами с синтаксической некорректностью запроса или ссылкой на несуществующие таблицы или столбцы, то запрос действительно не выполнен. Произошел сбой, и вам следует выяснить, почему программа пытается выдать некорректный запрос.

Если запрос выполняется без ошибок, но ничего не возвращает, это всего-навсего означает, что не найдено строк, соответствующих условию инструкции `WHERE`:

```
mysql> SELECT * FROM mail WHERE srcuser = 'no-such-user';
Empty set (0.01 sec)
```

Это *не* неуспешно выполненный запрос. Он успешно работает и формирует результат, результирующее множество же оказалось пустым лишь потому, что ни в одной строке значение `srcuser` не равно `no-such-user`.

Столбцы не обязательно сравнивать с константами, вы можете сравнивать их с другими столбцами. Предположим, что у вас есть таблица `cd` (компакт-

[1] Если вы из любопытства попробуете запустить этот запрос, то как вы будете трактовать полученный результат?

диски), содержащая столбцы year (год), artist (исполнитель) и title (название):[1]

```
mysql> SELECT year, artist, title FROM cd;
+------+-----------------+----------------------+
| year | artist          | title                |
+------+-----------------+----------------------+
| 1990 | Iona            | Iona                 |
| 1992 | Charlie Peacock | Lie Down in the Grass |
| 1993 | Iona            | Beyond These Shores  |
| 1987 | The 77s         | The 77s              |
| 1990 | Michael Gettel  | Return               |
| 1989 | Richard Souther | Cross Currents       |
| 1996 | Charlie Peacock | strangelanguage      |
| 1982 | Undercover      | Undercover           |
...
```

Тогда вы можете найти все компакт-диски, названия которых совпадают с именем исполнителя, сравнивая один столбец таблицы с другим:

```
mysql> SELECT year, artist, title FROM cd WHERE artist = title;
+------+------------+------------+
| year | artist     | title      |
+------+------------+------------+
| 1990 | Iona       | Iona       |
| 1987 | The 77s    | The 77s    |
| 1982 | Undercover | Undercover |
+------+------------+------------+
```

Особым случаем сравнения столбцов внутри таблицы является сравнение столбца с самим собой. Предположим, что вы собираете почтовые марки и храните сведения о своей коллекции в таблице stamp, которая содержит столбцы с идентификатором и годом выпуска каждого экземпляра. Если вы знаете, что какая-то марка имеет идентификатор 42, и хотите использовать соответствующее ему значение столбца year для нахождения всех других марок вашей коллекции, выпущенных в том же году, можно сравнивать год с годом, то есть столбец year со столбцом year:

```
mysql> SELECT stamp.* FROM stamp, stamp AS stamp2
    -> WHERE stamp.year = stamp2.year AND stamp2.id = 42 AND stamp.id != 42;
+-----+------+-------------------------+
| id  | year | description             |
+-----+------+-------------------------+
|  97 | 1987 | 1-cent transition stamp |
| 161 | 1987 | aviation stamp          |
+-----+------+-------------------------+
```

[1] Если вы уже читали какие-то книги по базам данных, вполне вероятно, что у вас есть такая таблица. Многим нравится в качестве упражнения создать базу данных для отслеживания своей коллекции компакт-дисков. Я бы сказал, что база данных CD уступает по популярности только базе данных деталей и поставщиков.

Такие запросы используют объединение таблицы с самой собой (self-join), псевдонимы таблиц и ссылки на столбцы по имени таблицы. Но сейчас я не стану углубляться дальше – все это вы найдете в главе 12.

3.7. Инструкция WHERE и псевдонимы столбцов

Задача

Вы хотите сослаться на псевдоним столбца в инструкции WHERE.

Решение

Очень жаль, но делать это нельзя.

Обсуждение

Нельзя использовать псевдонимы столбцов в инструкции WHERE. Поэтому следующий запрос работать не будет:

```
mysql> SELECT t, srcuser, dstuser, size/1024 AS kilobytes
    -> FROM mail WHERE kilobytes > 500;
ERROR 1054 at line 1: Unknown column 'kilobytes' in 'where clause'
```

Ошибка возникает из-за того, что псевдонимы именуют столбцы *вывода*, в то время как инструкция WHERE работает со столбцами *ввода*, чтобы определить, какие строки следует отобрать для вывода. Чтобы сделать запрос корректным, замените псевдоним в инструкции WHERE именем столбца или выражением, которое представляет псевдоним:

```
mysql> SELECT t, srcuser, dstuser, size/1024 AS kilobytes
    -> FROM mail WHERE size/1024 > 500;
+---------------------+---------+---------+-----------+
| t                   | srcuser | dstuser | kilobytes |
+---------------------+---------+---------+-----------+
| 2001-05-14 17:03:01 | tricia  | phil    |   2338.36 |
| 2001-05-15 10:25:52 | gene    | tricia  |    975.13 |
+---------------------+---------+---------+-----------+
```

3.8. Отображение результатов операций сравнения с целью контроля их выполнения

Задача

Вас интересует, как работает сравнение в инструкции WHERE. Или почему оно не работает.

Решение

Для получения информации о сравнении выводите его результат (полезно при диагностике и отладке программ).

Обсуждение

Обычно вы помещаете операции сравнения в инструкцию WHERE запроса и используете их для указания того, какие записи следует отображать:

```
mysql> SELECT * FROM mail WHERE srcuser < 'c' AND size > 5000;
+---------------------+---------+---------+---------+---------+-------+
| t                   | srcuser | srchost | dstuser | dsthost | size  |
+---------------------+---------+---------+---------+---------+-------+
| 2001-05-11 10:15:08 | barb    | saturn  | tricia  | mars    | 58274 |
| 2001-05-14 14:42:21 | barb    | venus   | barb    | venus   | 98151 |
+---------------------+---------+---------+---------+---------+-------+
```

Но иногда хотелось бы видеть собственно результат сравнения (например, если вы не уверены в том, что сравнение работает именно так, как нужно). Поместите выражение сравнения в список столбцов вывода (при желании можно включить туда и сравниваемые значения):

```
mysql> SELECT srcuser, srcuser < 'c', size, size > 5000 FROM mail;
+---------+---------------+---------+-------------+
| srcuser | srcuser < 'c' | size    | size > 5000 |
+---------+---------------+---------+-------------+
| barb    |             1 |   58274 |           1 |
| tricia  |             0 |  194925 |           1 |
| phil    |             0 |    1048 |           0 |
| barb    |             1 |     271 |           0 |
...
```

Вывод результатов сравнения особенно полезен при написании запросов, которые выполняют проверки без обращения к таблицам:

```
mysql> SELECT 'a' = 'A';
+-----------+
| 'a' = 'A' |
+-----------+
|         1 |
+-----------+
```

Из результата запроса видно, что операция сравнения строк по умолчанию не чувствительна к регистру (кстати, это полезно знать).

3.9. Инвертирование, или отрицание условий запроса

Задача

Вы знаете, как написать запрос, отвечающий на поставленный вопрос, и теперь хотите получить ответ на противоположный вопрос.

Решение

Используйте оператор отрицания для изменения условий инструкции WHERE на обратные.

Обсуждение

Условия WHERE в запросе можно изменить на обратные, задав тем самым противоположный вопрос. Рассмотрим запрос, выявляющий отправку пользователями сообщений себе самим:

```
mysql> SELECT * FROM mail WHERE srcuser = dstuser;
+---------------------+---------+---------+---------+---------+-------+
| t                   | srcuser | srchost | dstuser | dsthost | size  |
+---------------------+---------+---------+---------+---------+-------+
| 2001-05-12 15:02:49 | phil    | mars    | phil    | saturn  |  1048 |
| 2001-05-14 14:42:21 | barb    | venus   | barb    | venus   | 98151 |
| 2001-05-15 07:17:48 | gene    | mars    | gene    | saturn  |  3824 |
| 2001-05-15 08:50:57 | phil    | venus   | phil    | venus   |   978 |
| 2001-05-15 17:35:31 | gene    | saturn  | gene    | mars    |  3856 |
| 2001-05-19 22:21:51 | gene    | saturn  | gene    | venus   | 23992 |
+---------------------+---------+---------+---------+---------+-------+
```

Чтобы обратить этот запрос, то есть найти записи, в которых пользователи отправляют письма кому угодно, только *не* себе, замените оператор сравнения = (равно) на != (не равно):

```
mysql> SELECT * FROM mail WHERE srcuser != dstuser;
+---------------------+---------+---------+---------+---------+---------+
| t                   | srcuser | srchost | dstuser | dsthost | size    |
+---------------------+---------+---------+---------+---------+---------+
| 2001-05-11 10:15:08 | barb    | saturn  | tricia  | mars    |   58274 |
| 2001-05-12 12:48:13 | tricia  | mars    | gene    | venus   |  194925 |
| 2001-05-13 13:59:18 | barb    | saturn  | tricia  | venus   |     271 |
| 2001-05-14 09:31:37 | gene    | venus   | barb    | mars    |    2291 |
...
```

В более сложном запросе, использующем два условия, можно получить ответ на вопрос о том, кто отправляет письма самому себе и на ту же самую машину:

```
mysql> SELECT * FROM mail WHERE srcuser = dstuser AND srchost = dsthost;
+---------------------+---------+---------+---------+---------+-------+
| t                   | srcuser | srchost | dstuser | dsthost | size  |
+---------------------+---------+---------+---------+---------+-------+
| 2001-05-14 14:42:21 | barb    | venus   | barb    | venus   | 98151 |
| 2001-05-15 08:50:57 | phil    | venus   | phil    | venus   |   978 |
+---------------------+---------+---------+---------+---------+-------+
```

Для изменения условий этого запроса на обратные надо заменить не только операторы = на !=, но и AND на OR:

```
mysql> SELECT * FROM mail WHERE srcuser != dstuser OR srchost != dsthost;
+---------------------+---------+---------+---------+---------+---------+
| t                   | srcuser | srchost | dstuser | dsthost | size    |
+---------------------+---------+---------+---------+---------+---------+
| 2001-05-11 10:15:08 | barb    | saturn  | tricia  | mars    |   58274 |
| 2001-05-12 12:48:13 | tricia  | mars    | gene    | venus   |  194925 |
| 2001-05-12 15:02:49 | phil    | mars    | phil    | saturn  |    1048 |
```

```
| 2001-05-13 13:59:18 | barb    | saturn  | tricia  | venus   |     271 |
...
```

Может быть, вам покажется более простым такой способ: поместите все исходное выражение в скобки и выполните операцию отрицания для всего этого выражения, поставив перед ним NOT:

```
mysql> SELECT * FROM mail WHERE NOT (srcuser = dstuser AND srchost = dsthost);
+---------------------+---------+---------+---------+---------+---------+
| t                   | srcuser | srchost | dstuser | dsthost | size    |
+---------------------+---------+---------+---------+---------+---------+
| 2001-05-11 10:15:08 | barb    | saturn  | tricia  | mars    |   58274 |
| 2001-05-12 12:48:13 | tricia  | mars    | gene    | venus   |  194925 |
| 2001-05-12 15:02:49 | phil    | mars    | phil    | saturn  |    1048 |
| 2001-05-13 13:59:18 | barb    | saturn  | tricia  | venus   |     271 |
...
```

См. также

Если условие касается столбца, допускающего значения NULL, то инвертирование условий будет несколько сложнее (см. подробности в рецепте 3.12).

3.10. Удаление повторяющихся строк

Задача

Вывод запроса содержит повторяющиеся записи. Вы хотите избавиться от них.

Решение

Используйте ключевое слово DISTINCT.

Обсуждение

Некоторые запросы могут порождать результирующее множество, содержащее дублирующиеся записи. Например, чтобы узнать, кто писал письма, вы можете обратиться к таблице mail так:

```
mysql> SELECT srcuser FROM mail;
+---------+
| srcuser |
+---------+
| barb    |
| tricia  |
| phil    |
| barb    |
| gene    |
| phil    |
| barb    |
| tricia  |
| gene    |
```

```
| phil    |
| gene    |
| gene    |
| gene    |
| phil    |
| phil    |
| gene    |
+---------+
```

Очевидно, что полученная информация избыточна. DISTINCT удалит повторяющиеся записи, формируя множество уникальных значений:

```
mysql> SELECT DISTINCT srcuser FROM mail;
+---------+
| srcuser |
+---------+
| barb    |
| tricia  |
| phil    |
| gene    |
+---------+
```

DISTINCT может обрабатывать и многостолбцовый вывод. Следующий запрос показывает, какие даты представлены в таблице mail:

```
mysql> SELECT DISTINCT YEAR(t), MONTH(t), DAYOFMONTH(t) FROM mail;
+---------+----------+---------------+
| YEAR(t) | MONTH(t) | DAYOFMONTH(t) |
+---------+----------+---------------+
|    2001 |        5 |            11 |
|    2001 |        5 |            12 |
|    2001 |        5 |            13 |
|    2001 |        5 |            14 |
|    2001 |        5 |            15 |
|    2001 |        5 |            16 |
|    2001 |        5 |            17 |
|    2001 |        5 |            19 |
+---------+----------+---------------+
```

Чтобы подсчитать количество уникальных значений, выполните:

```
mysql> SELECT COUNT(DISTINCT srcuser) FROM mail;
+-------------------------+
| COUNT(DISTINCT srcuser) |
+-------------------------+
|                       4 |
+-------------------------+
```

Для COUNT(DISTINCT) требуется версия MySQL 3.23.2 и выше.

См. также

Мы еще поговорим о DISTINCT в главе 7. Удаление дубликатов подробно обсуждается в главе 14.

3.11. Обработка значений NULL

Задача

Вы безуспешно пытаетесь сравнить значения столбцов с NULL.

Решение

Необходимо использовать соответствующие операторы сравнения: IS NULL, IS NOT NULL или <=>.

Обсуждение

Для работы со значением NULL применяются специальные условия. Для проверки столбцов на значения NULL нельзя использовать = NULL или != NULL. Про такие сравнения невозможно сказать, истинны они или ложны, поэтому они никогда не будут работать. Не выполняется даже сравнение NULL = NULL. (Почему? Потому что невозможно определить, равна ли одна неизвестная величина какой-то другой неизвестной величине.)

Чтобы проверить, содержат ли столбцы значение NULL, используйте оператор IS NULL или IS NOT NULL. Предположим, что таблица taxpayer содержит фамилии и идентификационные номера налогоплательщиков. NULL в качестве номера означает, что значение неизвестно:

```
mysql> SELECT * FROM taxpayer;
+---------+--------+
| name    | id     |
+---------+--------+
| bernina | 198-48 |
| bertha  | NULL   |
| ben     | NULL   |
| bill    | 475-83 |
+---------+--------+
```

Можете убедиться в том, что = и != не работают со значениями NULL:

```
mysql> SELECT * FROM taxpayer WHERE id = NULL;
Empty set (0.00 sec)
mysql> SELECT * FROM taxpayer WHERE id != NULL;
Empty set (0.01 sec)
```

Чтобы найти записи, в которых столбец id содержит или не содержит NULL, следует строить запросы так:

```
mysql> SELECT * FROM taxpayer WHERE id IS NULL;
+--------+------+
| name   | id   |
+--------+------+
| bertha | NULL |
| ben    | NULL |
+--------+------+
```

```
mysql> SELECT * FROM taxpayer WHERE id IS NOT NULL;
+---------+--------+
| name    | id     |
+---------+--------+
| bernina | 198-48 |
| bill    | 475-83 |
+---------+--------+
```

Начиная с версии MySQL 3.23 вы также можете использовать для сравнения значений оператор <=>, который (в отличие от =) даст «истину» даже для двух значений NULL:

```
mysql> SELECT NULL = NULL, NULL <=> NULL;
+-------------+---------------+
| NULL = NULL | NULL <=> NULL |
+-------------+---------------+
|        NULL |             1 |
+-------------+---------------+
```

См. также

Значения NULL особым образом ведут себя также при сортировке и суммировании записей. См. рецепты 6.5 и 7.8.

3.12. Инвертирование условия для столбца, содержащего значения NULL

Задача

Вы пытаетесь инвертировать условие для столбца, в котором присутствует NULL, но ничего не получается.

Решение

При инвертировании NULL тоже ведет себя нетривиально.

Обсуждение

В рецепте 3.9 рассказывалось о том, что вы можете изменить условия запроса на обратные за счет замены операторов сравнения и логических операторов, а также при помощи NOT. Но если столбец допускает использование NULL, то возможны проблемы. Давайте вернемся к таблице taxpayer из рецепта 3.11:

```
+---------+--------+
| name    | id     |
+---------+--------+
| bernina | 198-48 |
| bertha  | NULL   |
| ben     | NULL   |
| bill    | 475-83 |
+---------+--------+
```

Предположим, что у вас есть запрос, ищущий записи, в которых значение идентификатора налогоплательщика лексически меньше, чем 200-00:

```
mysql> SELECT * FROM taxpayer WHERE id < '200-00';
+---------+--------+
| name    | id     |
+---------+--------+
| bernina | 198-48 |
+---------+--------+
```

Если инвертировать условие, использовав >= вместо <, ожидаемого результата не получится. Такой запрос подходит лишь в случае, если вы хотите выбрать только те записи, для которых известен (не NULL) идентификатор:

```
mysql> SELECT * FROM taxpayer WHERE id >= '200-00';
+------+--------+
| name | id     |
+------+--------+
| bill | 475-83 |
+------+--------+
```

Если же вы хотите получить все строки, не возвращенные исходным запросом, то простым изменением оператора сравнения этого не добиться. Значения NULL не удовлетворяют ни одному из условий < и >=, так что необходимо добавить для них специальную инструкцию:

```
mysql> SELECT * FROM taxpayer WHERE id >= '200-00' OR id IS NULL;
+--------+--------+
| name   | id     |
+--------+--------+
| bertha | NULL   |
| ben    | NULL   |
| bill   | 475-83 |
+--------+--------+
```

3.13. Использование в программах операций сравнения с участием NULL

Задача

Вы написали программу, формирующую запрос, но она не работает для значений NULL.

Решение

Попробуйте написать запрос отдельно для значений NULL и не-NULL.

Обсуждение

Необходимость использования отдельного оператора сравнения для значений NULL вызывает небольшое неудобство при создании строки запроса в программе. Если переменная может принимать значение NULL, необходимо учи-

тывать это при выполнении сравнений для данной переменной. Например, в Perl undef представляет значение NULL, поэтому для формирования предложения, которое ищет записи таблицы taxpayer, совпадающие с некоторым произвольным значением из переменной $id, нельзя поступить так:

```
$sth = $dbh->prepare ("SELECT * FROM taxpayer WHERE id = ?");
$sth->execute ($id);
```

Если $id содержит NULL, предложение не будет выполняться, так как превратится в следующее:

```
SELECT * FROM taxpayer WHERE id = NULL
```

Такое предложение не возвращает записей – сравнение с NULL всегда ложно. Чтобы учесть возможность того, что в $id содержится NULL, построим запрос, используя соответствующий оператор сравнения:

```
$operator = (defined ($id) ? "=" : "IS");
$sth = $dbh->prepare ("SELECT * FROM taxpayer WHERE id $operator ?");
$sth->execute ($id);
```

Тогда для таких значений $id, как undef (NULL) и 43 (не-NULL), запросы будут следующими:

```
SELECT * FROM taxpayer WHERE id IS NULL
SELECT * FROM taxpayer WHERE id = 43
```

При проверке на неравенство задайте $operator так:

```
$operator = (defined ($id) ? "!=" : "IS NOT");
```

3.14. Сопоставление значениям NULL других значений при выводе

Задача

Вывод запроса содержит значения NULL, а вам хотелось бы видеть что-то более осмысленное, например «Unknown».

Решение

При выводе сопоставьте значениям NULL какое-либо другое значение. Этот же прием можно использовать для перехвата ошибок деления на ноль.

Обсуждение

Иногда бывает полезно выводить вместо NULL какое-нибудь другое специальное значение, которое более понятно выглядит в контексте вашего приложения. Если NULL-значения идентификатора (столбец id) в таблице taxpayer означают, что номер неизвестен (unknown), вы можете при выводе сопоставить значениям NULL символьную строку Unknown (используя IF()):

```
mysql> SELECT name, IF(id IS NULL,'Unknown', id) AS 'id' FROM taxpayer;
```

```
+---------+---------+
| name    | id      |
+---------+---------+
| bernina | 198-48  |
| bertha  | Unknown |
| ben     | Unknown |
| bill    | 475-83  |
+---------+---------+
```

В действительности так можно поступать с любыми значениями, но в случае с NULL это наиболее эффективно, так как им придают разнообразнейшие толкования: неизвестно, отсутствует, еще не определено, вне диапазона и т. д.

Чтобы сократить запрос, можно использовать функцию IFNULL(), которая проверяет свой первый аргумент и возвращает его, если это не NULL, в противном случае возвращается второй аргумент.

```
mysql> SELECT name, IFNULL(id,'Unknown') AS 'id' FROM taxpayer;
+---------+---------+
| name    | id      |
+---------+---------+
| bernina | 198-48  |
| bertha  | Unknown |
| ben     | Unknown |
| bill    | 475-83  |
+---------+---------+
```

Другими словами, эквивалентны две такие проверки:

```
IF(выражение1 IS NOT NULL, выражение1, выражение2)
IFNULL(выражение1, выражение2)
```

С точки зрения читабельности IF() понятнее, следовательно, удобнее, чем IFNULL(). А вот вычислительные возможности выше у IFNULL(): дело в том, что здесь *выражение1* никогда не вычисляется дважды, в то время как при использовании IF() это вполне возможно.

Функции IF() и IFNULL() особо полезны для перехвата операций деления на ноль и вывода вместо этого чего-то конкретного. Например, среднее число очков отбивающего игрока (batting averages) в бейсболе вычисляется как отношение числа отбитых подач (hits) к числу выступлений отбивающего. Но если игрок ни разу не выступал как отбивающий, то отношение не определено:

```
mysql> SET @hits = 0, @atbats = 0;
mysql> SELECT @hits, @atbats, @hits/@atbats AS 'batting average';
+-------+---------+-----------------+
| @hits | @atbats | batting average |
+-------+---------+-----------------+
|     0 |       0 |            NULL |
+-------+---------+-----------------+
```

Чтобы выводить в подобных случаях 0, сделайте следующее:

```
mysql> SET @hits = 0, @atbats = 0;
mysql> SELECT @hits, @atbats, IFNULL(@hits/@atbats,0) AS 'batting average';
```

```
+-------+---------+-----------------+
| @hits | @atbats | batting average |
+-------+---------+-----------------+
|     0 |       0 |               0 |
+-------+---------+-----------------+
```

Такой же прием можно использовать для вычисления среднего числа очков (earned run average) подающего игрока (pitcher) при отсутствии подач. Часто встречаются и такие варианты:

```
IFNULL(выражение,'Missing')
IFNULL(выражение,'N/A')
IFNULL(выражение,'Unknown')
```

3.15. Упорядочивание результирующего множества

Задача

Результаты запроса отсортированы не так, как хотелось бы.

Решение

MySQL не умеет читать ваши мысли. Добавьте инструкцию `ORDER BY`, чтобы сообщить MySQL о том, как именно должно быть упорядочено результирующее множество.

Обсуждение

Когда вы выбираете строки, сервер MySQL может возвращать их в любом порядке, если только вы не проинструктируете его насчет порядка их отображения. Существует множество способов сортировки. В двух словах: вы упорядочиваете результирующее множество, добавляя инструкцию `ORDER BY`, в которой указывается имя столбца (или столбцов), по которому вы хотите выполнить сортировку:

```
mysql> SELECT * FROM mail WHERE size > 100000 ORDER BY size;
+---------------------+---------+---------+---------+---------+---------+
| t                   | srcuser | srchost | dstuser | dsthost | size    |
+---------------------+---------+---------+---------+---------+---------+
| 2001-05-12 12:48:13 | tricia  | mars    | gene    | venus   |  194925 |
| 2001-05-15 10:25:52 | gene    | mars    | tricia  | saturn  |  998532 |
| 2001-05-14 17:03:01 | tricia  | saturn  | phil    | venus   | 2394482 |
+---------------------+---------+---------+---------+---------+---------+
mysql> SELECT * FROM mail WHERE dstuser = 'tricia'
    -> ORDER BY srchost, srcuser;
+---------------------+---------+---------+---------+---------+--------+
| t                   | srcuser | srchost | dstuser | dsthost | size   |
+---------------------+---------+---------+---------+---------+--------+
| 2001-05-15 10:25:52 | gene    | mars    | tricia  | saturn  | 998532 |
```

```
| 2001-05-14 11:52:17 | phil    | mars    | tricia  | saturn  |   5781 |
| 2001-05-17 12:49:23 | phil    | mars    | tricia  | saturn  |    873 |
| 2001-05-11 10:15:08 | barb    | saturn  | tricia  | mars    |  58274 |
| 2001-05-13 13:59:18 | barb    | saturn  | tricia  | venus   |    271 |
+---------------------+---------+---------+---------+---------+--------+
```

Чтобы упорядочить столбец в обратном порядке (по убыванию), добавьте после его имени в инструкции `ORDER BY` ключевое слово `DESC`:

```
mysql> SELECT * FROM mail WHERE size > 50000 ORDER BY size DESC;
+---------------------+---------+---------+---------+---------+---------+
| t                   | srcuser | srchost | dstuser | dsthost | size    |
+---------------------+---------+---------+---------+---------+---------+
| 2001-05-14 17:03:01 | tricia  | saturn  | phil    | venus   | 2394482 |
| 2001-05-15 10:25:52 | gene    | mars    | tricia  | saturn  |  998532 |
| 2001-05-12 12:48:13 | tricia  | mars    | gene    | venus   |  194925 |
| 2001-05-14 14:42:21 | barb    | venus   | barb    | venus   |   98151 |
| 2001-05-11 10:15:08 | barb    | saturn  | tricia  | mars    |   58274 |
+---------------------+---------+---------+---------+---------+---------+
```

3.16. Выбор начальных или конечных записей результирующего множества

Задача

Вы хотите вывести только несколько строк результирующего множества, например первую или пять последних.

Решение

Используйте инструкцию `LIMIT`, возможно, в сочетании с инструкцией `ORDER BY`.

Обсуждение

MySQL поддерживает инструкцию `LIMIT`, которая предписывает серверу возвратить только часть результирующего множества. `LIMIT` — это собственное расширение MySQL для языка SQL, весьма полезное в тех случаях, когда результирующее множество содержит больше строк, чем вы хотели бы видеть за раз. Инструкция позволяет извлекать только первую часть результирующего множества или произвольный раздел множества. Обычно `LIMIT` применяется при решении задач следующих видов:

- Получение ответов на вопросы о первом и последнем, самом большом и самом маленьком, самом новом и самом старом, наиболее дорогом и дешевом и т. д.
- Разбиение результирующего множества на части для постепенной обработки. Такой прием часто используется в веб-приложениях для отображения объемного результата поиска на нескольких страницах. Разбиение результата на части и вывод небольших страниц облегчает восприятие.

В следующих примерах будет использоваться таблица `profile` (см. главу 2), которая выглядит так:

```
mysql> SELECT * FROM profile;
+----+---------+------------+-------+----------------------+------+
| id | name    | birth      | color | foods                | cats |
+----+---------+------------+-------+----------------------+------+
|  1 | Fred    | 1970-04-13 | black | lutefisk,fadge,pizza |    0 |
|  2 | Mort    | 1969-09-30 | white | burrito,curry,eggroll |    3 |
|  3 | Brit    | 1957-12-01 | red   | burrito,curry,pizza  |    1 |
|  4 | Carl    | 1973-11-02 | red   | eggroll,pizza        |    4 |
|  5 | Sean    | 1963-07-04 | blue  | burrito,curry        |    5 |
|  6 | Alan    | 1965-02-14 | red   | curry,fadge          |    1 |
|  7 | Mara    | 1968-09-17 | green | lutefisk,fadge       |    1 |
|  8 | Shepard | 1975-09-02 | black | curry,pizza          |    2 |
|  9 | Dick    | 1952-08-20 | green | lutefisk,fadge       |    0 |
| 10 | Tony    | 1960-05-01 | white | burrito,pizza        |    0 |
+----+---------+------------+-------+----------------------+------+
```

Чтобы выбрать первые *n* записей результирующего множества, добавьте в конец предложения `SELECT` инструкцию `LIMIT` *n*:

```
mysql> SELECT * FROM profile LIMIT 1;
+----+------+------------+-------+----------------------+------+
| id | name | birth      | color | foods                | cats |
+----+------+------------+-------+----------------------+------+
|  1 | Fred | 1970-04-13 | black | lutefisk,fadge,pizza |    0 |
+----+------+------------+-------+----------------------+------+
mysql> SELECT * FROM profile LIMIT 5;
+----+------+------------+-------+----------------------+------+
| id | name | birth      | color | foods                | cats |
+----+------+------------+-------+----------------------+------+
|  1 | Fred | 1970-04-13 | black | lutefisk,fadge,pizza |    0 |
|  2 | Mort | 1969-09-30 | white | burrito,curry,eggroll |    3 |
|  3 | Brit | 1957-12-01 | red   | burrito,curry,pizza  |    1 |
|  4 | Carl | 1973-11-02 | red   | eggroll,pizza        |    4 |
|  5 | Sean | 1963-07-04 | blue  | burrito,curry        |    5 |
+----+------+------------+-------+----------------------+------+
```

Однако поскольку строки результирующих множеств этих запросов не отсортированы в каком-то специальном порядке, они могут не нести особого смысла. Обычно в таких задачах принято использовать инструкцию `ORDER BY` для упорядочивания результата. Тогда при помощи инструкции `LIMIT` можно найти наибольшее и наименьшее значения. Например, чтобы найти строку с минимальной (самой ранней) датой рождения, выполним сортировку по столбцу `birth` и используем `LIMIT 1` для извлечения первой строки:

```
mysql> SELECT * FROM profile ORDER BY birth LIMIT 1;
+----+------+------------+-------+----------------+------+
| id | name | birth      | color | foods          | cats |
+----+------+------------+-------+----------------+------+
|  9 | Dick | 1952-08-20 | green | lutefisk,fadge |    0 |
+----+------+------------+-------+----------------+------+
```

Благодаря тому, что MySQL сначала обрабатывает инструкцию ORDER BY, сортируя строки, а затем уже применяет LIMIT, все работает правильно. Чтобы найти строку с последней датой рождения (самой поздней), используем тот же самый запрос, только сортировку будем выполнять по убыванию:

```
mysql> SELECT * FROM profile ORDER BY birth DESC LIMIT 1;
+----+---------+------------+-------+-------------+------+
| id | name    | birth      | color | foods       | cats |
+----+---------+------------+-------+-------------+------+
|  8 | Shepard | 1975-09-02 | black | curry,pizza |    2 |
+----+---------+------------+-------+-------------+------+
```

Вы можете получить те же результаты, если выполните запросы без инструкции LIMIT и не будете смотреть ни на какие строки, кроме первой. Преимущество использования LIMIT в том, что сервер возвращает только первую запись, а остальным не приходится путешествовать по сети. Такой способ гораздо эффективнее, чем извлечение всего результирующего множества, в котором все строки, кроме одной, вам не нужны.

Вы можете выполнять упорядочивание по любому столбцу или столбцам. Чтобы найти строку для человека, у которого наибольшее количество кошек, сортируйте по столбцу cats:

```
mysql> SELECT * FROM profile ORDER BY cats DESC LIMIT 1;
+----+------+------------+-------+---------------+------+
| id | name | birth      | color | foods         | cats |
+----+------+------------+-------+---------------+------+
|  5 | Sean | 1963-07-04 | blue  | burrito,curry |    5 |
+----+------+------------+-------+---------------+------+
```

Однако имейте в виду, что использование LIMIT *n* для выбора «*n* наименьших» или «*n* наибольших» значений может привести не к тому выводу, который вы ожидаете. Подробности формирования запросов LIMIT обсуждаются в рецепте 3.18.

Чтобы найти самый первый день рожденья в пределах календарного года, отсортируйте значения birth по месяцу и дню:

```
mysql> SELECT name, DATE_FORMAT(birth,'%m-%e') AS birthday
    -> FROM profile ORDER BY birthday LIMIT 1;
+------+----------+
| name | birthday |
+------+----------+
| Alan | 02-14    |
+------+----------+
```

Отметьте, что фактически LIMIT *n* означает «вернуть не более *n* строк». Если вы укажете LIMIT 10, а результирующее множество содержит всего 3 строки, то сервер вернет 3 строки.

См. также

Вы можете использовать LIMIT в сочетании с RAND() для выполнения случайных выборок из набора записей (см. главу 13).

Начиная с MySQL версии 3.22.7 вы можете использовать LIMIT для ограничения действия предложения DELETE для подмножества строк, которое иначе было бы удалено. В версии MySQL 3.23.3 то же самое справедливо и для UPDATE. Такую возможность удобно применять в сочетании с инструкцией WHERE. Например, если таблица содержит пять экземпляров записи, вы можете выбрать их в предложении DELETE с соответствующей инструкцией WHERE, затем удалить дубликаты, добавив в конец предложения LIMIT 4. Останется всего один экземпляр записи. Дополнительные сведения об использовании LIMIT для удаления повторяющихся записей приведены в главе 14.

3.17. Выбор строк из середины результирующего множества

Задача

Вас не интересуют первые и последние строки результирующего множества. Вам нужны строки из середины, например строки 21–40.

Решение

И такую операцию может выполнить LIMIT. Необходимо лишь указать (в дополнение к количеству строк для выборки) ту строку, с которой вы хотели бы начать.

Обсуждение

LIMIT *n* сообщает серверу, что следует вернуть первые *n* строк результирующего множества. LIMIT может принимать и два аргумента, что дает вам возможность выбирать произвольный раздел результата. Аргументы указывают, сколько строк необходимо пропустить, а сколько – извлечь. То есть с помощью LIMIT вы можете, например, пропустить две строки и вывести следующую за ними, ответив тем самым на вопрос: «Какое значение является *третьим* по убыванию (возрастанию)?», на который не так-то легко ответить с помощью функции MIN() или MAX():

```
mysql> SELECT * FROM profile ORDER BY birth LIMIT 2,1;
+----+------+------------+-------+--------------+------+
| id | name | birth      | color | foods        | cats |
+----+------+------------+-------+--------------+------+
| 10 | Tony | 1960-05-01 | white | burrito,pizza |    0 |
+----+------+------------+-------+--------------+------+
mysql> SELECT * FROM profile ORDER BY birth DESC LIMIT 2,1;
+----+------+------------+-------+---------------------+------+
| id | name | birth      | color | foods               | cats |
```

```
+----+------+------------+-------+---------------------+------+
|  1 | Fred | 1970-04-13 | black | lutefisk,fadge,pizza |    0 |
+----+------+------------+-------+---------------------+------+
```

Задавая два аргумента для LIMIT, вы сможете разбить результирующее множество на более мелкие фрагменты. Например, чтобы извлекать результирующее множество порциями по 20 строк, выполните несколько раз одно и то же предложение SELECT с разными инструкциями LIMIT:

```
SELECT ... FROM ... ORDER BY ... LIMIT 0, 20;   выбрать первые 20 строк
SELECT ... FROM ... ORDER BY ... LIMIT 20, 20;  пропустить 20 строк,
                                                выбрать следующие 20
SELECT ... FROM ... ORDER BY ... LIMIT 40, 20;  пропустить 40 строк,
                                                выбрать следующие 20
```

и т. д.

Веб-разработчики часто используют LIMIT для разбиения большого результата поиска на более мелкие и легко управляемые части, чтобы выводить его на нескольких страницах. Мы вернемся к обсуждению этого приема в рецепте 18.10.

Если вы хотите узнать, насколько велико результирующее множество, чтобы определить, на сколько порций его следует разбить, то можете сначала выполнить запрос COUNT(). Используйте в этом запросе такую же инструкцию WHERE, как и при извлечении строк. Например, если вы хотите вывести таблицу profile, отсортированную по фамилиям, фрагментами по четыре записи, то можете сначала вычислить количество записей таким запросом:

```
mysql> SELECT COUNT(*) FROM profile;
+----------+
| COUNT(*) |
+----------+
|       10 |
+----------+
```

Теперь вы знаете, что получите три набора строк (последний из которых будет состоять всего из двух записей), и можете извлечь их:

```
SELECT * FROM profile ORDER BY name LIMIT 0, 4;
SELECT * FROM profile ORDER BY name LIMIT 4, 4;
SELECT * FROM profile ORDER BY name LIMIT 8, 4;
```

Начиная с MySQL версии 4.0 вы можете не только выбирать часть результирующего множества, но и получать информацию о том, насколько оно было бы велико, если бы не использовалась инструкция LIMIT. Например, чтобы извлечь пять первых записей таблицы profile и вычислить размер полного результирующего множества, выполните такие запросы:

```
SELECT SQL_CALC_FOUND_ROWS * FROM profile ORDER BY name LIMIT 4;
SELECT FOUND_ROWS();
```

Ключевое слово SQL_CALC_FOUND_ROWS в первом запросе сообщает MySQL, что необходимо вычислить размер всего результирующего множества, несмотря на то, что запрос возвращает только его часть. Для подсчета количества

строк вызывается функция FOUND_ROWS(). Если возвращенное значение больше четырех, значит, в результирующем множестве еще остались записи для извлечения.

3.18. Выбор соответствующих значений для инструкции LIMIT

Задача

Похоже, что инструкция LIMIT делает не то, что хотелось бы.

Решение

Убедитесь в том, что понимаете, на какой вопрос вы хотите получить ответ. Может оказаться, что LIMIT выявляет в ваших данных какие-то тонкости, о которых вы не подозревали или не принимали их в расчет.

Обсуждение

LIMIT *n* удобно использовать в сочетании с ORDER BY для выбора наименьших или наибольших значений результирующего множества. Но действительно ли выдаются *n* наибольших или наименьших значений? Необязательно! Если ваши строки содержат уникальные значения – тогда да, а вот если в строках встречаются повторения – то нет. В некоторых случаях для выбора правильного значения для LIMIT полезно выполнить предварительный запрос.

Давайте разберем один пример: множество данных, где собрана информация о подающих Американской Лиги бейсбола, которые выиграли 15 и более игр в сезоне 2001 года:

```
mysql> SELECT name, wins FROM al_winner
    -> ORDER BY wins DESC, name;
+----------------+------+
| name           | wins |
+----------------+------+
| Mulder, Mark   |   21 |
| Clemens, Roger |   20 |
| Moyer, Jamie   |   20 |
| Garcia, Freddy |   18 |
| Hudson, Tim    |   18 |
| Abbott, Paul   |   17 |
| Mays, Joe      |   17 |
| Mussina, Mike  |   17 |
| Sabathia, C.C. |   17 |
| Zito, Barry    |   17 |
| Buehrle, Mark  |   16 |
| Milton, Eric   |   15 |
| Pettitte, Andy |   15 |
| Radke, Brad    |   15 |
| Sele, Aaron    |   15 |
+----------------+------+
```

Если вы хотите узнать, кто выиграл больше всего игр, добавьте LIMIT 1 в предыдущий запрос, и вы получите правильный ответ, так как максимальное значение (21) уникально, ему соответствует только один игрок – Mark Mulder. Но что если вы захотите определить четырех лучших? Для составления запроса необходимо понять, какая именно информация вам требуется. Возможны следующие варианты:

- Если вас интересуют только первые четыре строки, упорядочите записи и добавьте в запрос LIMIT 4:

```
mysql> SELECT name, wins FROM al_winner
    -> ORDER BY wins DESC, name
    -> LIMIT 4;
+----------------+------+
| name           | wins |
+----------------+------+
| Mulder, Mark   |   21 |
| Clemens, Roger |   20 |
| Moyer, Jamie   |   20 |
| Garcia, Freddy |   18 |
+----------------+------+
```

  Результат может оказаться неудовлетворительным, так как LIMIT останавливает выборку на середине группы подающих, имеющих одинаковое количество побед (Tim Hudson тоже выиграл 18 игр).

- Чтобы не разбивать множество строк с одинаковыми значениями, выбирайте строки со значениями, равными или превышающими значение из четвертой строки. Определите, что это за значение, при помощи LIMIT, а затем используйте его в инструкции WHERE второго запроса для выбора строк:

```
mysql> SELECT wins FROM al_winner
    -> ORDER BY wins DESC, name
    -> LIMIT 3, 1;
+------+
| wins |
+------+
|   18 |
+------+
mysql> SELECT name, wins FROM al_winner
    -> WHERE wins >= 18
    -> ORDER BY wins DESC, name;
+----------------+------+
| name           | wins |
+----------------+------+
| Mulder, Mark   |   21 |
| Clemens, Roger |   20 |
| Moyer, Jamie   |   20 |
| Garcia, Freddy |   18 |
| Hudson, Tim    |   18 |
+----------------+------+
```

- Если вас интересуют все подающие, у которых количество побед входит в первую четверку, следует применить другой подход. Сначала определим четыре наибольших значения при помощи DISTINCT, а затем используем их для выбора строк:

```
mysql> SELECT DISTINCT wins FROM al_winner
    -> ORDER BY wins DESC, name
    -> LIMIT 3, 1;
+------+
| wins |
+------+
|   17 |
+------+
mysql> SELECT name, wins FROM al_winner
    -> WHERE wins >= 17
    -> ORDER BY wins DESC, name;
+----------------+------+
| name           | wins |
+----------------+------+
| Mulder, Mark   |   21 |
| Clemens, Roger |   20 |
| Moyer, Jamie   |   20 |
| Garcia, Freddy |   18 |
| Hudson, Tim    |   18 |
| Abbott, Paul   |   17 |
| Mays, Joe      |   17 |
| Mussina, Mike  |   17 |
| Sabathia, C.C. |   17 |
| Zito, Barry    |   17 |
+----------------+------+
```

Все три метода выдают разные результаты для рассматриваемого множества данных. Мораль в том, что прежде чем использовать LIMIT, стоит немного подумать о том, что именно вы хотите получить.

3.19. Получение значений LIMIT из выражений

Задача

Вы хотите указывать аргументы LIMIT, используя выражения.

Решение

Жаль, но это нельзя сделать. Вы можете использовать только целые литералы, за исключением тех случаев, когда запрос выдается в программе, тогда вы можете сами вычислить выражения, а затем подставить их в строку запроса.

Обсуждение

Аргументами LIMIT должны быть литералы, а не выражения. Предложения, подобные приведенным ниже, некорректны:

```
SELECT * FROM profile LIMIT 5+5;
SELECT * FROM profile LIMIT @skip_count, @show_count;
```

Тот же принцип «недопустимости выражений» относится и к использованию выражения для вычисления значения LIMIT в программе, формирующей строку запроса. Вы должны сначала вычислить выражение, а затем поместить результат в запрос. Например, если вы создадите в Perl (или PHP) такую строку запроса, то при попытке выполнения запроса возникнет ошибка:

```
$str = "SELECT * FROM profile LIMIT $x + $y";
```

Чтобы избежать проблем, необходимо предварительно вычислить выражение:

```
$z = $x + $y;
$str = "SELECT * FROM profile LIMIT $z";
```

Или сделайте так (только не забудьте про скобки, иначе выражение будет вычислено неправильно):

```
$str = "SELECT * FROM profile LIMIT " . ($x + $y);
```

Если вы указываете инструкцию LIMIT с двумя аргументами, то перед помещением в строку запроса необходимо вычислить оба выражения.

3.20. Что делать, если для LIMIT нужен «неправильный» порядок сортировки

Задача

Обычно LIMIT прекрасно работает в сочетании с инструкцией ORDER BY, сортирующей строки. Но иногда порядок сортировки бывает противоположным тому, который вы хотите видеть в окончательном результате.

Решение

Перепишите запрос или напишите программу, извлекающую строки и сортирующую их в требуемом порядке.

Обсуждение

Если вам нужны последние четыре записи результирующего множества, нет ничего проще, чем отсортировать его в обратном порядке и применить LIMIT 4. Например, следующий запрос возвращает имена и даты рождения из таблицы profile для четырех человек, родившихся последними:

```
mysql> SELECT name, birth FROM profile ORDER BY birth DESC LIMIT 4;
+---------+------------+
| name    | birth      |
+---------+------------+
| Shepard | 1975-09-02 |
| Carl    | 1973-11-02 |
| Fred    | 1970-04-13 |
```

```
| Mort    | 1969-09-30 |
+---------+------------+
```

Но такой способ требует сортировки по полю `birth` в порядке убывания, чтобы самые молодые оказались в начале результирующего множества. Что делать, если вы хотите выводить строки в порядке возрастания? Можно использовать два запроса. Сначала при помощи `COUNT()` вычислить количество строк в таблице:

```
mysql> SELECT COUNT(*) FROM profile;
+----------+
| COUNT(*) |
+----------+
|       10 |
+----------+
```

Затем упорядочить значения по возрастанию и использовать `LIMIT` с двумя аргументами для того, чтобы пропустить все записи, кроме последних четырех:

```
mysql> SELECT name, birth FROM profile ORDER BY birth LIMIT 6, 4;
+---------+------------+
| name    | birth      |
+---------+------------+
| Mort    | 1969-09-30 |
| Fred    | 1970-04-13 |
| Carl    | 1973-11-02 |
| Shepard | 1975-09-02 |
+---------+------------+
```

Если же запросы выдаются из программы и у вас есть возможность обрабатывать результат запроса, то задачу можно решить всего одним запросом. Например, если вы выбираете строки в структуру данных, то можете изменить порядок значений в этой структуре на обратный. Приведем фрагмент кода на Perl, представляющий такой подход:

```
my $stmt = "SELECT name, birth FROM profile ORDER BY birth DESC LIMIT 4";
# выбирать значения в структуру данных
my $ref = $dbh->selectall_arrayref ($stmt);
# изменить порядок элементов структуры на обратный
my @val = reverse (@{$ref});
# использовать $val[$i] для получения ссылки на $i строки, затем использовать
# $val[$i]->[0] и $val[$i]->[1] для доступа к значениям столбцов
```

Другой вариант – просто обходить структуру в обратном порядке:

```
my $stmt = "SELECT name, birth FROM profile ORDER BY birth DESC LIMIT 4";
# выбирать значения в структуру данных
my $ref = $dbh->selectall_arrayref ($stmt);
# обходить структуру в обратном порядке
my $row_count = @{$ref};
for (my $i = $row_count - 1; $i >= 0; $i--)
{
    # здесь использовать $ref->[$i]->[0] и $ref->[$i]->[1] ...
}
```

3.21. Выбор результирующего множества в существующую таблицу

Задача

Вы хотите выполнить запрос SELECT и не отображать его результаты, а сохранить их в другой таблице.

Решение

Если другая таблица существует, используйте предложение INSERT INTO ... SELECT, которое будет описано в этом рецепте. Если же таблица не существует, перейдите к рецепту 3.22.

Обсуждение

Обычно сервер MySQL возвращает результат предложения SELECT клиентской программе, выдавшей запрос. Например, если вы выполняете запрос из *mysql*, сервер возвращает результат в программу *mysql*, которая в свою очередь отображает его для вас на экране. Есть возможность отправить результат предложения SELECT в другую таблицу. Копирование записей одной таблицы в другую может оказаться полезным в следующих ситуациях:

- Вы разрабатываете алгоритм, изменяющий таблицу. Чтобы не беспокоиться о последствиях возможных ошибок, лучше работать с копией таблицы. К тому же, если исходная таблица большая, то создание частичной копии может ускорить процесс разработки за счет уменьшения времени выполнения обращенных к ней запросов.
- Операции загрузки данных имеют дело с информацией, которая может содержать неточности. Вы можете загрузить записи во временную таблицу, выполнить предварительную проверку и в случае необходимости исправить ошибки. Убедившись, что с записями все в порядке, скопируйте их из временной таблицы в основную.
- Некоторые приложения эксплуатируют объемную таблицу-хранилище и небольшую рабочую таблицу, в которую регулярно вставляются записи. Приложения периодически копируют записи рабочей таблицы в хранилище, после чего очищают рабочую таблицу.
- Вы выполняете ряд похожих операций суммирования для большой таблицы. Возможно будет более эффективно, если вы один раз извлечете итоговую информацию в отдельную таблицу и будете анализировать ту, вторую таблицу, чем если вам придется несколько раз выполнять дорогостоящие операции суммирования над исходной таблицей.

В разделе показано, как использовать INSERT ... SELECT для извлечения результирующего множества в существующую таблицу. Следующий раздел посвящен предложению CREATE TABLE ... SELECT, появившемуся в MySQL версии 3.23 и позволяющему «на лету» формировать таблицу непосредственно из результирующего множества. Имена таблиц src_tbl и dst_tbl, встречающиеся в примерах, относятся к исходной (source) таблице, из которой

выбираются строки, и к таблице назначения (destination), в которой строки будут храниться.

Если таблица назначения уже существует, используйте INSERT ... SELECT для копирования в нее результирующего множества. Например, если таблица dst_tbl содержит целый столбец i и строковый столбец s, то следующее предложение копирует строки src_tbl в dst_tbl, присваивая столбец val столбцу i, а столбец name – столбцу s:

```
INSERT INTO dst_tbl (i, s) SELECT val, name FROM src_tbl;
```

Количество столбцов для вставки должно совпадать с количеством столбцов выборки. Соответствие между наборами столбцов устанавливается по позициям, а не по именам. Если вы хотите скопировать все строки одной таблицы в другую, то можете использовать сокращенную форму предложения:

```
INSERT INTO dst_tbl SELECT * FROM src_tbl;
```

Для того чтобы скопировать только некоторые строки, добавьте в запрос инструкцию WHERE, которая выберет желаемые строки:

```
INSERT INTO dst_tbl SELECT * FROM src_tbl WHERE val > 100 AND name LIKE 'A%';
```

Необязательно копировать значения из одной таблицы в другую без каких бы то ни было изменений. Предложение SELECT может формировать значения из выражений. Например, следующий запрос вычисляет количество вхождений каждой фамилии в таблицу src_tbl и сохраняет в таблице и фамилии, и соответствующие счетчики:

```
INSERT INTO dst_tbl (i, s) SELECT COUNT(*), name FROM src_tbl GROUP BY name;
```

При использовании INSERT ... SELECT одна и та же таблица не может быть одновременно и исходной таблицей, и таблицей назначения.

3.22. Создание таблицы из результирующего множества «на лету»

Задача

Вы хотите выполнить запрос SELECT и сохранить результат в другой таблице, но она еще не существует.

Решение

Создайте таблицу заранее либо создавайте ее непосредственно в предложении SELECT.

Обсуждение

Если таблица назначения не существует, вы можете сначала создать ее при помощи предложения CREATE TABLE, затем скопировать в нее строки, исполь-

зуя INSERT ... SELECT (см. рецепт 3.21). Подобную процедуру можно проделать в любой версии MySQL.

Начиная с версии MySQL 3.23 появляется новая возможность – использовать предложение CREATE TABLE ... SELECT, которое создает таблицу назначения непосредственно из результирующего множества запроса SELECT. Например, чтобы создать таблицу dst_tbl и скопировать в нее все содержимое src_tbl, выполните:

```
CREATE TABLE dst_tbl SELECT * FROM src_tbl;
```

MySQL создает столбцы в dst_tbl, используя в качестве основы имена, количество и типы столбцов src_tbl. Если вы хотите скопировать не все, а только некоторые строки, добавьте соответствующую инструкцию WHERE. Если вы хотите создать пустую таблицу, укажите в инструкции WHERE заведомо ложное условие:

```
CREATE TABLE dst_tbl SELECT * FROM src_tbl WHERE 0;
```

Для того чтобы скопировать только некоторые столбцы, укажите их имена в части SELECT предложения. Например, если таблица src_tbl содержит столбцы a, b, c и d, вы можете скопировать только b и d так:

```
CREATE TABLE dst_tbl SELECT b, d FROM src_tbl;
```

Чтобы создать столбцы не в том порядке, в котором они присутствуют в исходной таблице, просто укажите их имена в нужном порядке. Если исходная таблица содержит столбцы a, b и c, и вы хотите, чтобы в таблице назначения они появились в порядке c, a, b, выполните:

```
CREATE TABLE dst_tbl SELECT c, a, b FROM src_tbl;
```

Чтобы создать в таблице назначения дополнительные (по отношению к столбцам исходной таблицы) столбцы, определите эти столбцы в части CREATE TABLE предложения. Создадим таблицу dst_tbl, в которую скопируем столбцы a, b и c таблицы src_tbl, а также создадим и добавим новый столбец id типа AUTO_INCREMENT:

```
CREATE TABLE dst_tbl
(
    id INT NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (id)
)
SELECT a, b, c FROM src_tbl;
```

Результирующая таблица содержит четыре столбца в таком порядке: id, a, b, c. Столбцам, которые были определены, присваиваются значения по умолчанию, то есть столбцу id, определенному как AUTO_INCREMENT, будут последовательно присвоены порядковые номера, начиная с единицы (см. рецепт 11.1).

Если вы получаете значения столбца из выражения, то в целях предосторожности стоит использовать для столбца псевдоним. Предположим, что таблица src_tbl содержит такую информацию о счетах, как перечень товаров, указанных в каждом счете. Тогда приведенное ниже предложение формирует сводку счетов, входящих в таблицу, и для каждого из них выводит его общую сто-

имость. Для второго столбца использован псевдоним, так как именем по умолчанию для выражения является само выражение, а с ним неудобно работать:

```
CREATE TABLE dst_tbl
SELECT inv_no, SUM(unit_cost*quantity) AS total_cost
FROM src_tbl
GROUP BY inv_no;
```

На самом деле в более ранних, чем MySQL 3.23.6, версиях псевдоним не просто желателен, а необходим; правила, действующие в таких версиях для имен столбцов, более строгие и не допускают использования выражения для имени столбца таблицы.

Предложение `CREATE TABLE ... SELECT` чрезвычайно полезно, но имеет некоторые ограничения. В первую очередь это объясняется тем, что информация, которую можно получить из результирующего множества, не так обширна, как указываемая в предложении `CREATE TABLE`. Например, если вы получаете столбец из выражения, MySQL ничего не знает о том, следует ли индексировать этот столбец или каково его значение по умолчанию. Если вам необходимо, чтобы такая информация попала в таблицу назначения, вы можете сделать следующее:

- Если вы хотите индексировать создаваемую таблицу, укажите индексы явно. Например, если `src_tbl` имеет первичный ключ (`PRIMARY KEY`) – столбец `id`, и составной индекс (multiple-column index), включающий столбцы `state` и `city`, то вы можете указать их и для таблицы `dst_tbl`:

  ```
  CREATE TABLE dst_tbl (PRIMARY KEY (id), INDEX(state,city))
  SELECT * FROM src_tbl;
  ```

- Атрибуты столбцов, такие как `AUTO_INCREMENT` и значение столбца по умолчанию, не копируются в таблицу назначения. Чтобы сохранить их, создайте таблицу, затем соответствующим образом измените определение столбца. Например, если таблица `src_tbl` содержит столбец `id`, который является не только первичным ключом, но еще и столбцом типа `AUTO_INCREMENT`, вы можете скопировать таблицу, а затем изменить ее:

  ```
  CREATE TABLE dst_tbl (PRIMARY KEY (id)) SELECT * FROM src_tbl;
  ALTER TABLE dst_tbl MODIFY id INT UNSIGNED NOT NULL AUTO_INCREMENT;
  ```

- Если вы хотите, чтобы таблица назначения была полной копией исходной таблицы, то можете использовать технику клонирования, описанную в рецепте 3.25.

3.23. Безопасное перемещение записей из таблицы в таблицу

Задача

Вы перемещаете записи, копируя их из одной таблицы в другую, а затем удаляя их из исходной таблицы. Но похоже, что некоторые записи при этом теряются.

Решение

Будьте внимательны и удаляйте из исходной таблицы именно те записи, которые были скопированы в таблицу назначения.

Обсуждение

Приложения, копирующие строки из одной таблицы в другую, могут делать это в одной операции, такой как `INSERT ... SELECT`, извлекая соответствующие строки из исходной таблицы и вставляя их в таблицу назначения. Если же необходимо именно *переместить* (а не скопировать) строки, процедура несколько усложняется. После копирования строк вы должны удалить их из исходной таблицы. В теории кажется, что после `INSERT ... SELECT` следует просто добавить `DELETE`. На практике же приходится внимательно следить за тем, чтобы выбрать одни и те же наборы строк для предложений `INSERT` и `DELETE`. Если другие клиентские приложения вставят в исходную таблицу новые строки после того, как вы выполните `INSERT`, но до `DELETE`, может получиться не очень хорошо.

Предположим, например, что у вас есть приложение, которое использует рабочую таблицу регистрации `worklog`, в которую постоянно вносятся записи, и долгосрочное хранилище – таблицу регистрации `repolog`. Периодически вы перемещаете записи `worklog` в `repolog`, чтобы сохранить объем рабочей таблицы небольшим и чтобы клиенты могли создавать сложные аналитические запросы к журналу в хранилище, не блокируя при этом процессы создания новых записей в рабочей таблице регистрации.[1]

Как в данной ситуации корректно перенести записи из `worklog` в `repolog`, зная, что `worklog` постоянно подвергается изменениям? Очевидный (но неправильный) ответ заключается в выполнении запроса `INSERT ... SELECT` для копирования всех записей `worklog` в `repolog` с последующим выполнением предложения `DELETE` для удаления их из `worklog`:

```
INSERT INTO repolog SELECT * FROM worklog;
DELETE FROM worklog;
```

Так можно поступать, только если вы абсолютно уверены в том, что никто другой не будет вставлять записи в `worklog` в течение периода времени, разделяющего выполнение двух предложений. Если же другие клиентские предложения будут в это время вставлять новые записи, они будут сразу же удалены (до их копирования), и вы их потеряете. Если таблицы хранят протоколы запросов веб-страниц, то потеря может быть не очень существенной, но если речь идет о финансовых операциях, то вы столкнетесь с серьезной проблемой.

---

[1] Если вы используете таблицу регистрации типа MyISAM, в которую только вставляете записи, но никогда не обновляете ее и не удаляете из нее записи, то можете выполнять запросы для этой таблицы, не запрещая другим пользователям вставлять новые записи в конец таблицы.

Как можно предотвратить потерю записей? Можно выполнять два предложения в одной транзакции или блокировать таблицы на время вашей работы с ними. Об этих приемах рассказано в главе 15. Однако оба эти способа надолго блокируют работу других приложений, замораживая доступ к таблицам на время выполнения обоих запросов. В качестве альтернативы можно предложить перемещение только тех записей, дата которых меньше какого-то граничного значения. Например, если журнал содержит столбец `t` с временной меткой (timestamp), то вы можете ограничить объем выбираемых записей записями, созданными раньше вчерашнего дня. Тогда добавление новых записей в `worklog` в промежутке между операциями копирования и удаления ничего не изменит. Но выбирайте граничное значение внимательно. При некоторых обстоятельствах и этот метод может не сработать:

```
INSERT INTO repolog SELECT * FROM worklog WHERE t < CURDATE();
DELETE FROM worklog WHERE t < CURDATE();
```

Если случится так, что предложение `INSERT` будет выполнено за секунду до полуночи, а `SELECT` – секунду спустя, ничего не получится. Значение `CURDATE()` будет разным для двух предложений, и `DELETE` может удалить слишком много записей. Если вы собираетесь использовать предельное значение, убедитесь, что оно является фиксированным и не меняет значения от предложения к предложению. Например, можно использовать переменную SQL для хранения значения `CURDATE()` в такой форме, которая не будет меняться с течением времени:

```
SET @cutoff = CURDATE();
INSERT INTO repolog SELECT * FROM worklog WHERE t < @cutoff;
DELETE FROM worklog WHERE t < @cutoff;
```

Теперь оба предложения используют одно и то же граничное значение времени, так что `DELETE` не удалит ничего лишнего.

3.24. Создание временных таблиц

Задача

Таблица нужна вам только для временного использования, а затем вы хотели бы, чтобы она автоматически исчезла.

Решение

Создайте таблицу `TEMPORARY` и предоставьте MySQL заниматься ее уничтожением.

Обсуждение

Для некоторых операций необходима таблица, которая бы существовала только в течение короткого периода времени и исчезала, когда больше не нужна. Конечно, можно по окончании работы явно удалить таблицу при помощи `DROP TABLE`. Начиная с версии MySQL 3.23.2 есть и другой вариант –

использовать предложение CREATE TEMPORARY TABLE. Это предложение аналогично CREATE TABLE, за тем лишь исключением, что оно создает временную таблицу, которая исчезает при закрытии соединения с сервером (если вы раньше не удалили ее сами). Такая возможность очень удобна, так как вам не приходится помнить о том, что необходимо удалить таблицу, MySQL автоматически удаляет ее за вас.

Временные таблицы создаются в рамках соединения, поэтому несколько клиентских программ могут создать временную таблицу с одним и тем же именем, при этом никакого наложения не произойдет. То есть при создании приложений, использующих временные таблицы, не приходится обеспечивать уникальность имен таблиц для каждого клиента (обсуждение этого вопроса продолжается в рецепте 3.26).

Еще одним свойством временных таблиц является возможность присвоения им того же имени, что и у постоянной таблицы. В этом случае временная таблица «скрывает» постоянную на время своего существования, что удобно для создания копии таблицы, которую можно обновлять, не боясь по ошибке повредить оригинал. В следующем наборе запросов предложение DELETE выбирает записи из временной таблицы mail, при этом исходная постоянная таблица не изменяется:

```
mysql> CREATE TEMPORARY TABLE mail SELECT * FROM mail;
mysql> SELECT COUNT(*) FROM mail;
+----------+
| COUNT(*) |
+----------+
|       16 |
+----------+
mysql> DELETE FROM mail;
mysql> SELECT COUNT(*) FROM mail;
+----------+
| COUNT(*) |
+----------+
|        0 |
+----------+
mysql> DROP TABLE mail;
mysql> SELECT COUNT(*) FROM mail;
+----------+
| COUNT(*) |
+----------+
|       16 |
+----------+
```

Достоинства временных таблиц, созданных при помощи CREATE TEMPORARY TABLE, вам уже известны. Упомянем несколько тонкостей их использования:

- Если вы хотите повторно использовать временную таблицу в рамках одного сеанса, вам придется явно удалить ее перед пересозданием. Явного удаления не требует только *последняя* из использованных таблиц с одним именем (то есть если вы уже создали временную таблицу с некоторым именем, то попытка создать вторую с тем же именем приведет к ошибке).

- Некоторые API поддерживают постоянное соединение в веб-окружении. В этом случае временные таблицы не удаляются при завершении работы сценариев (как хотелось бы), так как веб-сервер сохраняет соединение открытым для других сценариев. Сервер может и закрыть соединение, но вам никак не удастся проконтролировать этот процесс. Поэтому в подобных ситуациях следует перед созданием временной таблицы сформировать следующее предложение на тот случай, если она осталась от предыдущего выполнения данного сценария:

  ```
  DROP TABLE IF EXISTS имя_таблицы
  ```

- Если вы изменяете временную таблицу, которая «скрывает» постоянную с тем же именем, не забудьте об ошибках, которые могут возникнуть в результате прерванного соединения. Если клиентская программа автоматически восстанавливает соединения после разрыва, то после повторного установления соединения вы будете работать с исходной (постоянной) таблицей.

3.25. Клонирование таблицы

Задача

Вам нужна точная копия таблицы, а предложение `CREATE TABLE ... SELECT` не отвечает вашим целям, так как копия должна содержать те же индексы, значения по умолчанию и т. д., что и исходная таблица.

Решение

Используйте предложение `SHOW CREATE TABLE` для получения предложения `CREATE TABLE`, указывающего структуру исходной таблицы, индексы и все остальное. Затем преобразуйте предложение, заменив имя таблицы на имя клона, и выполните предложение. Если вам необходимо скопировать и содержимое таблицы, выполните также предложение `INSERT INTO ... SELECT`.

Обсуждение

Поскольку предложение `CREATE TABLE ... SELECT` не копирует ни индексы, ни полный набор атрибутов столбцов, абсолютно необязательно таблица назначения будет полной копией исходной таблицы. Поэтому полезным может оказаться запрос `SHOW CREATE TABLE`, обращенный к исходной таблице. Это предложение появилось в MySQL версии 3.23.20; оно возвращает строку, содержащую имя таблицы и предложение `CREATE TABLE`, соответствующее структуре данной таблицы (включая индексы, атрибуты столбцов и тип таблицы):

```
mysql> SHOW CREATE TABLE mail\G
*************************** 1. row ***************************
       Table: mail
Create Table: CREATE TABLE `mail` (
  `t` datetime default NULL,
  `srcuser` char(8) default NULL,
```

```
  `srchost` char(20) default NULL,
  `dstuser` char(8) default NULL,
  `dsthost` char(20) default NULL,
  `size` bigint(20) default NULL,
  KEY `t` (`t`)
) TYPE=MyISAM
```

Создав в программе предложение SHOW CREATE TABLE и выполнив замену строки для изменения имени таблицы, вы получаете предложение, которое может быть использовано для создания новой таблицы с той же структурой, что и исходная. Напишем на Python функцию, принимающую три аргумента (объект соединения и имена исходной таблицы и таблицы назначения). Она извлекает предложение CREATE TABLE для исходной таблицы, подставляет в него имя таблицы назначения и возвращает результат:

```
# Сформировать предложение CREATE TABLE для создания таблицы dst_tbl
# с той же структурой, что и существующая таблица src_tbl.
# Вернуть None в случае ошибки. Необходим модуль re.

def gen_clone_query (conn, src_tbl, dst_tbl):
    try:
        cursor = conn.cursor ()
        cursor.execute ("SHOW CREATE TABLE " + src_tbl)
        row = cursor.fetchone ()
        cursor.close ()
        if row == None:
            query = None
        else:
            # Заменить src_tbl на dst_tbl в предложении CREATE TABLE
            query = re.sub ("CREATE TABLE .*`" + src_tbl + "`",
                            "CREATE TABLE `" + dst_tbl + "`",
                            row[1])
    except:
        query = None
    return query
```

Вы можете использовать полученное предложение для создания новой таблицы в том виде, в котором оно получено:

```
query = gen_clone_query (conn, old_tbl, new_tbl)
cursor = conn.cursor ()
cursor.execute (query)
cursor.close ()
```

А можете подойти к делу более творчески. Например, создав не постоянную, а временную таблицу или перед выполнением предложения заменив CREATE на CREATE TEMPORARY:

```
query = gen_clone_query (conn, old_tbl, new_tbl)
query = re.sub ("CREATE ", "CREATE TEMPORARY ", query)
cursor = conn.cursor ()
cursor.execute (query)
cursor.close ()
```

Выполнение предложения, возвращенного функцией gen_clone_query(), приводит к созданию пустой копии исходной таблицы. Чтобы скопировать и содержимое таблицы, после того как копия создана, сделайте нечто подобное:

```
cursor = conn.cursor ()
cursor.execute ("INSERT INTO " + new_tbl + " SELECT * FROM " + old_tbl)
cursor.close ()
```

До MySQL версии 3.23.50 существовало несколько атрибутов, которые можно было задавать в CREATE TABLE, но в SHOW CREATE TABLE они не отображались. Если ваша исходная таблица создана с одним из таких атрибутов, то приведенная техника клонирования обеспечит создание таблицы, структура которой будет несколько отличаться от структуры исходной таблицы.

3.26. Формирование уникальных имен таблиц

Задача

Вам нужно создать таблицу с именем, которого гарантированно еще не существует.

Решение

Если вы можете создать таблицу TEMPORARY, то не имеет значения, существовало ли такое имя ранее. В противном случае попытайтесь сформировать значение, которое однозначно определяется клиентской программой, и встройте его в имя таблицы.

Обсуждение

MySQL – это многопользовательский сервер базы данных, поэтому если сценарий, создающий временную таблицу, может вызываться одновременно несколькими клиентскими программами, вы должны позаботиться о том, чтобы несколько вызовов сценария не боролись за одно и то же имя таблицы. Если сценарий создает таблицы с помощью предложения CREATE TEMPORARY TABLE, этой проблемы нет, так как разные клиенты могут бесконфликтно создавать временные таблицы с одинаковыми именами.

Если же вы работаете с сервером более ранней, чем 3.23.2, версии и не можете использовать CREATE TEMPORARY TABLE, то должны сделать так, чтобы каждый вызов сценария создавал таблицу с уникальным именем. Для этого встройте в имя таблицы некоторое значение, которое гарантированно является уникальным для данного вызова. Временная метка не подходит, потому что два экземпляра сценария вполне могут быть вызваны в одну и ту же секунду. Случайные числа – это уже лучше. Например, в Java вы можете использовать для формирования имени таблицы класс java.util.Random:

```
import java.util.Random;
import java.lang.Math;
```

```
Random rand = new Random ();
int n = rand.nextInt ();            // получить случайное число
n = Math.abs (n);                   // взять его абсолютное значение
String tblName = "tmp_tbl_" + n;
```

К сожалению, случайные числа только уменьшают вероятность конфликтов имен, но не устраняют ее. Более удачным источником уникальных значений служат идентификаторы процесса (PID – Process ID). PID используются повторно через некоторое время, но никогда – для двух процессов, работающих одновременно, поэтому каждый PID гарантированно уникален в множестве процессов, выполняемых в текущий момент времени. Этот факт можно применить для создания уникальных имен таблиц:

Perl:

```
my $tbl_name = "tmp_tbl_$$";
```

PHP:

```
$tbl_name = "tmp_tbl_" . posix_getpid ();
```

Python:

```
import os
tbl_name = "tmp_tbl_%d" % os.getpid ()
```

Даже если вы хотите создать имя таблицы, используя такое значение, как PID, заведомо уникальное для данного вызова сценария, все еще остается вероятность того, что таблица уже существует. Так может случиться, если предыдущий вызов сценария с тем же PID создал таблицу с тем же именем и аварийно завершился, не успев удалить ее. С другой стороны, любая такая таблица уже не может использоваться, так как она была создана процессом, который уже завершен. Учитывая вышесказанное, надежнее удалять существовавшую таблицу (если она была), используя такое предложение:

```
DROP TABLE IF EXISTS имя_таблицы
```

После этого можно приступать к созданию новой таблицы.

4

Работа со строками

4.0. Введение

Как и многие другие типы данных, строки (string) можно сравнивать на равенство или неравенство, а также на взаимный порядок. Но в работе со строками есть и особенности:

- Строки могут быть или не быть чувствительными к смене регистра, что может влиять на результат выполнения строковых операций.
- Вы можете сравнивать как целые строки, так и их части, извлекая подстроки.
- Для поиска строк, имеющих определенную структуру, вы можете выполнять операции поиска по образцу.

В этой главе будет рассмотрен ряд полезных строковых операций, в том числе будет рассказано о том, как учитывать возможность чувствительности строк к смене регистра.

Во многих разделах главы используется таблица `metal`:

```
mysql> SELECT * FROM metal;
+----------+
| name     |
+----------+
| copper   |
| gold     |
| iron     |
| lead     |
| mercury  |
| platinum |
| silver   |
| tin      |
+----------+
```

Таблица очень проста и содержит всего один строковый столбец:

```
CREATE TABLE metal
(
```

```
    name    VARCHAR(20)
);
```

Можно создать таблицу при помощи сценария *metal.sql* из каталога *tables* дистрибутива `recipes`.

Типы строк

MySQL может работать с обычными (regular) или двоичными (binary) строками. В данном случае понятие «двоичный» не связано с присутствием не-ASCII значений, так что сразу внесем ясность:

- Двоичные *данные* (data) могут содержать байты, выходящие за пределы обычного диапазона печатаемых знаков ASCII.
- Двоичная *строка* (string) MySQL – это строка, которую MySQL воспринимает в операциях сравнения как чувствительную к смене регистра. Для двоичных строк символы `A` и `a` считаются различными, а в обычных строках эти два символа рассматриваются как одинаковые.

К столбцам двоичного типа относятся столбцы, содержащие двоичные строки. Некоторые типы столбцов MySQL являются двоичными (чувствительными к регистру), а некоторые – нет (табл. 4.1):

Таблица 4.1. Типы столбцов и их чувствительность к регистру

Тип столбца	Двоичный (чувствительный к регистру)
`CHAR`, `VARCHAR`	Нет
`CHAR BINARY`, `VARCHAR BINARY`	Да
`TEXT`	Нет
`BLOB`	Да
`ENUM`, `SET`	Нет

4.1. Создание строк, содержащих кавычки или другие специальные символы

Задача

Вы хотите создать строку, заключенную в кавычки, но оказывается, что она содержит кавычки или другие специальные символы, и MySQL ее отвергает.

Решение

Изучите правила синтаксиса, регулирующие обработку строк в запросах.

Обсуждение

Чтобы вставить строку в предложение SQL, заключите ее в кавычки:

```
mysql> SELECT 'hello, world';
+--------------+
| hello, world |
+--------------+
| hello, world |
+--------------+
```

Но бывает так, что сама строка содержит кавычки, тогда, если заключить ее в кавычки «как есть», будет выдана синтаксическая ошибка:

```
mysql> SELECT 'I'm asleep';
ERROR 1064 at line 1: You have an error in your SQL syntax near 'asleep''
at line 1
```

Исправить положение можно несколькими способами:

- В отличие от некоторых других процессоров SQL, MySQL позволяет использовать как одинарные, так и двойные кавычки, так что вы можете заключить строку, содержащую одинарные кавычки, в двойные:

```
mysql> SELECT "I'm asleep";
+------------+
| I'm asleep |
+------------+
| I'm asleep |
+------------+
```

Можно сделать и наоборот: заключить строку, содержащую двойные кавычки, в одинарные:

```
mysql> SELECT 'He said, "Boo!"';
+-----------------+
| He said, "Boo!" |
+-----------------+
| He said, "Boo!" |
+-----------------+
```

- Для того чтобы включить символ кавычки в строку, заключенную в кавычки того же типа, следует или продублировать кавычку, или поставить перед ней символ обратного слэша (\). MySQL, прочитав строку запроса, уберет дополнительную кавычку или обратный слэш:

```
mysql> SELECT 'I''m asleep', 'I\'m wide awake';
+------------+----------------+
| I'm asleep | I'm wide awake |
+------------+----------------+
| I'm asleep | I'm wide awake |
+------------+----------------+
1 row in set (0.00 sec)
mysql> SELECT "He said, ""Boo!""", "And I said, \"Yikes!\"";
+-----------------+----------------------+
| He said, "Boo!" | And I said, "Yikes!" |
+-----------------+----------------------+
| He said, "Boo!" | And I said, "Yikes!" |
+-----------------+----------------------+
```

Обратный слэш отключает специальную интерпретацию следующего за ним символа. (Происходит как бы временный уход от обычных правил обработки строк, поэтому такие последовательности, как \' и \" называют escape-последовательностями.) Соответственно сам обратный слэш тоже является специальным символом, и чтобы буквально использовать его внутри строки, вы должны продублировать его:

```
mysql> SELECT 'Install MySQL in C:\\mysql on Windows';
+-------------------------------------+
| Install MySQL in C:\mysql on Windows |
+-------------------------------------+
| Install MySQL in C:\mysql on Windows |
+-------------------------------------+
```

MySQL также распознает такие управляющие последовательности, как \b (backspace, забой), \n (перевод строки, или новая строка), \r (возврат каретки), \t (табуляция) и \0 (ASCII-ноль, NUL).

См. также

Использование управляющих последовательностей при написании строк лучше ограничить текстовыми значениями. Если вы хотите включить в строку такие значения, как картинки, состоящие из произвольных данных, то каждый их специальный символ тоже должен быть экранирован. Но о попытке подобного ввода изображения страшно даже подумать. Такие запросы должны создаваться в программе, где можно использовать механизм заполнителей, реализованный в API выбранного языка (см. рецепт 2.6).

4.2. Сохранение замыкающих пробелов в строковых столбцах

Задача

MySQL удаляет из строк замыкающие пробелы, а вы хотели бы их сохранить.

Решение

Используйте другой тип столбца.

Обсуждение

Если вы сохраняете в базе данных строковое значение, содержащее замыкающие пробелы, то при извлечении значения можете обнаружить, что они исчезли. Обычно MySQL именно так ведет себя по отношению к столбцам CHAR и VARCHAR; сервер возвращает из столбцов двух этих типов значения без замыкающих пробелов. Если вам нужно сохранить замыкающие пробелы, используйте столбцы типа TEXT или BLOB (тип TEXT не чувствителен к регистру, а BLOB чувствителен). Рассмотрим различия в поведении столбцов VARCHAR и TEXT на примере:

```
mysql> CREATE TABLE t (c VARCHAR(255));
mysql> INSERT INTO t (c) VALUES('abc       ');
mysql> SELECT c, LENGTH(c) FROM t;
+------+-----------+
| c    | LENGTH(c) |
+------+-----------+
| abc  |         3 |
+------+-----------+
mysql> DROP TABLE t;
mysql> CREATE TABLE t (c TEXT);
mysql> INSERT INTO t (c) VALUES('abc       ');
mysql> SELECT c, LENGTH(c) FROM t;
+------------+-----------+
| c          | LENGTH(c) |
+------------+-----------+
| abc        |        10 |
+------------+-----------+
```

В следующей версии MySQL планируется ввод типа VARCHAR, сохраняющего замыкающие пробелы.

4.3. Проверка равенства и взаимного порядка строк

Задача

Вы хотите проверить строки на равенство или узнать, какая из них является первой лексически.

Решение

Используйте оператор сравнения.

Обсуждение

К строкам можно применять обычные операторы проверки на равенство и неравенство:

```
mysql> SELECT name, name = 'lead', name != 'lead' FROM metal;
+----------+---------------+----------------+
| name     | name = 'lead' | name != 'lead' |
+----------+---------------+----------------+
| copper   |             0 |              1 |
| gold     |             0 |              1 |
| iron     |             0 |              1 |
| lead     |             1 |              0 |
| mercury  |             0 |              1 |
| platinum |             0 |              1 |
| silver   |             0 |              1 |
| tin      |             0 |              1 |
+----------+---------------+----------------+
```

Также можно использовать операторы сравнения, такие как <, <=, >= и >, для проверки лексического порядка строк:

```
mysql> SELECT name, name < 'lead', name > 'lead' FROM metal;
+----------+---------------+---------------+
| name     | name < 'lead' | name > 'lead' |
+----------+---------------+---------------+
| copper   |             1 |             0 |
| gold     |             1 |             0 |
| iron     |             1 |             0 |
| lead     |             0 |             0 |
| mercury  |             0 |             1 |
| platinum |             0 |             1 |
| silver   |             0 |             1 |
| tin      |             0 |             1 |
+----------+---------------+---------------+
```

Для того чтобы определить, попадает ли строка в определенный диапазон значений, можно использовать два сравнения:

```
mysql> SELECT name, 'iron' <= name AND name <= 'platinum' FROM metal;
+----------+---------------------------------------+
| name     | 'iron' <= name AND name <= 'platinum' |
+----------+---------------------------------------+
| copper   |                                     0 |
| gold     |                                     0 |
| iron     |                                     1 |
| lead     |                                     1 |
| mercury  |                                     1 |
| platinum |                                     1 |
| silver   |                                     0 |
| tin      |                                     0 |
+----------+---------------------------------------+
```

Для проверки вхождения в диапазон можно использовать и оператор BETWEEN. Следующий запрос аналогичен предыдущему:

```
SELECT name, name BETWEEN 'iron' AND 'platinum' FROM metal;
```

См. также

Результат сравнения строк может зависеть от того, являются ли операнды двоичными строками (см. рецепт 4.9).

4.4. Разбиение и объединение строк

Задача

Вы хотите разбить строку на части, чтобы извлечь подстроку, или объединить строки для формирования одной большой строки.

Решение

Чтобы получить часть строки, используйте функцию извлечения подстрок. Для объединения строк используйте CONCAT().

Обсуждение

Можно извлекать и выводить на экран части строк. Например, функции LEFT(), MID() и RIGHT() извлекают подстроки с левого конца строки, из середины или с правого конца:

```
mysql> SELECT name, LEFT(name,2), MID(name,3,1), RIGHT(name,3) FROM metal;
+----------+--------------+---------------+---------------+
| name     | LEFT(name,2) | MID(name,3,1) | RIGHT(name,3) |
+----------+--------------+---------------+---------------+
| copper   | co           | p             | per           |
| gold     | go           | l             | old           |
| iron     | ir           | o             | ron           |
| lead     | le           | a             | ead           |
| mercury  | me           | r             | ury           |
| platinum | pl           | a             | num           |
| silver   | si           | l             | ver           |
| tin      | ti           | n             | tin           |
+----------+--------------+---------------+---------------+
```

Второй аргумент функций LEFT() и RIGHT() указывает, сколько символов следует вернуть, начиная с левого или правого конца строки. Второй аргумент функции MID() – это та позиция, с которой начинается интересующая вас подстрока (нумерация начинается с 1), а третий аргумент показывает, сколько символов должно быть возвращено.

Функция SUBSTRING() принимает в качестве аргументов строку и точку отсчета, а возвращает все, что находится справа от заданной позиции:[1]

```
mysql> SELECT name, SUBSTRING(name,4), MID(name,4) FROM metal;
+----------+-------------------+-------------+
| name     | SUBSTRING(name,4) | MID(name,4) |
+----------+-------------------+-------------+
| copper   | per               | per         |
| gold     | d                 | d           |
| iron     | n                 | n           |
| lead     | d                 | d           |
| mercury  | cury              | cury        |
| platinum | tinum             | tinum       |
| silver   | ver               | ver         |
| tin      |                   |             |
+----------+-------------------+-------------+
```

[1] Если не указать третий аргумент функции MID(), она работает точно так же. Фактически MID() – это синоним SUBSTRING().

Чтобы вывести все, что расположено слева или справа от заданного символа, используйте функцию SUBSTRING_INDEX(*str*,*c*,*n*). Она ищет в строке *str* *n*-е вхождение символа *c* и возвращает все, что находится слева от него. Если число *n* отрицательное, то поиск символа *c* начинается справа, и возвращается все, что находится справа от найденного символа:

```
mysql> SELECT name,
    -> SUBSTRING_INDEX(name,'r',2),
    -> SUBSTRING_INDEX(name,'i',-1)
    -> FROM metal;
+----------+----------------------------+-----------------------------+
| name     | SUBSTRING_INDEX(name,'r',2) | SUBSTRING_INDEX(name,'i',-1) |
+----------+----------------------------+-----------------------------+
| copper   | copper                     | copper                      |
| gold     | gold                       | gold                        |
| iron     | iron                       | ron                         |
| lead     | lead                       | lead                        |
| mercury  | mercu                      | mercury                     |
| platinum | platinum                   | num                         |
| silver   | silver                     | lver                        |
| tin      | tin                        | n                           |
+----------+----------------------------+-----------------------------+
```

Заметим, что если *n*-е вхождение символа *c* не найдено, то возвращается вся строка. Функция SUBSTRING_INDEX() чувствительна к регистру.

Подстроки можно не только выводить, но и использовать их в операциях сравнения. Следующий запрос ищет названия металлов, начинающиеся с букв второй половины алфавита:

```
mysql> SELECT name from metal WHERE LEFT(name,1) >= 'n';
+----------+
| name     |
+----------+
| platinum |
| silver   |
| tin      |
+----------+
```

Если вас интересует не разбиение, а соединение строк, используйте функцию CONCAT(). Она соединяет все свои аргументы и возвращает результат:

```
mysql> SELECT CONCAT('Hello, ',USER(),', welcome to MySQL!') AS greeting;
+-----------------------------------------+
| greeting                                |
+-----------------------------------------+
| Hello, paul@localhost, welcome to MySQL! |
+-----------------------------------------+
mysql> SELECT CONCAT(name,' ends in "d": ',IF(RIGHT(name,1)='d','YES','NO'))
    -> AS 'ends in "d"?'
    -> FROM metal;
+-------------------------+
| ends in "d"?            |
```

```
+--------------------------+
| copper ends in "d": NO   |
| gold ends in "d": YES    |
| iron ends in "d": NO     |
| lead ends in "d": YES    |
| mercury ends in "d": NO  |
| platinum ends in "d": NO |
| silver ends in "d": NO   |
| tin ends in "d": NO      |
+--------------------------+
```

Соединять строки удобно для изменения значений столбцов прямо «на месте». Например, такое предложение `UPDATE` добавляет строку в конец каждого значения `name` таблицы `metal`:

```
mysql> UPDATE metal SET name = CONCAT(name,'ide');
mysql> SELECT name FROM metal;
+-------------+
| name        |
+-------------+
| copperide   |
| goldide     |
| ironide     |
| leadide     |
| mercuryide  |
| platinumide |
| silveride   |
| tinide      |
+-------------+
```

Чтобы отменить эту операцию, удалите три последних символа (функция `LENGTH()` возвращает длину строки):

```
mysql> UPDATE metal SET name = LEFT(name,LENGTH(name)-3);
mysql> SELECT name FROM metal;
+----------+
| name     |
+----------+
| copper   |
| gold     |
| iron     |
| lead     |
| mercury  |
| platinum |
| silver   |
| tin      |
+----------+
```

Изменение столбца прямо на месте применимо и к значениям типов `ENUM` и `SET`, которые обычно могут интерпретироваться как строки, несмотря на то, что внутренне хранятся как числа. Например, чтобы соединить элемент `SET` с существующим столбцом `SET`, используйте функцию `CONCAT()` для добавления

нового значения к существующему через запятую. Не забудьте и о том, что существующее значение может оказаться пустой строкой или NULL, тогда присвойте столбцу новое значение без начальной запятой:

```
UPDATE имя_таблицы
SET столбец_set = IF(столбец_set IS NULL OR столбец_set = '',val,CONCAT(столбец_set,',',val));
```

4.5. Проверка вхождения подстроки в строку

Задача

Вы хотите узнать, встречается ли указанная строка в другой строке.

Решение

Используйте функцию LOCATE().

Обсуждение

Функция LOCATE() принимает два аргумента – искомую подстроку и строку, в которой вы ее ищете. Возвращаемое значение – это позиция, в которой найдена подстрока, или 0, если такой подстроки нет. Можно указать необязательный третий аргумент – позицию, с которой следует начинать поиск внутри строки:

```
mysql> SELECT name, LOCATE('in',name), LOCATE('in',name,3) FROM metal;
+----------+-------------------+--------------------+
| name     | LOCATE('in',name) | LOCATE('in',name,3) |
+----------+-------------------+--------------------+
| copper   |                 0 |                  0 |
| gold     |                 0 |                  0 |
| iron     |                 0 |                  0 |
| lead     |                 0 |                  0 |
| mercury  |                 0 |                  0 |
| platinum |                 5 |                  5 |
| silver   |                 0 |                  0 |
| tin      |                 2 |                  0 |
+----------+-------------------+--------------------+
```

Функция LOCATE() не чувствительна к регистру начиная с MySQL 4.0.0, но была чувствительна к нему в более ранних версиях.

4.6. Поиск по образцу с помощью шаблонов SQL

Задача

Вы хотите выполнить поиск по образцу, а не буквальное сравнение.

Решение

Используйте оператор LIKE и шаблон SQL, описанный в данном разделе. Или воспользуйтесь регулярными выражениями, описанными в рецепте 4.7.

Обсуждение

Шаблоны – это строки, содержащие специальные символы. Специальные символы также называют метасимволами, так как они означают нечто отличное от самих себя. MySQL поддерживает два вида поиска по образцу. В одном из них используются шаблоны SQL, а в другом – регулярные выражения. Шаблоны SQL универсальнее при переходе от одной СУБД к другой, но регулярные выражения являются более мощным средством. Эти два вида поиска по образцу используют различные операторы и различные наборы метасимволов. В данном разделе мы поговорим о шаблонах SQL, а регулярным выражениям посвящен рецепт 4.7.

При поиске по шаблону SQL для сравнения строк с образцом используются операторы LIKE и NOT LIKE вместо = и !=. Шаблон может содержать два специальных метасимвола: _ (подчеркивание) соответствует любому отдельному символу, а % (знак процента) – любой последовательности символов, включая пустую строку. Вы можете использовать эти символы для создания разнообразных шаблонов:

- Строки, начинающиеся с определенной подстроки:

```
mysql> SELECT name FROM metal WHERE name LIKE 'co%';
+--------+
| name   |
+--------+
| copper |
+--------+
```

- Строки, заканчивающиеся определенной подстрокой:

```
mysql> SELECT name FROM metal WHERE name LIKE '%er';
+--------+
| name   |
+--------+
| copper |
| silver |
+--------+
```

- Строки, содержащие (в любом месте) определенную подстроку:

```
mysql> SELECT name FROM metal WHERE name LIKE '%er%';
+---------+
| name    |
+---------+
| copper  |
| mercury |
| silver  |
+---------+
```

- Строки, содержащие определенную подстроку, которая начинается с указанной позиции (с шаблоном совпадут только те строки, в которых pp присутствует и начинается с третьей позиции столбца name):

```
mysql> SELECT name FROM metal WHERE name LIKE '__pp%';
+--------+
| name   |
+--------+
| copper |
+--------+
```

Совпадение с шаблоном SQL достигается, только если ему соответствует все сравниваемое значение целиком. То есть из двух приведенных ниже сравнений успешно выполнится только второе:

```
'abc' LIKE 'b'
'abc' LIKE '%b%'
```

Чтобы изменить условие поиска на обратное, используйте оператор NOT LIKE. Следующий запрос находит строки, которые не содержат символов i:

```
mysql> SELECT name FROM metal WHERE name NOT LIKE '%i%';
+---------+
| name    |
+---------+
| copper  |
| gold    |
| lead    |
| mercury |
+---------+
```

Шаблоны SQL не соответствуют значениям NULL (это относится и к LIKE, и к NOT LIKE):

```
mysql> SELECT NULL LIKE '%', NULL NOT LIKE '%';
+---------------+-------------------+
| NULL LIKE '%' | NULL NOT LIKE '%' |
+---------------+-------------------+
|          NULL |              NULL |
+---------------+-------------------+
```

В некоторых случаях поиск по образцу эквивалентен поиску подстроки. Например, использование шаблонов для поиска строк, находящихся с одного или другого конца строки (табл. 4.2), подобно использованию LEFT() или RIGHT():

Таблица 4.2. Аналогичные операции

Поиск по образцу	Поиск подстроки
str LIKE 'abc%'	LEFT(str,3) = 'abc'
str LIKE '%abc'	RIGHT(str,3) = 'abc'

Если вы работаете с индексированным столбцом, то, выбирая между поиском по образцу и эквивалентным выражением LEFT(), вы, вероятно, обнаружите, что поиск по образцу работает быстрее. MySQL может использовать индекс для сужения области поиска по образцу, начинающемуся с литерной строки, а при работе с LEFT() такой возможности нет.

4.7. Поиск по образцу с помощью регулярных выражений

Задача

Вы хотите выполнить не буквальное сравнение, а проверку на соответствие образцу.

Использование образцов для нестроковых значений

В отличие от некоторых других баз данных MySQL допускает поиск по образцу для числовых значений и дат. Ниже представлено несколько способов проверки значения d типа DATE при помощи вызовов функций, которые извлекают части даты и проверяют их соответствие образцу. Пары выражений в табл. 4.3 истинны для дат 1976 года, относящихся к апрелю или являющихся первым днем месяца.

Таблица 4.3. Способы проверки дат

Проверка значения функции	Проверка соответствия образцу
YEAR(d) = 1976	d LIKE '1976-%'
MONTH(d) = 4	d LIKE '%-04-%'
DAYOFMONTH(d) = 1	d LIKE '%-01'

Решение

Используйте оператор REGEXP и регулярные выражения, представленные в данном разделе, или воспользуйтесь шаблоном SQL, описанным в рецепте 4.6.

Обсуждение

Шаблоны SQL (см. рецепт 4.6) присутствуют и в других системах управления базами данных, поэтому они могут быть вынесены за пределы MySQL. Но их возможности ограничены. Например, можно без труда написать такой шаблон SQL, как %abc%, для нахождения строк, содержащих abc, но нельзя создать единый шаблон, который соответствовал бы всем строкам, содержащим любой из символов a, b или c. Невозможно построить образец, который распознавал бы содержимое строки символьного типа, например, буквы это или цифры. Для выполнения таких операций MySQL позволяет применять

другой способ поиска по образцу на основе использования регулярных выражений и оператора REGEXP (или NOT REGEXP для инвертирования).[1] Операция поиска с использованием регулярных выражений имеет собственный набор специальных символов (табл. 4.4), отличных от % и _ (оба эти символа не имеют специального значения в регулярных выражениях):

Таблица 4.4. Специальные символы в регулярных выражениях

Образец	Что соответствует такому образцу
^	Начало строки
$	Конец строки
.	Любой одиночный символ
[...]	Любой символ, приведенный в квадратных скобках
[^...]	Любой символ, не приведенный в квадратных скобках
p1|p2|p3	Дизъюнкция; соответствие любому из образцов p1, p2 или p3
*	Ноль или более экземпляров предыдущего элемента
+	Один или более экземпляров предыдущего элемента
{*n*}	*n* экземпляров предыдущего элемента
{*m*,*n*}	От *m* до *n* экземпляров предыдущего элемента

Символы шаблонов регулярных выражений могут быть вам уже знакомы, так как многие из них используются в *vi*, *grep*, *sed* и других программах UNIX, поддерживающих регулярные выражения. Большинство из них используется в регулярных выражениях, распознаваемых Perl, PHP и Python. (Например, в главе 10 рассказано о поиске по образцу в сценариях Perl.) Что касается Java, библиотеки классов Jakarta ORO и Regexp содержат функции поиска по образцу, также использующие вышеперечисленные символы.

В предыдущем разделе, описывающем шаблоны SQL, было рассказано, как искать подстроки, находящиеся в начале, конце или начинающиеся с любой или указанной позиции в строке. То же самое можно делать при помощи регулярных выражений:

- Строки, начинающиеся с определенной подстроки:

```
mysql> SELECT name FROM metal WHERE name REGEXP '^co';
+--------+
| name   |
+--------+
| copper |
+--------+
```

- Строки, завершающиеся определенной подстрокой:

[1] Оператор RLIKE — это синоним REGEXP. Он создан для совместимости с mSQL (mini-SQL) и упрощает портирование запросов из mSQL в MySQL.

```
mysql> SELECT name FROM metal WHERE name REGEXP 'er$';
+--------+
| name   |
+--------+
| copper |
| silver |
+--------+
```

- Строки, содержащие определенную подстроку:

```
mysql> SELECT name FROM metal WHERE name REGEXP 'er';
+---------+
| name    |
+---------+
| copper  |
| mercury |
| silver  |
+---------+
```

- Строки, содержащие определенную подстроку, начинающуюся с указанной позиции:

```
mysql> SELECT name FROM metal WHERE name REGEXP '^..pp';
+--------+
| name   |
+--------+
| copper |
+--------+
```

Кроме того, регулярные выражения имеют дополнительные возможности и могут выполнять такие виды поисков, которые недоступны шаблонам SQL. Например, регулярные выражения могут содержать классы символов, соответствующие любому символу класса:

- Чтобы создать класс символов, перечислите символы, которым должен будет соответствовать класс, в квадратных скобках. Например, образец `[abc]` соответствует любому из символов `a`, `b` или `c`.
- Классы могут указывать диапазоны символов: задайте начало и конец диапазона и поставьте между ними тире. Образец `[a-z]` соответствует любой букве, `[0-9]` – любой цифре, а `[a-z0-9]` соответствует и буквам, и цифрам.
- Чтобы инвертировать класс символов (задать соответствие любому символу, кроме указанных в классе), предварите список символом `^`. Например, `[^0-9]` соответствует любым символам, кроме цифр.

Регулярные выражения MySQL также поддерживают классы символов POSIX. Эти классы соответствуют специальным наборам символов (табл. 4.5).

Таблица 4.5. Классы символов POSIX

Класс POSIX	Чему соответствует класс
`[:alnum:]`	Буквенные и цифровые символы
`[:alpha:]`	Буквенные символы

Таблица 4.5 (продолжение)

Класс POSIX	Чему соответствует класс
[:blank:]	Пробельные символы (пробел или знак табуляции)
[:cntrl:]	Управляющие символы
[:digit:]	Цифры
[:graph:]	Графические символы (не пробельные)
[:lower:]	Буквенные символы в нижнем регистре
[:print:]	Графические символы или пробел
[:punct:]	Знаки пунктуации
[:space:]	Пробел, табуляция, новая строка, возврат каретки
[:upper:]	Буквенные символы в верхнем регистре
[:xdigit:]	Шестнадцатеричные цифры (0–9, a–f, A–F)

Классы POSIX предназначены для использования в классах символов, так что заключайте их в квадратные скобки. Следующее выражение соответствует значениям, которые могут содержать любые символы шестнадцатеричных цифр:

```
mysql> SELECT name, name REGEXP '[[:xdigit:]]' FROM metal;
+----------+---------------------------+
| name     | name REGEXP '[[:xdigit:]]' |
+----------+---------------------------+
| copper   |                         1 |
| gold     |                         1 |
| iron     |                         0 |
| lead     |                         1 |
| mercury  |                         1 |
| platinum |                         1 |
| silver   |                         1 |
| tin      |                         0 |
+----------+---------------------------+
```

Регулярные выражения могут содержать дизъюнкцию:

выбор1|*выбор2*|...

Дизъюнкция похожа на класс символов – она соответствует любому из вариантов. Но в отличие от класса символов, дизъюнкция может включать не только отдельные символы, но и строки, и даже образцы. Например, следующая дизъюнкция соответствует строкам, которые начинаются с гласной или заканчиваются на `er`:

```
mysql> SELECT name FROM metal WHERE name REGEXP '^[aeiou]|er$';
+--------+
| name   |
+--------+
```

```
| copper |
| iron   |
| silver |
+--------+
```

Можно группировать дизъюнкции, используя скобки. Например, если вы хотите найти строки, целиком состоящие только из букв или только из цифр, попробуйте выполнить такой запрос, использующий дизъюнкцию:

```
mysql> SELECT '0m' REGEXP '^[[:digit:]]+|[[:alpha:]]+$';
+-----------------------------------------+
| '0m' REGEXP '^[[:digit:]]+|[[:alpha:]]+$' |
+-----------------------------------------+
|                                       1 |
+-----------------------------------------+
```

Как видно из результата запроса, поиск по образцу не работает. Дело в том, что ^ применяется к первому варианту, а $ – ко второму. Так что на самом деле образец соответствует строкам, которые начинаются с одной или нескольких цифр, или строкам, которые заканчиваются одной или несколькими буквами. А вот если заключить дизъюнкцию в скобки, то ^ и $ будут применены к обоим вариантам, и образец будет работать, как и ожидалось:

```
mysql> SELECT '0m' REGEXP '^([[:digit:]]+|[[:alpha:]]+)$';
+-------------------------------------------+
| '0m' REGEXP '^([[:digit:]]+|[[:alpha:]]+)$' |
+-------------------------------------------+
|                                         0 |
+-------------------------------------------+
```

В отличие от поиска по шаблонам SQL, когда соответствие достигается только при совпадении с шаблоном всего значения, поиск при помощи регулярных выражений успешен, если образец совпадает с любой частью значения. Два приведенных ниже примера поиска эквиваленты в том смысле, что любому из них соответствуют только строки, содержащие символ b, но первый пример эффективнее, так как образец проще:

```
'abc' REGEXP 'b'
'abc' REGEXP '^.*b.*$'
```

Регулярные выражения не соответствуют значениям NULL. Это относится и к REGEXP, и к NOT REGEXP:

```
mysql> SELECT NULL REGEXP '.*', NULL NOT REGEXP '.*';
+-----------------+---------------------+
| NULL REGEXP '.*' | NULL NOT REGEXP '.*' |
+-----------------+---------------------+
|            NULL |                NULL |
+-----------------+---------------------+
```

Так как регулярное выражение соответствует строке, если образец найден в любой ее части, необходимо следить за тем, чтобы случайно не задать образец, соответствующий пустой строке. Если вы укажете такой образец, он будет соответствовать любым значениям, отличным от NULL. Например,

образец a* соответствует любому количеству символов a, даже нулевому. Если вашей целью является получение только строк, содержащих непустые последовательности символов a, используйте a+. Такой образец (с +) требует вхождения в строку одного или более экземпляров указанного элемента.

Аналогично поиску по шаблонам SQL при помощи LIKE, поиск при помощи регулярных выражений в некоторых случаях эквивалентен поиску подстрок. Метасимволы ^ и $ действуют подобно LEFT() и RIGHT(), по крайней мере, если речь идет о буквенных строках (табл. 4.6):

Таблица 4.6. Аналогичные операции

Поиск по образцу	Поиск подстроки
`str REGEXP '^abc'`	`LEFT(str,3) = 'abc'`
`str REGEXP 'abc$'`	`RIGHT(str,3) = 'abc'`

Для небуквенных строк создать эквивалентный поиск при помощи подстроки обычно не удается. Например, чтобы найти строки, начинающиеся с любой непустой последовательности цифр, можно использовать такое сравнение с образцом:

```
str REGEXP '^[0-9]+'
```

Функция LEFT() этого не умеет (как и LIKE, кстати).

4.8. Буквальная интерпретация метасимволов в шаблонах

Задача

Вы хотите выполнить поиск по образцу, в который входит специальный символ, так, чтобы этот символ интерпретировался буквально, а не как специальный.

Решение

Экранируйте специальный символ при помощи обратного слэша. Или даже двух.

Обсуждение

В основе поиска по образцу лежит использование метасимволов, имеющих специальное значение и означающих нечто, отличное от них самих. Поэтому для выполнения сравнения с литеральным экземпляром метасимвола необходимо как-то отключить его специальное значение. Используем символ обратного слэша (\). Предположим, что таблица metachar содержит такие строки:

```
mysql> SELECT c FROM metachar;
+------+
| c    |
```

```
+------+
| %    |
| _    |
| .    |
| ^    |
| $    |
| \    |
+------+
```

Образец, состоящий только из метасимволов SQL, соответствует всем значениям таблицы, кроме самих метасимволов:

```
mysql> SELECT c, c LIKE '%', c LIKE '_' FROM metachar;
+------+------------+------------+
| c    | c LIKE '%' | c LIKE '_' |
+------+------------+------------+
| %    |          1 |          1 |
| _    |          1 |          1 |
| .    |          1 |          1 |
| ^    |          1 |          1 |
| $    |          1 |          1 |
| \    |          1 |          1 |
+------+------------+------------+
```

Для того чтобы в запросе использовались буквальные значения метасимволов SQL, поставьте перед образцом обратный слэш:

```
mysql> SELECT c, c LIKE '\%', c LIKE '\_' FROM metachar;
+------+-------------+-------------+
| c    | c LIKE '\%' | c LIKE '\_' |
+------+-------------+-------------+
| %    |           1 |           0 |
| _    |           0 |           1 |
| .    |           0 |           0 |
| ^    |           0 |           0 |
| $    |           0 |           0 |
| \    |           0 |           0 |
+------+-------------+-------------+
```

Нечто подобное делалось и для метасимволов регулярных выражений. Например, каждое из приведенных ниже регулярных выражений соответствует каждой строке таблицы:

```
mysql> SELECT c, c REGEXP '.', c REGEXP '^', c REGEXP '$' FROM metachar;
+------+--------------+--------------+--------------+
| c    | c REGEXP '.' | c REGEXP '^' | c REGEXP '$' |
+------+--------------+--------------+--------------+
| %    |            1 |            1 |            1 |
| _    |            1 |            1 |            1 |
| .    |            1 |            1 |            1 |
| ^    |            1 |            1 |            1 |
| $    |            1 |            1 |            1 |
| \    |            1 |            1 |            1 |
+------+--------------+--------------+--------------+
```

Для буквальной интерпретации метасимволов нужно просто добавить обратный слэш, не так ли? Давайте попробуем:

```
mysql> SELECT c, c REGEXP '\.', c REGEXP '\^', c REGEXP '\$' FROM metachar;
+------+---------------+---------------+---------------+
| c    | c REGEXP '\.' | c REGEXP '\^' | c REGEXP '\$' |
+------+---------------+---------------+---------------+
| %    |             1 |             1 |             1 |
| _    |             1 |             1 |             1 |
| .    |             1 |             1 |             1 |
| ^    |             1 |             1 |             1 |
| $    |             1 |             1 |             1 |
| \    |             1 |             1 |             1 |
+------+---------------+---------------+---------------+
```

Ничего не получилось, потому что регулярные выражения обрабатываются несколько иначе, чем шаблоны SQL. Если вы используете `REGEXP`, то для буквальной интерпретации метасимволов потребуются два обратных слэша:

```
mysql> SELECT c, c REGEXP '\\.', c REGEXP '\\^', c REGEXP '\\$' FROM metachar;
+------+----------------+----------------+----------------+
| c    | c REGEXP '\\.' | c REGEXP '\\^' | c REGEXP '\\$' |
+------+----------------+----------------+----------------+
| %    |              0 |              0 |              0 |
| _    |              0 |              0 |              0 |
| .    |              1 |              0 |              0 |
| ^    |              0 |              1 |              0 |
| $    |              0 |              0 |              1 |
| \    |              0 |              0 |              0 |
+------+----------------+----------------+----------------+
```

Обратный слэш подавляет специальную интерпретацию других символов, то есть сам тоже является специальным символом. Чтобы отключить специальную интерпретацию символа обратного слэша, используйте два (в шаблонах SQL) или четыре (в регулярных выражениях) обратных слэша:

```
mysql> SELECT c, c LIKE '\\', c REGEXP '\\\\' FROM metachar;
+------+-------------+-----------------+
| c    | c LIKE '\\' | c REGEXP '\\\\' |
+------+-------------+-----------------+
| %    |           0 |               0 |
| _    |           0 |               0 |
| .    |           0 |               0 |
| ^    |           0 |               0 |
| $    |           0 |               0 |
| \    |           1 |               1 |
+------+-------------+-----------------+
```

Страшно даже представить себе, сколько обратных слэшей придется использовать при выдаче запроса из программы. При этом более чем вероятно, что обратный слэш является специальным символом в вашем языке программирования, тогда каждый из них придется продублировать.

Внутри класса символов для включения в него используемых в нем же символов следуйте таким правилам:

- Для включения литерала] укажите его первым в списке.
- Для включения литерала - укажите его первым или последним.
- Для включения литерала ^ укажите его не первым.
- Для включения литерала \ продублируйте его.

4.9. Управление чувствительностью к регистру при сравнении строк

Задача

Сравнение строк чувствительно к регистру тогда, когда это нежелательно, или наоборот.

Решение

Измените чувствительность строк к регистру.

Обсуждение

В примерах предыдущих разделов не учитывался регистр букв. Но в некоторых случаях необходимо иметь уверенность в том, что строковая операция чувствительна (или не чувствительна) к регистру. В этом разделе рассказано о том, как добиться этого для обычных сравнений. В рецепте 4.10 описана чувствительность к регистру операций сравнения с образцом.

По умолчанию сравнение строк в MySQL не чувствительно к регистру:

```
mysql> SELECT name, name = 'lead', name = 'LEAD' FROM metal;
+----------+---------------+---------------+
| name     | name = 'lead' | name = 'LEAD' |
+----------+---------------+---------------+
| copper   |             0 |             0 |
| gold     |             0 |             0 |
| iron     |             0 |             0 |
| lead     |             1 |             1 |
| mercury  |             0 |             0 |
| platinum |             0 |             0 |
| silver   |             0 |             0 |
| tin      |             0 |             0 |
+----------+---------------+---------------+
```

Нечувствительность к регистру затрагивает и сравнения на взаимный порядок:

```
mysql> SELECT name, name < 'lead', name < 'LEAD' FROM metal;
+----------+---------------+---------------+
| name     | name < 'lead' | name < 'LEAD' |
+----------+---------------+---------------+
```

```
| copper   |             1 |             1 |
| gold     |             1 |             1 |
| iron     |             1 |             1 |
| lead     |             0 |             0 |
| mercury  |             0 |             0 |
| platinum |             0 |             0 |
| silver   |             0 |             0 |
| tin      |             0 |             0 |
+----------+---------------+---------------+
```

Если вы знакомы со схемой сортировки ASCII, то знаете, что коды ASCII для букв нижнего регистра больше, чем коды букв верхнего регистра, так что результаты второго столбца должны были бы вас удивить. Такие результаты показывают, что по умолчанию упорядочение строк производится без учета регистра букв, так что и A, и a считаются лексически меньшими, чем B.

Сравнения строк чувствительны к регистру, только если хотя бы один из операндов является двоичной строкой. Существуют следующие методы контроля чувствительности к регистру в операциях сравнения строк:

- Чтобы сделать чувствительным к регистру сравнение, которое само по себе таким не было, приведите один из операндов в двоичную форму, используя ключевое слово BINARY. Не имеет значения, какую именно из строк вы сделаете двоичной, – как только одна из них станет двоичной, сравнение станет чувствительным к регистру:

```
mysql> SELECT name, name = BINARY 'lead', BINARY name = 'LEAD' FROM metal;
+----------+----------------------+----------------------+
| name     | name = BINARY 'lead' | BINARY name = 'LEAD' |
+----------+----------------------+----------------------+
| copper   |                    0 |                    0 |
| gold     |                    0 |                    0 |
| iron     |                    0 |                    0 |
| lead     |                    1 |                    0 |
| mercury  |                    0 |                    0 |
| platinum |                    0 |                    0 |
| silver   |                    0 |                    0 |
| tin      |                    0 |                    0 |
+----------+----------------------+----------------------+
```

  Начиная с MySQL 3.23 в качестве оператора приведения типа используется BINARY.

- Чтобы сделать нечувствительной к регистру операцию сравнения, которая должна была бы учитывать регистр, преобразуйте обе строки к одному регистру, используя функцию UPPER() или LOWER():

```
mysql> SELECT UPPER('A'), UPPER('b'), UPPER('A') < UPPER('b');
+------------+------------+-------------------------+
| UPPER('A') | UPPER('b') | UPPER('A') < UPPER('b') |
+------------+------------+-------------------------+
| A          | B          |                       1 |
+------------+------------+-------------------------+
```

```
mysql> SELECT LOWER('A'), LOWER('b'), LOWER('A') < LOWER('b');
+------------+------------+-------------------------+
| LOWER('A') | LOWER('b') | LOWER('A') < LOWER('b') |
+------------+------------+-------------------------+
| a          | b          |                       1 |
+------------+------------+-------------------------+
```

Эти же приемы можно применять и к функциям сравнения строк. Например, функция `STRCMP()` принимает два строковых аргумента и возвращает –1, 0 или 1 в зависимости от того, лексически меньше, равна или больше первая строка по отношению ко второй. До версии MySQL 4.0.0 включительно функция `STRCMP()` была чувствительна к регистру; она всегда воспринимала свои аргументы как двоичные строки независимо от их реального типа:

```
mysql> SELECT STRCMP('Abc','abc'), STRCMP('abc','abc'), STRCMP('abc','Abc');
+--------------------+--------------------+--------------------+
| STRCMP('Abc','abc') | STRCMP('abc','abc') | STRCMP('abc','Abc') |
+--------------------+--------------------+--------------------+
|                 -1 |                  0 |                  1 |
+--------------------+--------------------+--------------------+
```

Однако начиная с MySQL 4.0.1 функция `STRCMP()` больше не чувствительна к регистру:

```
mysql> SELECT STRCMP('Abc','abc'), STRCMP('abc','abc'), STRCMP('abc','Abc');
+--------------------+--------------------+--------------------+
| STRCMP('Abc','abc') | STRCMP('abc','abc') | STRCMP('abc','Abc') |
+--------------------+--------------------+--------------------+
|                  0 |                  0 |                  0 |
+--------------------+--------------------+--------------------+
```

Чтобы сохранить поведение функции в версиях до 4.0.1, сделайте один из ее аргументов двоичной строкой:

```
mysql> SELECT STRCMP(BINARY 'Abc','abc'), STRCMP(BINARY 'abc','Abc');
+---------------------------+---------------------------+
| STRCMP(BINARY 'Abc','abc') | STRCMP(BINARY 'abc','Abc') |
+---------------------------+---------------------------+
|                        -1 |                         1 |
+---------------------------+---------------------------+
```

Кстати, заметьте, что нулевое и ненулевое значения, возвращаемые функцией `STRCMP()`, означают равенство и неравенство соответственно. В этом отличие функции от оператора сравнения `=`, который возвращает нулевое и ненулевое значения для неравенства и равенства соответственно.

Чтобы избежать проблем, запомните основные правила, определяющие, является ли строка двоичной:

- Любую буквенную строку, строковое выражение или строковый столбец можно сделать двоичными, предварив их ключевым словом `BINARY`. Если же ключевого слова нет, действуют следующие правила.
- Строковое выражение является двоичным, если хотя бы одна из составляющих его строк двоичная, иначе оно не является двоичным. Например,

результат, возвращенный выражением CONCAT(), является двоичным, так как второй аргумент – двоичный:

```
CONCAT('This is a ',BINARY 'binary',' string')
```

- Чувствительность строкового столбца к регистру определяется его типом. Типы CHAR и VARCHAR по умолчанию не чувствительны к регистру, но их можно объявить как BINARY, тогда они станут чувствительными к регистру. Столбцы типов ENUM, SET и TEXT не чувствительны к регистру, а столбцы типа BLOB – чувствительны (см. таблицу в рецепте 4.0).

Итак, операции сравнения чувствительны к регистру, если в них участвует двоичная буквенная строка, или строковое выражение, или столбец типа CHAR BINARY, VARCHAR BINARY или BLOB. Сравнения же, в которых участвуют только недвоичные буквенные строки, или строковые выражения, или столбцы типа CHAR, VARCHAR, ENUM, SET или TEXT, не чувствительны к регистру.

Столбцы ENUM и SET не чувствительны к регистру. Более того, поскольку они внутренне хранятся в числовом виде, их нельзя объявлять как чувствительные к регистру в определении таблицы (добавляя ключевое слово BINARY). Но вы можете поставить ключевое слово BINARY в сравнении перед значениями ENUM и SET, чтобы сделать операцию чувствительной к регистру.

Чувствительность к регистру и скорость выполнения сравнений

Обычно чувствительные к регистру сравнения, содержащие двоичные строки, работают немного быстрее, чем нечувствительные к регистру, так как MySQL не приходится во время операции приводить буквы к одному регистру.

Если оказалось, что вы объявили столбец, используя тип, несовместимый с теми операциями сравнения, которые предполагается для него применять, вы можете изменить тип столбца при помощи предложения ALTER TABLE. Предположим, что у вас есть таблица для хранения новостных статей:

```
CREATE TABLE news
(
    id      INT UNSIGNED NOT NULL AUTO_INCREMENT,
    article BLOB NOT NULL,
    PRIMARY KEY (id)
);
```

Столбец article объявлен как BLOB, то есть тип, чувствительный к регистру. Если вы захотите преобразовать столбец так, чтобы он не был чувствителен к регистру, то можете изменить его тип на TEXT, используя одно из предложений ALTER TABLE:

```
ALTER TABLE news MODIFY article TEXT NOT NULL;
ALTER TABLE news CHANGE article article TEXT NOT NULL;
```

До версии MySQL 3.22.16 предложение `ALTER TABLE ... MODIFY` недоступно, и если вы работаете с более ранней версией, то можете использовать только `ALTER TABLE ... CHANGE`. Дополнительная информация приведена в главе 8.

4.10. Управление чувствительностью к регистру при поиске по образцу

Задача

Поиск по образцу чувствителен к регистру в тех случаях, когда вы этого не хотите, или наоборот.

Решение

Измените чувствительность строк к регистру.

Обсуждение

По умолчанию операция `LIKE` не чувствительна к регистру:

```
mysql> SELECT name, name LIKE '%i%', name LIKE '%I%' FROM metal;
+----------+-----------------+-----------------+
| name     | name LIKE '%i%' | name LIKE '%I%' |
+----------+-----------------+-----------------+
| copper   |               0 |               0 |
| gold     |               0 |               0 |
| iron     |               1 |               1 |
| lead     |               0 |               0 |
| mercury  |               0 |               0 |
| platinum |               1 |               1 |
| silver   |               1 |               1 |
| tin      |               1 |               1 |
+----------+-----------------+-----------------+
```

В настоящий момент не чувствительна к регистру и операция `REGEXP`.

```
mysql> SELECT name, name REGEXP 'i', name REGEXP 'I' FROM metal;
+----------+-----------------+-----------------+
| name     | name REGEXP 'i' | name REGEXP 'I' |
+----------+-----------------+-----------------+
| copper   |               0 |               0 |
| gold     |               0 |               0 |
| iron     |               1 |               1 |
| lead     |               0 |               0 |
| mercury  |               0 |               0 |
| platinum |               1 |               1 |
| silver   |               1 |               1 |
| tin      |               1 |               1 |
+----------+-----------------+-----------------+
```

Однако до версии MySQL 3.23.4 операции `REGEXP` *были чувствительны* к регистру:

```
mysql> SELECT name, name REGEXP 'i', name REGEXP 'I' FROM metal;
+----------+-----------------+-----------------+
| name     | name REGEXP 'i' | name REGEXP 'I' |
+----------+-----------------+-----------------+
| copper   |               0 |               0 |
| gold     |               0 |               0 |
| iron     |               1 |               0 |
| lead     |               0 |               0 |
| mercury  |               0 |               0 |
| platinum |               1 |               0 |
| silver   |               1 |               0 |
| tin      |               1 |               0 |
+----------+-----------------+-----------------+
```

Обратите внимание на то, что текущее поведение REGEXP (нечувствительность к регистру) может привести к некоторым интуитивно непонятным результатам:

```
mysql> SELECT 'a' REGEXP '[[:lower:]]', 'a' REGEXP '[[:upper:]]';
+-------------------------+-------------------------+
| 'a' REGEXP '[[:lower:]]' | 'a' REGEXP '[[:upper:]]' |
+-------------------------+-------------------------+
|                       1 |                       1 |
+-------------------------+-------------------------+
```

Оба выражения истинны, так как в случае нечувствительности к регистру [:lower:] и [:upper:] эквиваленты.

Изменить нежелательное для вас поведение операции поиска по образцу в отношении чувствительности к регистру можно посредством тех же приемов, что и для операции сравнения строк:

- Чтобы сделать поиск по образцу чувствительным к регистру, используйте двоичную строку для любого из операндов (например, при помощи ключевого слова BINARY). Следующий запрос показывает, что обычно недвоичный столбец name не чувствителен к регистру:

```
mysql> SELECT name, name LIKE '%i%%', name REGEXP 'i' FROM metal;
+----------+------------------+-----------------+
| name     | name LIKE '%i%%' | name REGEXP 'i' |
+----------+------------------+-----------------+
| copper   |                0 |               0 |
| gold     |                0 |               0 |
| iron     |                1 |               1 |
| lead     |                0 |               0 |
| mercury  |                0 |               0 |
| platinum |                1 |               1 |
| silver   |                1 |               1 |
| tin      |                1 |               1 |
+----------+------------------+-----------------+
```

Используем ключевое слово BINARY, чтобы заставить значения name стать чувствительными к регистру:

```
mysql> SELECT name, BINARY name LIKE '%I%', BINARY name REGEXP 'I' FROM metal;
+----------+-----------------------+-----------------------+
| name     | BINARY name LIKE '%I%' | BINARY name REGEXP 'I' |
+----------+-----------------------+-----------------------+
| copper   |                     0 |                     0 |
| gold     |                     0 |                     0 |
| iron     |                     0 |                     0 |
| lead     |                     0 |                     0 |
| mercury  |                     0 |                     0 |
| platinum |                     0 |                     0 |
| silver   |                     0 |                     0 |
| tin      |                     0 |                     0 |
+----------+-----------------------+-----------------------+
```

Использование BINARY заставляет [:lower:] и [:upper:] работать в регулярных выражениях так, как вам хотелось бы. Второе выражение следующего запроса выдает результат, который на самом деле является истинным только для букв верхнего регистра:

```
mysql> SELECT 'a' REGEXP '[[:upper:]]', BINARY 'a' REGEXP '[[:upper:]]';
+-------------------------+---------------------------------+
| 'a' REGEXP '[[:upper:]]' | BINARY 'a' REGEXP '[[:upper:]]' |
+-------------------------+---------------------------------+
|                       1 |                               0 |
+-------------------------+---------------------------------+
```

- Поиск по образцу для двоичного столбца чувствителен к регистру. Чтобы сделать его нечувствительным, преобразуйте оба операнда в один регистр. Давайте изменим таблицу metal, добавив в нее столбец binname, аналогичный столбцу name, но имеющий тип VARCHAR BINARY, а не VARCHAR:

```
mysql> ALTER TABLE metal ADD binname VARCHAR(20) BINARY;
mysql> UPDATE metal SET binname = name;
```

Первый из представленных ниже запросов показывает, что двоичный столбец binname чувствителен к регистру при поиске по образцу, а второй запрос показывает, как заставить столбец изменить свое поведение с помощью UPPER():

```
mysql> SELECT binname, binname LIKE '%I%', binname REGEXP 'I'
    -> FROM metal;
+----------+--------------------+--------------------+
| binname  | binname LIKE '%I%' | binname REGEXP 'I' |
+----------+--------------------+--------------------+
| copper   |                  0 |                  0 |
| gold     |                  0 |                  0 |
| iron     |                  0 |                  0 |
| lead     |                  0 |                  0 |
| mercury  |                  0 |                  0 |
| platinum |                  0 |                  0 |
| silver   |                  0 |                  0 |
```

```
| tin      |                  0 |                  0 |
+----------+--------------------+--------------------+
mysql> SELECT binname, UPPER(binname) LIKE '%I%', UPPER(binname) REGEXP 'I'
    -> FROM metal;
+----------+--------------------------+--------------------------+
| binname  | UPPER(binname) LIKE '%I%' | UPPER(binname) REGEXP 'I' |
+----------+--------------------------+--------------------------+
| copper   |                        0 |                        0 |
| gold     |                        0 |                        0 |
| iron     |                        1 |                        1 |
| lead     |                        0 |                        0 |
| mercury  |                        0 |                        0 |
| platinum |                        1 |                        1 |
| silver   |                        1 |                        1 |
| tin      |                        1 |                        1 |
+----------+--------------------------+--------------------------+
```

4.11. Поиск с помощью индекса FULLTEXT

Задача

Вы хотите выполнить поиск в тексте большого объема.

Решение

Используйте индекс FULLTEXT.

Обсуждение

Поиск по образцу может работать с любым количеством строк, но чем их больше, тем медленнее выполняется эта операция. Кроме того, часто приходится искать один и тот же текст в нескольких строковых столбцах, что приводит к созданию громоздких запросов:

```
SELECT * from имя_таблицы
WHERE столбец1 LIKE 'шаблон' OR столбец2 LIKE 'шаблон' OR столбец3 LIKE 'шаблон' ...
```

Полезной альтернативой (доступной начиная с версии MySQL 3.23.23) является использование FULLTEXT-поиска, предназначенного для просмотра больших объемов текста с одновременным просмотром нескольких столбцов. Добавьте в таблицу индекс FULLTEXT, затем используйте оператор MATCH для поиска строк индексированного столбца или столбцов. Индексирование FULLTEXT может применяться в таблицах MyISAM для столбцов типа CHAR, VARCHAR или TEXT.

FULLTEXT-поиск лучше всего продемонстрировать на тексте подходящего размера. Если у вас нет тестового набора данных, можно воспользоваться одним из свободно доступных хранилищ электронных текстов, имеющихся в Интернете. В примерах данного раздела использован текст Библии в версии King James Version (KJV) – достаточно большой и очень хорошо структури-

рованный текст: книга, глава, стих. Из-за своего объема этот набор данных не включен в дистрибутив recipes, но для него на веб-сайте книги «MySQL Cookbook» создан собственный дистрибутив mcb-kjv[1] (см. приложение А). Дистрибутив включает файл *kjv.txt*, который содержит записи стихов. Записи выглядят так:

```
0   Genesis 1   1   1   In the beginning God created the heaven and the earth.
0   Exodus  2   20  13  Thou shalt not kill.
N   Luke    42  17  32  Remember Lot's wife.
```

Каждая запись содержит поля:

- Раздел книги. Это или 0, или N, что означает Ветхий (Old) и Новый (New) Завет.
- Название книги и соответствующий номер, от 1 до 66.
- Номер главы и стиха.
- Текст стиха.

Чтобы импортировать записи в MySQL, создайте такую таблицу kjv:

```
CREATE TABLE kjv
(
    bsect   ENUM('O','N') NOT NULL,         # раздел книги (Завет)
    bname   VARCHAR(20) NOT NULL,           # название книги
    bnum    TINYINT UNSIGNED NOT NULL,      # номер книги
    cnum    TINYINT UNSIGNED NOT NULL,      # номер главы
    vnum    TINYINT UNSIGNED NOT NULL,      # номер стиха
    vtext   TEXT NOT NULL                   # текст стиха
) TYPE = MyISAM;
```

Затем загрузите файл *kjv.txt* в таблицу, выполнив такое предложение:

```
mysql> LOAD DATA LOCAL INFILE 'kjv.txt' INTO TABLE kjv;
```

Таблица kjv содержит столбцы как для названий книг (Genesis – Книга Бытия, Exodus – Исход, ...), так и для их номеров (1, 2, ...). Названия и номера книг четко соответствуют друг другу, и одни могут быть однозначно получены из других. Налицо избыточность данных, то есть таблица не приведена к нормальной форме. Чтобы избавиться от такой избыточности, можно хранить только номера книг (они занимают меньше места, чем названия) и при необходимости выводить названия книг как результаты запроса, используя соединение (join) с простой вспомогательной таблицей, сопоставляющей номерам книг их названия. Но пока я хочу избежать соединения. Поэтому таблица будет содержать названия книг для удобства восприятия результатов поиска и номера книг для удобства сортировки результатов по книгам.

---

[1] Дистрибутив mcb-kjv получен из текста KJV, доступного на сайте университета Биола (Biola) *http://unbound.biola.edu*, который был несколько изменен для того, чтобы его легче было использовать в рецептах. В дистрибутив mcb-kjv включена информация о том, чем он отличается от дистрибутива Biola.

После заполнения таблицы данными подготовим ее к полнотекстовому поиску, добавив индекс FULLTEXT. Для этого выполним предложение ALTER TABLE:[1]

```
mysql> ALTER TABLE kjv ADD FULLTEXT (vtext);
```

Чтобы выполнить поиск по индексу, используем MATCH() для указания индексированного столбца и AGAINST() для определения того, какой текст следует искать. Например, чтобы ответить на вопрос «Как часто встречается имя Mizraim» (ведь вас это всегда интересовало, не так ли?), будем просматривать столбец vtext при помощи такого запроса:

```
mysql> SELECT COUNT(*) from kjv WHERE MATCH(vtext) AGAINST('Mizraim');
+----------+
| COUNT(*) |
+----------+
|        4 |
+----------+
```

Чтобы найти соответствующие стихи, выберем столбцы, которые вы хотели бы видеть (для того чтобы результат поместился на странице, в примере используется \G):

```
mysql> SELECT bname, cnum, vnum, vtext
    -> FROM kjv WHERE MATCH(vtext) AGAINST('Mizraim')\G
*************************** 1. row ***************************
bname: Genesis
 cnum: 10
 vnum: 6
vtext: And the sons of Ham; Cush, and Mizraim, and Phut, and Canaan.
*************************** 2. row ***************************
bname: Genesis
 cnum: 10
 vnum: 13
vtext: And Mizraim begat Ludim, and Anamim, and Lehabim, and Naphtuhim,
*************************** 3. row ***************************
bname: 1 Chronicles
 cnum: 1
 vnum: 8
vtext: The sons of Ham; Cush, and Mizraim, Put, and Canaan.
*************************** 4. row ***************************
bname: 1 Chronicles
 cnum: 1
 vnum: 11
vtext: And Mizraim begat Ludim, and Anamim, and Lehabim, and Naphtuhim,
```

В данном конкретном случае результаты выводятся по порядку номеров книг, глав и стихов, но это простая случайность. По умолчанию FULLTEXT-поиск вы-

[1] Можно было включить определение индекса в исходное предложение CREATE TABLE, но обычно оказывается, что создание неиндексированной таблицы и добавление индекса при помощи предложения ALTER TABLE после заполнения таблицы данными эффективнее, чем загрузка большого набора данных в индексированную таблицу.

числяет величину релевантности и сортирует результаты начиная с наиболее релевантных. Чтобы обеспечить сортировку результата в необходимом вам порядке, добавьте явную инструкцию ORDER BY:

```
SELECT bname, cnum, vnum, vtext
FROM kjv WHERE MATCH(vtext) AGAINST('строка_поиска')
ORDER BY bnum, cnum, vnum;
```

Для сужения области поиска можно задать дополнительные условия. В следующем фрагменте выполняются постепенно уточняющиеся запросы для нахождения частоты упоминания имени Abraham во всем тексте KJV, в Новом Завете (New Testament), в книге «К евреям» (Hebrews) и в главе 11 этой книги:

```
mysql> SELECT COUNT(*) from kjv WHERE MATCH(vtext) AGAINST('Abraham');
+----------+
| COUNT(*) |
+----------+
|      216 |
+----------+
mysql> SELECT COUNT(*) from kjv
    -> WHERE MATCH(vtext) AGAINST('Abraham')
    -> AND bsect = 'N';
+----------+
| COUNT(*) |
+----------+
|       66 |
+----------+
mysql> SELECT COUNT(*) from kjv
    -> WHERE MATCH(vtext) AGAINST('Abraham')
    -> AND bname = 'Hebrews';
+----------+
| COUNT(*) |
+----------+
|       10 |
+----------+
mysql> SELECT COUNT(*) from kjv
    -> WHERE MATCH(vtext) AGAINST('Abraham')
    -> AND bname = 'Hebrews' AND cnum = 11;
+----------+
| COUNT(*) |
+----------+
|        2 |
+----------+
```

Если вы планируете использовать условия поиска, относящиеся к другим не-FULLTEXT столбцам, то для повышения производительности таких запросов можно добавить обычные индексы для этих столбцов. Например, можно проиндексировать столбцы номеров книги, главы и стиха:

```
mysql> ALTER TABLE kjv ADD INDEX (bnum), ADD INDEX (cnum), ADD INDEX (vnum);
```

Строка поиска в запросах FULLTEXT может представлять собой не только отдельное слово. Казалось бы, указание дополнительных слов в строке поиска должно делать поиск более точным. Но на самом деле поиск только расширяется, так как при полнотекстовом поиске возвращаются записи, содержащие любое из указанных слов (фактически выполняется поиск с логическим ИЛИ для всех указанных слов). Рассмотрим запросы, возвращающие все большее количество стихов по мере добавления новых слов поиска:

```
mysql> SELECT COUNT(*) from kjv
    -> WHERE MATCH(vtext) AGAINST('Abraham');
+----------+
| COUNT(*) |
+----------+
|      216 |
+----------+
mysql> SELECT COUNT(*) from kjv
    -> WHERE MATCH(vtext) AGAINST('Abraham Sarah');
+----------+
| COUNT(*) |
+----------+
|      230 |
+----------+
mysql> SELECT COUNT(*) from kjv
    -> WHERE MATCH(vtext) AGAINST('Abraham Sarah Ishmael Isaac');
+----------+
| COUNT(*) |
+----------+
|      317 |
+----------+
```

Выполнение поиска, возвращающего записи, в которых присутствуют все слова строки поиска, описано в рецепте 4.13.

Если вы хотите использовать FULLTEXT-поиск для параллельного просмотра нескольких столбцов, укажите их имена при создании индекса:

```
ALTER TABLE имя_таблицы ADD FULLTEXT (столбец1, столбец2, столбец3);
```

Чтобы создать запрос, использующий такой индекс, укажите имена тех же самых столбцов в списке MATCH():

```
SELECT ... FROM имя_таблицы
WHERE MATCH(столбец1, столбец2, столбец3) AGAINST('строка_поиска');
```

См. также

Индексы FULLTEXT обеспечивают быстрый и легкий способ создания простой поисковой машины. Можно использовать эту возможность для организации веб-интерфейса к индексированному тексту. На сайте книги «MySQL Cookbook» представлена реализованная таким способом страница поиска KJV.

4.12. FULLTEXT–поиск и короткие слова

Задача

FULLTEXT-поиск по коротким словам не возвращает записей.

Решение

Измените значение параметра минимальной длины слова для механизма индексирования.

Обсуждение

В тексте, подобном KJV, некоторые слова имеют особое значение, например «Бог» или «грех». Но если вы, работая с сервером MySQL 3.23, выполните FULLTEXT-поиск этих слов в таблице kjv, то обнаружите любопытный результат – ни того, ни другого слова как будто никогда и не было в тексте:

```
mysql> SELECT COUNT(*) FROM kjv WHERE MATCH(vtext) AGAINST('God');
+----------+
| COUNT(*) |
+----------+
|        0 |
+----------+
mysql> SELECT COUNT(*) FROM kjv WHERE MATCH(vtext) AGAINST('sin');
+----------+
| COUNT(*) |
+----------+
|        0 |
+----------+
```

Одно из свойств индексатора – игнорирование «слишком общих» слов (то есть слов, присутствующих более чем в половине записей). Так, из индекса удаляются слова типа «the» и «and», но в данном случае мы имеем дело с чем-то иным. Давайте сосчитаем общее количество записей и (при помощи шаблонов SQL) количество записей, содержащих каждое из слов:[1]

```
mysql> SELECT COUNT(*) AS 'total verses',
    -> COUNT(IF(vtext LIKE '%God%',1,NULL)) AS 'verses containing "God"',
    -> COUNT(IF(vtext LIKE '%sin%',1,NULL)) AS 'verses containing "sin"'
    -> FROM kjv;
+--------------+------------------------+------------------------+
| total verses | verses containing "God" | verses containing "sin" |
+--------------+------------------------+------------------------+
|        31102 |                   4118 |                   1292 |
+--------------+------------------------+------------------------+
```

[1] Использование COUNT() для формирования нескольких счетчиков для одного набора значений описано в рецепте 7.1.

Ни одно из слов не присутствует более чем в половине стихов, так что полнотекстовый поиск не удался не из-за частого употребления слов. Причина в том, что по умолчанию в индексы не включаются слова, длина которых меньше четырех символов. Если вы работаете с сервером MySQL 3.23, то вам ничего не удастся с этим поделать (по крайней мере, ничего более простого, чем обращение к исходным текстам MySQL с их повторной компиляцией). Но начиная с версии MySQL 4.0 минимальная длина слова является настраиваемым параметром, который можно изменить, задав переменную сервера `ft_min_word_len`. Например, чтобы включать в индекс слова, содержащие три и более символов, добавьте строку `set-variable` в группу `[mysqld]` файла */etc/my.cnf* (или другого файла, в котором вы храните настройки сервера):

```
[mysqld]
set-variable = ft_min_word_len=3
```

Сохраните изменения, перезапустите сервер и пересоздайте индекс `FULLTEXT`, чтобы новое значение вступило в силу:

```
mysql> ALTER TABLE kjv DROP INDEX vtext;
mysql> ALTER TABLE kjv ADD FULLTEXT (vtext);
```

Давайте посмотрим, включает ли новый индекс короткие слова:

```
mysql> SELECT COUNT(*) FROM kjv WHERE MATCH(vtext) AGAINST('God');
+----------+
| COUNT(*) |
+----------+
|     3878 |
+----------+
mysql> SELECT COUNT(*) FROM kjv WHERE MATCH(vtext) AGAINST('sin');
+----------+
| COUNT(*) |
+----------+
|      389 |
+----------+
```

Так-то лучше!

Но почему запрос с `MATCH()` находит 3878 и 389 записей, в то время как приведенный ранее запрос с `LIKE` нашел 4118 и 1292 записей? Поиск по образцу с помощью `LIKE` ищет соответствующие подстроки, а поиск `FULLTEXT`, осуществляемый `MATCH()`, ищет только целые слова.

4.13. Включение и исключение слов из FULLTEXT-поиска

Задача

Вы хотите специально указать слова, которые должны присутствовать или быть исключены из `FULLTEXT`-поиска.

Решение

Используйте FULLTEXT-поиск в логическом (Boolean) режиме.

Обсуждение

Обычно FULLTEXT-поиск возвращает записи, содержащие любое из слов строки поиска, даже если некоторые другие отсутствуют. Например, следующий запрос находит записи, которые содержат хотя бы одно из имен David или Goliath:

```
mysql> SELECT COUNT(*) FROM kjv
    -> WHERE MATCH(vtext) AGAINST('David Goliath');
+----------+
| COUNT(*) |
+----------+
|      934 |
+----------+
```

Но что делать, если вам нужны только записи, содержащие оба слова? Можно переписать запрос так, чтобы искать слова по отдельности, и соединить результаты при помощи оператора AND:

```
mysql> SELECT COUNT(*) FROM kjv
    -> WHERE MATCH(vtext) AGAINST('David')
    -> AND MATCH(vtext) AGAINST('Goliath');
+----------+
| COUNT(*) |
+----------+
|        2 |
+----------+
```

Начиная с версии MySQL 4.0.1 есть еще одна возможность потребовать присутствия в выводе нескольких слов – поиск в логическом режиме. Для того чтобы показать, что слово строки поиска должно содержаться в каждой возвращенной строке, поставьте перед ним символ + и завершите строку словами IN BOOLEAN MODE:

```
mysql> SELECT COUNT(*) FROM kjv
    -> WHERE MATCH(vtext) AGAINST('+David +Goliath' IN BOOLEAN MODE)
+----------+
| COUNT(*) |
+----------+
|        2 |
+----------+
```

Поиск в логическом режиме также позволяет исключать слова. Просто поставьте перед каждым нежелательным словом символ -. Следующие запросы выбирают записи таблицы kjv, содержащие имя David, но не содержащие имя Goliath, или наоборот:

```
mysql> SELECT COUNT(*) FROM kjv
    -> WHERE MATCH(vtext) AGAINST('+David -Goliath' IN BOOLEAN MODE)
```

```
+----------+
| COUNT(*) |
+----------+
|      928 |
+----------+
mysql> SELECT COUNT(*) FROM kjv
    -> WHERE MATCH(vtext) AGAINST('-David +Goliath' IN BOOLEAN MODE)
+----------+
| COUNT(*) |
+----------+
|        4 |
+----------+
```

Еще одним специальным символом логического режима поиска является `*`, добавляемый в конец слова и действующий как групповой символ. Следующий запрос находит записи, которые содержат не только `whirl`, но и такие слова, как `whirls`, `whirleth` и `whirlwind`:

```
mysql> SELECT COUNT(*) FROM kjv
    -> WHERE MATCH(vtext) AGAINST('whirl*' IN BOOLEAN MODE);
+----------+
| COUNT(*) |
+----------+
|       28 |
+----------+
```

4.14. Поиск фразы при помощи индекса FULLTEXT

Задача

Вы хотите найти при помощи индекса `FULLTEXT` фразу, то есть набор смежных слов, расположенных в определенном порядке.

Решение

Используйте возможность поиска фразы, предоставляемую `FULLTEXT`-поиском, или комбинируйте `FULLTEXT`-поиск слов и обычный поиск по образцу.

Обсуждение

Чтобы найти записи, содержащие определенную фразу, недостаточно просто выполнить `FULLTEXT`-поиск:

```
mysql> SELECT COUNT(*) FROM kjv
    -> WHERE MATCH(vtext) AGAINST('still small voice');
+----------+
| COUNT(*) |
+----------+
|      548 |
+----------+
```

Запрос возвращает результат, но не тот, который хотелось бы получить. FULLTEXT-поиск вычисляет релевантность по присутствию каждого отдельного слова, вне зависимости от того, где именно в столбце vtext оно встретилось. Величина релевантности будет ненулевой до тех пор, пока поиск будет обнаруживать хотя бы одно слово. Поэтому такие запросы обычно находят слишком много записей.

В MySQL версии 4.0.2 у FULLTEXT-поиска появилась возможность поиска фраз в логическом режиме. Если вы хотите найти строки, содержащие какую-то фразу, просто заключите ее в двойные кавычки:

```
mysql> SELECT COUNT(*) FROM kjv
    -> WHERE MATCH(vtext) AGAINST('"still small voice"' IN BOOLEAN MODE);
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
```

Если же вы используете более раннюю версию, необходим обходной путь. Можно выполнить поиск в логическом режиме, потребовав присутствия каждого слова, но проблема все же не будет решена, так как порядок слов никак не учитывается:

```
mysql> SELECT COUNT(*) FROM kjv
    -> WHERE MATCH(vtext)
    -> AGAINST('+still +small +voice' IN BOOLEAN MODE);
+----------+
| COUNT(*) |
+----------+
|        3 |
+----------+
```

Если же использовать поиск по шаблону SQL, то будет возвращен правильный результат:

```
mysql> SELECT COUNT(*) FROM kjv
    -> WHERE vtext LIKE '%still small voice%';
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
```

Однако поиск по шаблону SQL обычно работает медленнее, чем FULLTEXT-поиск. Похоже, вы оказались перед неприятным выбором: использовать быстрый способ, не выводящий желаемых результатов, или же корректно работающий, но медленный способ. К счастью, есть еще вариант: вы можете объединить оба способа в одном запросе:

```
mysql> SELECT COUNT(*) FROM kjv
    -> WHERE MATCH(vtext) AGAINST('still small voice')
    -> AND vtext LIKE '%still small voice%';
```

```
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
```

Берем лучшее из каждого способа:

- С помощью выражения `MATCH()` MySQL может выполнить `FULLTEXT`-поиск для формирования множества строк-кандидатов, содержащих слова из фразы. Тем самым значительно сужается круг поиска.
- Используя сравнение с шаблоном SQL, MySQL просматривает строки-кандидаты для вывода тех строк, в которых слова расположены в нужном порядке.

Данный прием не сработает, если все слова короче минимума, указанного для индексирования, или если слова встречаются более чем в половине записей. В подобных случаях `FULLTEXT`-поиск не вернет ни одной строки, но вы все еще можете выполнить поиск по шаблону SQL.

5

Работа с датами и временем

5.0. Введение

В MySQL есть несколько типов данных для представления значений дат и времени, а также ряд функций для работы с ними. MySQL хранит даты и время в специальном формате. Необходимо знать эти форматы, чтобы не удивляться тому, как MySQL интерпретирует вводимые даты. MySQL предоставляет и функции форматирования, с помощью которых можно вывести дату и время в формате, отличном от формата по умолчанию. Данная глава охватывает следующие аспекты работы со значениями времени в MySQL:

Вывод дат и времени. MySQL по умолчанию выводит значения дат и времени в определенном формате, но вы можете задать другие форматы при помощи вызова соответствующих функций.

Определение текущих даты и времени. MySQL предоставляет функции, возвращающие дату и время, что полезно для приложений, которым необходима такая информация или которые вычисляют другие значения времени на основе этой информации.

Разложение дат и времени на составляющие. В этом разделе рассказано о том, как разбивать значения дат и времени на части, если вам нужна только какая-то определенная составляющая, например, месяц или час.

Формирование дат и времени из составляющих. Дополнением разбиения значений времени на части является создание таких значений из частей. В данном разделе описано, как это делается.

Преобразование дат и времени в базовые единицы. Некоторые операции с датами проще выполнять, используя количество дней или секунд, представленное датой или временем, а не саму дату или время. MySQL разрешает различные виды преобразований значений дат и времени в более простые единицы, такие как дни или секунды. Такие преобразования удобны для вычисления интервалов (промежутков времени между двумя указанными значениями).

Арифметические операции с датами и временем. В MySQL можно добавлять к значениям даты и времени временные интервалы, а также вычислять интервалы между значениями даты или времени. Арифметические операции со значениями времени проще, чем с датами. Время представлено часами, минутами и секундами – единицами фиксированной продолжительности. А в операциях с датами участвуют годы и месяцы, продолжительность которых может быть различной.

Использование арифметических операций с датами и временем. Здесь показано, как применять приемы, описанные в предыдущем разделе, для вычисления возраста, относительных дат, сдвига дат и определения високосных годов.

Выбор записей по временным условиям. Вычисления из предыдущих разделов, возвращающие значения, могут использоваться и в инструкциях `WHERE` для выборки записей по временным параметрам.

Использование временных меток (значений TIMESTAMP). Столбец типа `TIMESTAMP` имеет некоторые особенности, делающие его удобным для автоматической регистрации времени создания и изменения записи. В этом разделе рассказано о поведении столбцов типа `TIMESTAMP` и их использовании. Кроме того, обсуждаются возможности вывода значений `TIMESTAMP` в удобной для чтения форме.

В главе представлено множество функций MySQL для обработки значений дат и времени, но есть и другие. При желании ознакомиться с полным набором функций, обратитесь к справочному руководству по MySQL. Разнообразие функций позволяет выполнить одно и то же действие несколькими способами. Иногда я буду приводить несколько вариантов получения желаемого результата, но охватить все способы решения задач, предложенных в данной главе, невозможно. Попробуйте сами поэкспериментировать, чтобы найти другие решения. Вполне возможно, что вам удастся найти более эффективный или более удобочитаемый метод.

Сценарии рецептов главы хранятся в каталоге *dates* дистрибутива исходных текстов `recipes`. Сценарии, создающие используемые таблицы, находятся в каталоге *tables*.

Форматы даты и времени MySQL

В MySQL имеются столбцы типов `DATE` и `TIME` для отдельного представления даты и времени, а также типа `DATETIME` для комбинированных значений дата-и-время. Значения могут быть представлены в следующих форматах:

- Значения типа `DATE` обрабатываются как строки в формате *CCYY-MM-DD*, где *CC*, *YY*, *MM* и *DD* представляют собой век, год века, месяц и день.
- Значения `TIME` представлены строками в формате *hh:mm:ss*, где *hh*, *mm* и *ss* – это часовая, минутная и секундная составляющие времени. Значения `TIME` часто воспринимаются как время суток, но на самом деле MySQL рассматривает их как продолжительность, так что они могут быть больше,

чем 23:59:59, и могут быть отрицательными (разрешенный диапазон значений от -838:59:59 до 838:59:59).

- Значения DATETIME представлены как комбинированные строки дата-и-время в формате *CCYY-MM-DD hh:mm:ss*.
- Значения TIMESTAMP также содержат части даты и времени, но в формате *CCYYMMDDhhmmss*. Этот тип столбца имеет специальные свойства, о которых будет рассказано в рецепте 5.31. В примерах главы чаще используются значения DATETIME, а не TIMESTAMP (которые менее удобны для восприятия), но в большинстве случаев эти два типа обрабатываются одинаково.

В примерах главы в основном используются следующие таблицы, содержащие столбцы со значениями типа TIME, DATE, DATETIME и TIMESTAMP. (Таблица time_val содержит два столбца, которые будут использоваться для вычисления интервалов времени.)

```
mysql> SELECT t1, t2 FROM time_val;
+----------+----------+
| t1       | t2       |
+----------+----------+
| 15:00:00 | 15:00:00 |
| 05:01:30 | 02:30:20 |
| 12:30:20 | 17:30:45 |
+----------+----------+
mysql> SELECT d FROM date_val;
+------------+
| d          |
+------------+
| 1864-02-28 |
| 1900-01-15 |
| 1987-03-05 |
| 1999-12-31 |
| 2000-06-04 |
+------------+
mysql> SELECT dt FROM datetime_val;
+---------------------+
| dt                  |
+---------------------+
| 1970-01-01 00:00:00 |
| 1987-03-05 12:30:15 |
| 1999-12-31 09:00:00 |
| 2000-06-04 15:45:30 |
+---------------------+
mysql> SELECT ts FROM timestamp_val;
+----------------+
| ts             |
+----------------+
| 19700101000000 |
| 19870305123015 |
| 19991231090000 |
| 20000604154530 |
+----------------+
```

5.1. Изменение формата даты MySQL

Задача

Вы хотите изменить формат, в котором MySQL представляет значения дат.

Решение

Это невозможно. Однако вы можете при сохранении данных преобразовать их в нужный формат, а при отображении привести их практически к любому виду, используя функцию `DATE_FORMAT()`.

Обсуждение

Формат *CCYY-MM-DD*, используемый MySQL для значений дат, соответствует стандарту ISO 8601 представления дат. Удобство этого формата в том, что год, месяц и день в нем имеют фиксированную длину и появляются в строке даты слева направо, так что даты естественным образом сортируются в правильном хронологическом порядке.[1] Но формат ISO используется не во всех системах баз данных, что может вызвать проблемы при перемещении данных из одной СУБД в другую. Кроме того, многие предпочитают другие форматы дат, такие как *MM/DD/YY* или *DD-MM-CCYY*. Это может вызывать проблемы, связанные с несоответствием ожиданий пользователей относительно формата хранения даты и реального поведения MySQL.

Новички в MySQL часто задают вопрос: «Как указать MySQL на необходимость хранить данные в определенном формате, например *MM/DD/CCYY*?» Ответом, к сожалению, будет: «Никак». MySQL всегда хранит даты в формате ISO, и последствия этого факта затрагивают как ввод данных, так и отображение результирующего множества:

- Для того чтобы при вводе данных сохранить значения, имеющие формат не-ISO, следует сначала преобразовать их. (Если вы не хотите этого делать, придется хранить их как строки, например в столбце `CHAR`. Но тогда вы не сможете работать с ними как с датами.) В некоторых случаях, если формат значений близок к ISO, преобразование может и не потребоваться. Например, возьмем строковые значения `87-1-7` и `1987-1-7` и числа `870107` и `19870107`. При загрузке в столбец `DATE` все они интерпретируются MySQL как дата `1987-01-07`. О преобразовании дат для ввода значений подробно рассказано в главе 10.
- Что касается отображения данных, можно представлять даты не в формате ISO, преобразовав их. Помочь в этом может функция MySQL `DATE_FORMAT()`, предоставляющая широкий выбор форматов вывода (см. рецепты 5.2 и 5.4). Вы также можете использовать такие функции, как `YEAR()`, для извлечения частей дат (см. рецепт 5.5). Дополнительная информация об изменении формата вывода приведена в главе 10, где также представлен

[1] Сортировка и группирование значений дат обсуждаются в главах 6 и 7.

небольшой сценарий, который выгружает содержимое таблицы с переформатированными столбцами дат.

5.2. Определение форматов отображения даты и времени

Задача

Вы хотите выводить дату или время не в том формате, который MySQL использует по умолчанию.

Решение

Используйте функции `DATE_FORMAT()` и `TIME_FORMAT()` для переформатирования значений.

Обсуждение

Как уже говорилось, MySQL отображает даты в формате ISO, если только вы не указали другой формат. Чтобы изменить формат значений дат, используйте функцию `DATE_FORMAT()`, которая принимает два аргумента: значение типа `DATE`, `DATETIME` или `TIMESTAMP` и строку, описывающую, как следует выводить значение. В строке форматирования применяются специальные последовательности: *%c*, где *c* показывает, какую часть даты необходимо отобразить. Например, `%Y`, `%M` и `%d` обозначают четырехзначный год, название месяца и двузначный день месяца. Следующий запрос выводит значения таблицы `date_val` в двух видах: в формате вывода по умолчанию MySQL и переформатированными при помощи `DATE_FORMAT()`:

```
mysql> SELECT d, DATE_FORMAT(d,'%M %d, %Y') FROM date_val;
+------------+---------------------------+
| d          | DATE_FORMAT(d,'%M %d, %Y') |
+------------+---------------------------+
| 1864-02-28 | February 28, 1864          |
| 1900-01-15 | January 15, 1900           |
| 1987-03-05 | March 05, 1987             |
| 1999-12-31 | December 31, 1999          |
| 2000-06-04 | June 04, 2000              |
+------------+---------------------------+
```

Функция `DATE_FORMAT()` формирует длинные заголовки столбцов, поэтому для вывода более точного и понятного названия разумно использовать псевдоним:

```
mysql> SELECT d, DATE_FORMAT(d,'%M %d, %Y') AS date FROM date_val;
+------------+-------------------+
| d          | date              |
+------------+-------------------+
| 1864-02-28 | February 28, 1864 |
| 1900-01-15 | January 15, 1900  |
```

```
| 1987-03-05 | March 05, 1987    |
| 1999-12-31 | December 31, 1999 |
| 2000-06-04 | June 04, 2000     |
+------------+-------------------+
```

Полный список форматирующих последовательностей приведен в справочном руководстве по MySQL, мы же рассмотрим лишь наиболее распространенные (табл. 5.1):

Таблица 5.1. Форматирование дат

Последовательность	Значение
%Y	Четырехзначный год
%y	Двузначный год
%M	Полное название месяца
%b	Три первые буквы названия месяца
%m	Двузначный месяц года (01..12)
%c	Месяц года (1..12)
%d	Двузначный день месяца (01..31)
%e	День месяца (1..31)
%r	Время в 12-часовом формате с суффиксом AM или PM
%T	Время в 24-часовом формате
%H	Двузначный час
%i	Двузначная минута
%s	Двузначная секунда
%%	Литерал %

Приведенные в табл. 5.1 последовательности форматирования времени удобны, только если вы передаете функции DATE_FORMAT() значение, включающее и дату, и время (тип DATETIME или TIMESTAMP). Следующий запрос показывает, как выводить значения DATETIME таблицы datetime_val, используя форматы, содержащие время дня:

```
mysql> SELECT dt,
    -> DATE_FORMAT(dt,'%c/%e/%y %r') AS format1,
    -> DATE_FORMAT(dt,'%M %e, %Y %T') AS format2
    -> FROM datetime_val;
+---------------------+----------------------+---------------------------+
| dt                  | format1              | format2                   |
+---------------------+----------------------+---------------------------+
| 1970-01-01 00:00:00 | 1/1/70 12:00:00 AM   | January 1, 1970 00:00:00  |
| 1987-03-05 12:30:15 | 3/5/87 12:30:15 PM   | March 5, 1987 12:30:15    |
| 1999-12-31 09:00:00 | 12/31/99 09:00:00 AM | December 31, 1999 09:00:00 |
| 2000-06-04 15:45:30 | 6/4/00 03:45:30 PM   | June 4, 2000 15:45:30     |
+---------------------+----------------------+---------------------------+
```

Функция TIME_FORMAT() похожа на DATE_FORMAT(), но воспринимает только спецификаторы форматирующей строки, относящиеся к времени. TIME_FORMAT() работает со значениями TIME, DATETIME и TIMESTAMP.

```
mysql> SELECT dt,
    -> TIME_FORMAT(dt, '%r') AS '12-hour time',
    -> TIME_FORMAT(dt, '%T') AS '24-hour time'
    -> FROM datetime_val;
+---------------------+--------------+--------------+
| dt                  | 12-hour time | 24-hour time |
+---------------------+--------------+--------------+
| 1970-01-01 00:00:00 | 12:00:00 AM  | 00:00:00     |
| 1987-03-05 12:30:15 | 12:30:15 PM  | 12:30:15     |
| 1999-12-31 09:00:00 | 09:00:00 AM  | 09:00:00     |
| 2000-06-04 15:45:30 | 03:45:30 PM  | 15:45:30     |
+---------------------+--------------+--------------+
```

5.3. Определение текущей даты или времени

Задача

Какой сегодня день? Который час?

Решение

Используйте функции NOW(), CURDATE() и CURTIME().

Обсуждение

Некоторым приложениям необходимо знать текущие дату и время, например, таким, которые выводят сведения о состоянии с пометками об их дате или времени. Такая информация полезна в вычислениях, зависящих от текущей даты, например, для получения первого или последнего дня месяца или определения того, каким числом месяца будет следующая среда.

Текущие значения даты и времени возвращают три функции: NOW() возвращает и текущую дату, и время; CURDATE() и CURTIME() возвращают дату и время по отдельности:

```
mysql> SELECT NOW(), CURDATE(), CURTIME();
+---------------------+------------+-----------+
| NOW()               | CURDATE()  | CURTIME() |
+---------------------+------------+-----------+
| 2002-07-15 10:59:30 | 2002-07-15 | 10:59:30  |
+---------------------+------------+-----------+
```

Функции CURRENT_TIMESTAMP и SYSDATE() являются синонимами NOW(). Функции CURRENT_DATE и CURRENT_TIME — синонимы CURDATE() и CURTIME().

Если вы хотите получить составляющие этих значений (например, текущий день месяца или текущий час дня), прочтите несколько следующих разделов.

Функцию NOW() нельзя использовать как значение столбца по умолчанию

Такие функции, как NOW() и CURDATE(), часто ошибочно используются в предложениях CREATE TABLE в качестве значений по умолчанию:

```
mysql> CREATE TABLE testtbl (dt DATETIME DEFAULT NOW());
You have an error in your SQL syntax near 'NOW())' at line 1
```

Здесь хотелось, чтобы значения столбца dt автоматически инициализировались в дату и время создания записей. Но ничего не получится; значения по умолчанию MySQL должны быть константами. Если вы хотите в момент создания записи установить столбец в текущую дату и время, используйте тип TIMESTAMP, автоматически инициализируемый MySQL, или тип DATETIME, для которого самостоятельно устанавливайте исходные значения при создании записей.

В будущем, с появлением MySQL 4.1 необходимость использовать константы в качестве значений по умолчанию будет устранена.

5.4. Разбиение дат и времени на части с помощью функций форматирования

Задача

Вы хотите получить только часть даты или времени.

Решение

Используйте функцию форматирования, такую как DATE_FORMAT() или TIME_FORMAT(), со строкой форматирования, содержащей спецификатор для той части значения, которую вы хотите получить.

Обсуждение

MySQL предоставляет ряд возможностей для разбиения значений дат и времени на составляющие. Функции DATE_FORMAT() и TIME_FORMAT() обеспечивают один из способов извлечения отдельных частей значений времени:

```
mysql> SELECT dt,
    -> DATE_FORMAT(dt,'%Y') AS year,
    -> DATE_FORMAT(dt,'%d') AS day,
    -> TIME_FORMAT(dt,'%H') AS hour,
    -> TIME_FORMAT(dt,'%s') AS second
    -> FROM datetime_val;
```

```
+---------------------+------+------+------+--------+
| dt                  | year | day  | hour | second |
+---------------------+------+------+------+--------+
| 1970-01-01 00:00:00 | 1970 | 01   | 00   | 00     |
| 1987-03-05 12:30:15 | 1987 | 05   | 12   | 15     |
| 1999-12-31 09:00:00 | 1999 | 31   | 09   | 00     |
| 2000-06-04 15:45:30 | 2000 | 04   | 15   | 30     |
+---------------------+------+------+------+--------+
```

Функции форматирования позволяют извлекать не только одну составляющую значения, но и несколько. Например, чтобы получить всю дату или время из значений DATETIME, выполните такое предложение:

```
mysql> SELECT dt,
    -> DATE_FORMAT(dt,'%Y-%m-%d') AS 'date part',
    -> TIME_FORMAT(dt,'%T') AS 'time part'
    -> FROM datetime_val;
+---------------------+------------+-----------+
| dt                  | date part  | time part |
+---------------------+------------+-----------+
| 1970-01-01 00:00:00 | 1970-01-01 | 00:00:00  |
| 1987-03-05 12:30:15 | 1987-03-05 | 12:30:15  |
| 1999-12-31 09:00:00 | 1999-12-31 | 09:00:00  |
| 2000-06-04 15:45:30 | 2000-06-04 | 15:45:30  |
+---------------------+------------+-----------+
```

Преимуществом использования форматирующих функций является то, что вы можете отобразить извлеченные значения не в том виде, в котором они были представлены в исходных значениях. Если вы хотите представить дату в формате, отличном от *CCYY-MM-DD*, или, например, время – без секунд, это легко сделать:

```
mysql> SELECT ts,
    -> DATE_FORMAT(ts,'%M %e, %Y') AS 'descriptive date',
    -> TIME_FORMAT(ts,'%H:%i') AS 'hours/minutes'
    -> FROM timestamp_val;
+----------------+-------------------+---------------+
| ts             | descriptive date  | hours/minutes |
+----------------+-------------------+---------------+
| 19700101000000 | January 1, 1970   | 00:00         |
| 19870305123015 | March 5, 1987     | 12:30         |
| 19991231090000 | December 31, 1999 | 09:00         |
| 20000604154530 | June 4, 2000      | 15:45         |
+----------------+-------------------+---------------+
```

См. также

В рецепте 5.5 описаны другие функции, применяемые для извлечения отдельных составляющих дат и времени. Рецепт 5.6 показывает, как использовать строковые функции для извлечения составляющих.

5.5. Разбиение дат и времени с помощью функций извлечения составляющих

Задача

Вы хотите получить только часть даты или времени.

Решение

Вызовите функцию, специально предназначенную для извлечения части значения времени, такую как MONTH() или MINUTE(). Для получения отдельных составляющих значений времени эффективнее использовать эти функции, а не DATE_FORMAT().

Обсуждение

MySQL содержит множество функций извлечения частей даты и времени из значений времени. Некоторые из них перечислены в табл. 5.2, если же вам нужен полный список, обратитесь к справочному руководству по MySQL. Функции дат работают со значениями DATE, DATETIME и TIMESTAMP, а функции времени – со значениями TIME, DATETIME и TIMESTAMP.

Таблица 5.2. Функции извлечения составляющих дат и времени

Функция	Возвращаемое значение
YEAR()	Год даты
MONTH()	Номер месяца (1..12)
MONTHNAME()	Название месяца (January..December)
DAYOFMONTH()	День месяца (1..31)
DAYNAME()	День недели (Sunday..Saturday)
DAYOFWEEK()	День недели (1..7 для Sunday..Saturday)
WEEKDAY()	День недели (0..6 для Monday..Sunday)
DAYOFYEAR()	День года (1..366)
HOUR()	Час времени (0..23)
MINUTE()	Минута времени (0..59)
SECOND()	Секунда времени (0..59)

Рассмотрим пример:

```
mysql> SELECT dt,
    -> YEAR(dt), DAYOFMONTH(dt),
    -> HOUR(dt), SECOND(dt)
    -> FROM datetime_val;
+---------------------+----------+----------------+----------+------------+
| dt                  | YEAR(dt) | DAYOFMONTH(dt) | HOUR(dt) | SECOND(dt) |
```

```
+---------------------+----------+----------------+----------+------------+
| 1970-01-01 00:00:00 |     1970 |              1 |        0 |          0 |
| 1987-03-05 12:30:15 |     1987 |              5 |       12 |         15 |
| 1999-12-31 09:00:00 |     1999 |             31 |        9 |          0 |
| 2000-06-04 15:45:30 |     2000 |              4 |       15 |         30 |
+---------------------+----------+----------------+----------+------------+
```

Такие функции, как YEAR() и DAYOFMONTH(), извлекают значения, имеющие очевидное соответствие подстроке значений дат. Некоторые другие функции извлекают значения, для которых нет такого четкого соответствия. Например, это относится к дню года:

```
mysql> SELECT d, DAYOFYEAR(d) FROM date_val;
+------------+--------------+
| d          | DAYOFYEAR(d) |
+------------+--------------+
| 1864-02-28 |           59 |
| 1900-01-15 |           15 |
| 1987-03-05 |           64 |
| 1999-12-31 |          365 |
| 2000-06-04 |          156 |
+------------+--------------+
```

А также к дню недели, который можно извлекать и как имя, и как номер:

- DAYNAME() возвращает полное название дня недели. Нет функции, возвращающей трехбуквенную аббревиатуру названия дня, но вы всегда можете передать его полное название в функцию LEFT():

```
mysql> SELECT d, DAYNAME(d), LEFT(DAYNAME(d),3) FROM date_val;
+------------+------------+--------------------+
| d          | DAYNAME(d) | LEFT(DAYNAME(d),3) |
+------------+------------+--------------------+
| 1864-02-28 | Sunday     | Sun                |
| 1900-01-15 | Monday     | Mon                |
| 1987-03-05 | Thursday   | Thu                |
| 1999-12-31 | Friday     | Fri                |
| 2000-06-04 | Sunday     | Sun                |
+------------+------------+--------------------+
```

- Чтобы получить номер дня недели, используйте функцию DAYOFWEEK() или WEEKDAY(), но обратите внимание на диапазон значений, возвращаемый каждой из функций. DAYOFWEEK() возвращает значения от 1 до 7, нумерация ведется от воскресенья (Sunday) до субботы (Saturday). WEEKDAY() возвращает значения от 0 до 6, соответствующие диапазону от понедельника (Monday) до воскресенья (Sunday):

```
mysql> SELECT d, DAYNAME(d), DAYOFWEEK(d), WEEKDAY(d) FROM date_val;
+------------+------------+--------------+------------+
| d          | DAYNAME(d) | DAYOFWEEK(d) | WEEKDAY(d) |
+------------+------------+--------------+------------+
| 1864-02-28 | Sunday     |            1 |          6 |
| 1900-01-15 | Monday     |            2 |          0 |
```

```
| 1987-03-05 | Thursday   |            5 |          3 |
| 1999-12-31 | Friday     |            6 |          4 |
| 2000-06-04 | Sunday     |            1 |          6 |
+------------+------------+--------------+------------+
```

Для получения отдельных частей значений времени можно также использовать функцию EXTRACT():

```
mysql> SELECT dt,
    -> EXTRACT(DAY FROM dt),
    -> EXTRACT(HOUR FROM dt)
    -> FROM datetime_val;
+---------------------+----------------------+-----------------------+
| dt                  | EXTRACT(DAY FROM dt) | EXTRACT(HOUR FROM dt) |
+---------------------+----------------------+-----------------------+
| 1970-01-01 00:00:00 |                    1 |                     0 |
| 1987-03-05 12:30:15 |                    5 |                    12 |
| 1999-12-31 09:00:00 |                   31 |                     9 |
| 2000-06-04 15:45:30 |                    4 |                    15 |
+---------------------+----------------------+-----------------------+
```

Ключевое слово, показывающее, какую именно часть значения следует извлечь, – это спецификаторы YEAR, MONTH, DAY, HOUR, MINUTE или SECOND. Функция EXTRACT() доступна начиная с версии MySQL 3.23.0.

Получение текущего года, месяца, дня, часа, минуты или секунды

Функции извлечения, представленные в разделе, можно применять к CURDATE() или NOW() для получения текущего года, месяца, дня или дня недели:

```
mysql> SELECT CURDATE(), YEAR(CURDATE()) AS year,
    -> MONTH(CURDATE()) AS month, MONTHNAME(CURDATE()) AS monthname,
    -> DAYOFMONTH(CURDATE()) AS day, DAYNAME(CURDATE()) AS dayname;
+------------+------+-------+-----------+------+---------+
| CURDATE()  | year | month | monthname | day  | dayname |
+------------+------+-------+-----------+------+---------+
| 2002-07-15 | 2002 |     7 | July      |   15 | Monday  |
+------------+------+-------+-----------+------+---------+
```

Аналогично можно получить текущий час, минуту и секунду, передав CURTIME() или NOW() в функцию извлечения составляющих времени:

```
mysql> SELECT NOW(), HOUR(NOW()) AS hour,
    -> MINUTE(NOW()) AS minute, SECOND(NOW()) AS second;
+---------------------+------+--------+--------+
| NOW()               | hour | minute | second |
+---------------------+------+--------+--------+
| 2002-07-15 11:21:12 |   11 |     21 |     12 |
+---------------------+------+--------+--------+
```

См. также

Изученные функции обеспечивают извлечение отдельных составляющих значений времени. Если вы хотите получить значение, состоящее из нескольких компонентов исходного значения, удобнее применять DATE_FORMAT() (см. рецепт 5.4).

5.6. Разбиение дат и времени с помощью строковых функций

Задача

Вы хотите получить только часть даты или времени.

Решение

Рассматривайте значение времени как строку и используйте такие функции, как LEFT() и MID(), для извлечения подстрок, соответствующих интересующей части значения.

Обсуждение

В рецептах 5.4 и 5.5 рассказывалось о том, как извлекать составляющие значений времени при помощи функции DATE_FORMAT() и функций типа YEAR() и MONTH(). Если вы передаете строковой функции дату или время, то MySQL интерпретирует их как строку, то есть у вас появляется возможность извлечения подстрок. Поэтому для извлечения частей значений времени можно использовать такие строковые функции, как LEFT() и MID().

```
mysql> SELECT dt,
    -> LEFT(dt,4) AS year,
    -> MID(dt,9,2) AS day,
    -> RIGHT(dt,2) AS second
    -> FROM datetime_val;
+---------------------+------+------+--------+
| dt                  | year | day  | second |
+---------------------+------+------+--------+
| 1970-01-01 00:00:00 | 1970 | 01   | 00     |
| 1987-03-05 12:30:15 | 1987 | 05   | 15     |
| 1999-12-31 09:00:00 | 1999 | 31   | 00     |
| 2000-06-04 15:45:30 | 2000 | 04   | 30     |
+---------------------+------+------+--------+
```

Вы можете выделить из значения DATETIME все составляющие времени или даты, используя функции извлечения строк LEFT() или RIGHT():

```
mysql> SELECT dt,
    -> LEFT(dt,10) AS date,
    -> RIGHT(dt,8) AS time
    -> FROM datetime_val;
```

```
+---------------------+------------+----------+
| dt                  | date       | time     |
+---------------------+------------+----------+
| 1970-01-01 00:00:00 | 1970-01-01 | 00:00:00 |
| 1987-03-05 12:30:15 | 1987-03-05 | 12:30:15 |
| 1999-12-31 09:00:00 | 1999-12-31 | 09:00:00 |
| 2000-06-04 15:45:30 | 2000-06-04 | 15:45:30 |
+---------------------+------------+----------+
```

Аналогично можно обрабатывать и значения TIMESTAMP. Однако поскольку они не содержат символов-разделителей, индексы для LEFT() или RIGHT() несколько отличаются (как и форматы вывода значений):

```
mysql> SELECT ts,
    -> LEFT(ts,8) AS date,
    -> RIGHT(ts,6) AS time
    -> FROM timestamp_val;
+----------------+----------+--------+
| ts             | date     | time   |
+----------------+----------+--------+
| 19700101000000 | 19700101 | 000000 |
| 19870305123015 | 19870305 | 123015 |
| 19991231090000 | 19991231 | 090000 |
| 20000604154530 | 20000604 | 154530 |
+----------------+----------+--------+
```

При использовании строковых функций для разбиения значений времени на составляющие необходимо помнить о двух ограничениях (которыми не связаны функции извлечения составляющих и форматирующие функции):

- Для того чтобы использовать такие функции, как LEFT(), MID() или RIGHT(), необходимо, чтобы строки имели фиксированную длину. MySQL может интерпретировать значение 1987-1-1 как 1987-01-01, если вы вставите его в столбец DATE. Но если для извлечения дня использовать функцию RIGHT('1987-1-1',2), то ничего не получится. Если значения содержат подстроки переменной длины, можно использовать SUBSTRING_INDEX(). Если значения близки к формату ISO, вы можете стандартизировать их, используя приемы, описанные в рецепте 5.18.
- Строковые функции нельзя использовать для получения значений, которые не соответствуют подстроке значения даты, например дня недели или дня года.

5.7. Синтез дат и времени с помощью функций форматирования

Задача

Вы хотите получить из имеющейся даты новую, заменив часть значений исходной даты.

Решение

Используйте функцию DATE_FORMAT() или TIME_FORMAT() для объединения частей существующих значений с теми частями, которые вы хотите заменить.

Обсуждение

Операцией, дополняющей разбиение значений дат и времени на части, является синтез значения из составляющих. Синтез дат и времени осуществляется при помощи функций форматирования (описано в данном рецепте) и объединения строк (описано в рецепте 5.8).

Значение даты часто формируется на основе имеющегося с заменой соответствующих частей. Например, чтобы найти первый день месяца, к которому относится дата, используйте DATE_FORMAT() для извлечения года и месяца из имеющейся даты и объедините их со значением дня 01:

```
mysql> SELECT d, DATE_FORMAT(d,'%Y-%m-01') FROM date_val;
+------------+--------------------------+
| d          | DATE_FORMAT(d,'%Y-%m-01') |
+------------+--------------------------+
| 1864-02-28 | 1864-02-01               |
| 1900-01-15 | 1900-01-01               |
| 1987-03-05 | 1987-03-01               |
| 1999-12-31 | 1999-12-01               |
| 2000-06-04 | 2000-06-01               |
+------------+--------------------------+
```

Функцию TIME_FORMAT() можно использовать аналогично:

```
mysql> SELECT t1, TIME_FORMAT(t1,'%H:%i:00') FROM time_val;
+----------+---------------------------+
| t1       | TIME_FORMAT(t1,'%H:%i:00') |
+----------+---------------------------+
| 15:00:00 | 15:00:00                  |
| 05:01:30 | 05:01:00                  |
| 12:30:20 | 12:30:00                  |
+----------+---------------------------+
```

5.8. Синтез дат и времени с помощью функций извлечения составляющих

Задача

У вас есть части даты или времени, и вы хотите объединить их для получения значения даты или времени.

Решение

Соедините части, используя функцию CONCAT().

Обсуждение

В качестве еще одного способа конструирования значений времени можно предложить использование функций извлечения частей дат в сочетании с CONCAT(). Но этот метод не столь удачен, как описанная в рецепте 5.7 техника использования DATE_FORMAT(), и иногда выводит несколько отличающиеся результаты:

```
mysql> SELECT d,
    -> CONCAT(YEAR(d),'-',MONTH(d),'-01')
    -> FROM date_val;
+------------+-----------------------------------+
| d          | CONCAT(YEAR(d),'-',MONTH(d),'-01') |
+------------+-----------------------------------+
| 1864-02-28 | 1864-2-01                         |
| 1900-01-15 | 1900-1-01                         |
| 1987-03-05 | 1987-3-01                         |
| 1999-12-31 | 1999-12-01                        |
| 2000-06-04 | 2000-6-01                         |
+------------+-----------------------------------+
```

Обратите внимание на то, что значения месяцев в некоторых датах состоят из одной цифры. Чтобы обеспечить двузначность значений, требуемую форматом ISO, используйте LPAD() при необходимости добавления к значению начального нуля:

```
mysql> SELECT d,
    -> CONCAT(YEAR(d),'-',LPAD(MONTH(d),2,'0'),'-01')
    -> FROM date_val;
+------------+-------------------------------------------------+
| d          | CONCAT(YEAR(d),'-',LPAD(MONTH(d),2,'0'),'-01') |
+------------+-------------------------------------------------+
| 1864-02-28 | 1864-02-01                                      |
| 1900-01-15 | 1900-01-01                                      |
| 1987-03-05 | 1987-03-01                                      |
| 1999-12-31 | 1999-12-01                                      |
| 2000-06-04 | 2000-06-01                                      |
+------------+-------------------------------------------------+
```

Другой способ решения проблемы представлен в рецепте 5.18.

Значения TIME могут формироваться из значений часов, минут, секунд при помощи тех же методов, которые используются для создания значений DATE. Например, для того чтобы изменить значение TIME так, чтобы часть секунд стала равна 00, извлеките части минут и часов, а затем воссоедините их при помощи функций TIME_FORMAT() и CONCAT():

```
mysql> SELECT t1,
    -> TIME_FORMAT(t1,'%H:%i:00') AS method1,
    -> CONCAT(LPAD(HOUR(t1),2,'0'),':',LPAD(MINUTE(t1),2,'0'),':00') AS method2
    -> FROM time_val;
```

```
+----------+----------+----------+
| t1       | method1  | method2  |
+----------+----------+----------+
| 15:00:00 | 15:00:00 | 15:00:00 |
| 05:01:30 | 05:01:00 | 05:01:00 |
| 12:30:20 | 12:30:00 | 12:30:00 |
+----------+----------+----------+
```

5.9. Объединение даты и времени в значение дата-и-время

Задача

Вы хотите сконструировать объединенное значение дата-и-время из имеющихся отдельных значений даты и времени.

Решение

Соедините значения, поставив между ними пробел.

Обсуждение

Для объединения значения даты со значением времени для формирования значения дата-и-время просто соедините значения, добавив между ними пробел:

```
mysql> SET @d = '2002-02-28';
mysql> SET @t = '13:10:05';
mysql> SELECT @d, @t, CONCAT(@d,' ',@t);
+------------+----------+---------------------+
| @d         | @t       | CONCAT(@d,' ',@t)   |
+------------+----------+---------------------+
| 2002-02-28 | 13:10:05 | 2002-02-28 13:10:05 |
+------------+----------+---------------------+
```

5.10. Преобразование времени в секунды и обратно

Задача

У вас есть значение времени, которое вы хотите пересчитать в секунды, или наоборот.

Решение

Значения `TIME` являются специальным представлением простейших единиц времени – секунд, и у вас есть возможность преобразовывать одни в другие и обратно, используя функции `TIME_TO_SEC()` и `SEC_TO_TIME()`.

Обсуждение

Функция TIME_TO_SEC() преобразует значение TIME в соответствующее количество секунд, а SEC_TO_TIME() делает обратное. Следующий запрос иллюстрирует простое преобразование в обоих направлениях:

```
mysql> SELECT t1,
    -> TIME_TO_SEC(t1) AS 'TIME to seconds',
    -> SEC_TO_TIME(TIME_TO_SEC(t1)) AS 'TIME to seconds to TIME'
    -> FROM time_val;
+----------+-----------------+-------------------------+
| t1       | TIME to seconds | TIME to seconds to TIME |
+----------+-----------------+-------------------------+
| 15:00:00 |           54000 | 15:00:00                |
| 05:01:30 |           18090 | 05:01:30                |
| 12:30:20 |           45020 | 12:30:20                |
+----------+-----------------+-------------------------+
```

Чтобы выразить значения времени в минутах, часах или днях, выполните необходимое деление:

```
mysql> SELECT t1,
    -> TIME_TO_SEC(t1) AS 'seconds',
    -> TIME_TO_SEC(t1)/60 AS 'minutes',
    -> TIME_TO_SEC(t1)/(60*60) AS 'hours',
    -> TIME_TO_SEC(t1)/(24*60*60) AS 'days'
    -> FROM time_val;
+----------+---------+---------+-------+------+
| t1       | seconds | minutes | hours | days |
+----------+---------+---------+-------+------+
| 15:00:00 |   54000 |  900.00 | 15.00 | 0.62 |
| 05:01:30 |   18090 |  301.50 |  5.03 | 0.21 |
| 12:30:20 |   45020 |  750.33 | 12.51 | 0.52 |
+----------+---------+---------+-------+------+
```

Если вы предпочитаете целые числа, а не значения с плавающей точкой, используйте функцию FLOOR():

```
mysql> SELECT t1,
    -> TIME_TO_SEC(t1) AS 'seconds',
    -> FLOOR(TIME_TO_SEC(t1)/60) AS 'minutes',
    -> FLOOR(TIME_TO_SEC(t1)/(60*60)) AS 'hours',
    -> FLOOR(TIME_TO_SEC(t1)/(24*60*60)) AS 'days'
    -> FROM time_val;
+----------+---------+---------+-------+------+
| t1       | seconds | minutes | hours | days |
+----------+---------+---------+-------+------+
| 15:00:00 |   54000 |     900 |    15 |    0 |
| 05:01:30 |   18090 |     301 |     5 |    0 |
| 12:30:20 |   45020 |     750 |    12 |    0 |
+----------+---------+---------+-------+------+
```

Если вы передаете функции TIME_TO_SEC() значение дата-и-время, то извлекается часть времени, а дата отбрасывается. Тем самым обеспечивается еще

один способ (в дополнение к описанным ранее в этой главе) извлечения времени из значений DATETIME и TIMESTAMP:

```
mysql> SELECT dt,
    -> TIME_TO_SEC(dt) AS 'time part in seconds',
    -> SEC_TO_TIME(TIME_TO_SEC(dt)) AS 'time part as TIME'
    -> FROM datetime_val;
+---------------------+----------------------+-------------------+
| dt                  | time part in seconds | time part as TIME |
+---------------------+----------------------+-------------------+
| 1970-01-01 00:00:00 |                    0 | 00:00:00          |
| 1987-03-05 12:30:15 |                45015 | 12:30:15          |
| 1999-12-31 09:00:00 |                32400 | 09:00:00          |
| 2000-06-04 15:45:30 |                56730 | 15:45:30          |
+---------------------+----------------------+-------------------+
mysql> SELECT ts,
    -> TIME_TO_SEC(ts) AS 'time part in seconds',
    -> SEC_TO_TIME(TIME_TO_SEC(ts)) AS 'time part as TIME'
    -> FROM timestamp_val;
+----------------+----------------------+-------------------+
| ts             | time part in seconds | time part as TIME |
+----------------+----------------------+-------------------+
| 19700101000000 |                    0 | 00:00:00          |
| 19870305123015 |                45015 | 12:30:15          |
| 19991231090000 |                32400 | 09:00:00          |
| 20000604154530 |                56730 | 15:45:30          |
+----------------+----------------------+-------------------+
```

5.11. Преобразование дат в дни и обратно

Задача

У вас есть дата, а хотелось бы получить значение в днях, или наоборот.

Решение

Значения DATE можно преобразовывать в дни и получать из них при помощи функций TO_DAYS() и FROM_DAYS(). Значения дата-и-время также можно преобразовывать в дни, если вас не пугает потеря составляющей времени.

Обсуждение

Функция TO_DAYS() преобразует дату в соответствующее количество дней, а FROM_DAYS() выполняет обратную операцию:

```
mysql> SELECT d,
    -> TO_DAYS(d) AS 'DATE to days',
    -> FROM_DAYS(TO_DAYS(d)) AS 'DATE to days to DATE'
    -> FROM date_val;
+------------+--------------+----------------------+
| d          | DATE to days | DATE to days to DATE |
+------------+--------------+----------------------+
```

```
| 1864-02-28 |       680870 | 1864-02-28          |
| 1900-01-15 |       693975 | 1900-01-15          |
| 1987-03-05 |       725800 | 1987-03-05          |
| 1999-12-31 |       730484 | 1999-12-31          |
| 2000-06-04 |       730640 | 2000-06-04          |
+------------+--------------+---------------------+
```

Если вы используете TO_DAYS(), следует прислушаться к рекомендации руководства по MySQL и избегать значений DATE, предшествующих началу летоисчисления по григорианскому календарю (1582). Изменения длин календарных годов и месяцев, имевшие место до этой даты, вызывают сложности с объяснением того, что такое «день 0». Что же касается TIME_TO_SEC(), в данном случае соответствие значения TIME результирующему значению в секундах является очевидным и имеет осмысленную контрольную точку в 0 секунд.

Если вы передаете функции TO_DAYS() значение дата-и-время, то время отбрасывается, и извлекается дата. Этот способ можно применять для извлечения дат из значений типа DATETIME и TIMESTAMP:

```
mysql> SELECT dt,
    -> TO_DAYS(dt) AS 'date part in days',
    -> FROM_DAYS(TO_DAYS(dt)) AS 'date part as DATE'
    -> FROM datetime_val;
+---------------------+------------------+------------------+
| dt                  | date part in days | date part as DATE |
+---------------------+------------------+------------------+
| 1970-01-01 00:00:00 |            719528 | 1970-01-01        |
| 1987-03-05 12:30:15 |            725800 | 1987-03-05        |
| 1999-12-31 09:00:00 |            730484 | 1999-12-31        |
| 2000-06-04 15:45:30 |            730640 | 2000-06-04        |
+---------------------+------------------+------------------+
mysql> SELECT ts,
    -> TO_DAYS(ts) AS 'date part in days',
    -> FROM_DAYS(TO_DAYS(ts)) AS 'date part as DATE'
    -> FROM timestamp_val;
+----------------+------------------+------------------+
| ts             | date part in days | date part as DATE |
+----------------+------------------+------------------+
| 19700101000000 |            719528 | 1970-01-01        |
| 19870305123015 |            725800 | 1987-03-05        |
| 19991231090000 |            730484 | 1999-12-31        |
| 20000604154530 |            730640 | 2000-06-04        |
+----------------+------------------+------------------+
```

5.12. Преобразование значений дата-и-время в секунды и обратно

Задача

У вас есть значение дата-и-время, а требуется значение в секундах, или наоборот.

Решение

Функции UNIX_TIMESTAMP() и FROM_UNIXTIME() преобразуют значения DATETIME и TIMESTAMP, начиная с 1970 года и заканчивая приблизительно 2037 годом, в количество секунд, прошедших с начала 1970 года, и обратно. Преобразование в секунды делает значения более точными, чем при преобразовании в дни, но ценой более узкого диапазона исходных значений.

Обсуждение

При работе со значениями дата-и-время вы можете использовать функции TO_DAYS() и FROM_DAYS() для преобразования дат в дни и наоборот, как было показано в предыдущем разделе. Для дат, не относящихся к периоду, предшествующему 1970-01-01 00:00:00 по Гринвичу (GMT), и не более поздних, чем приблизительно 2037 год[1], предоставлена возможность достижения более высокой точности за счет преобразования в секунды и обратно. Функция UNIX_TIMESTAMP() преобразует значения дата-и-время в количество секунд, прошедших с начала 1970 года, а FROM_UNIXTIME() выполняет обратную операцию:

```
mysql> SELECT dt,
    -> UNIX_TIMESTAMP(dt) AS seconds,
    -> FROM_UNIXTIME(UNIX_TIMESTAMP(dt)) AS timestamp
    -> FROM datetime_val;
+---------------------+-----------+---------------------+
| dt                  | seconds   | timestamp           |
+---------------------+-----------+---------------------+
| 1970-01-01 00:00:00 |     21600 | 1970-01-01 00:00:00 |
| 1987-03-05 12:30:15 | 541967415 | 1987-03-05 12:30:15 |
| 1999-12-31 09:00:00 | 946652400 | 1999-12-31 09:00:00 |
| 2000-06-04 15:45:30 | 960151530 | 2000-06-04 15:45:30 |
+---------------------+-----------+---------------------+
```

Упоминание о UNIX в названиях функций и отсчет значений от 1970 года объясняется тем, что 1970-01-01 00:00:00 по Гринвичу – это начало «эпохи UNIX», точка отсчета для измерений времени в UNIX-системах.[2] Теперь, когда вы это знаете, может показаться странным то, что в предыдущем примере UNIX_TIMESTAMP() выводит значение 21600 для первого значения таблицы datetime_val. В чем дело? Почему не 0? Это противоречие – только кажущееся. Дело в том, что сервер MySQL при отображении значений преобразует их к своему часовому поясу. Мой сервер работает в часовом поясе U.S. Central Time, который на шесть часов (то есть на 21600 секунд) западнее Гринвича.

Функция UNIX_TIMESTAMP() может преобразовывать значения DATE в секунды, при этом они трактуются как содержащие неявную часть времени 00:00:00:

```
mysql> SELECT CURDATE(), FROM_UNIXTIME(UNIX_TIMESTAMP(CURDATE()));
```

[1] Сложно указать точный верхний предел значений, так как он зависит от конкретной системы.

[2] 1970-01-01 00:00:00 по Гринвичу – это точка отсчета времени и в Java.

```
+------------+-----------------------------------------+
| CURDATE()  | FROM_UNIXTIME(UNIX_TIMESTAMP(CURDATE())) |
+------------+-----------------------------------------+
| 2002-07-15 | 2002-07-15 00:00:00                     |
+------------+-----------------------------------------+
```

5.13. Сложение значений времени

Задача

Вы хотите добавить к значению времени указанное количество секунд или сложить два значения времени.

Решение

Используйте функцию `TIME_TO_SEC()`, чтобы обеспечить представление всех значений в секундах, затем сложите их. Результат будет получен в секундах. Если вы хотите преобразовать его обратно в значение времени, используйте функцию `SEC_TO_TIME()`.

Обсуждение

Основными инструментами выполнения арифметических операций со значениями времени являются функции `TIME_TO_SEC()` и `SEC_TO_TIME()`, преобразующие значения `TIME` в секунды и обратно. Чтобы добавить значение интервала времени в секундах к значению `TIME`, преобразуйте `TIME` в секунды, чтобы оба значения были представлены в одинаковых единицах измерения, затем выполните сложение и преобразуйте результат обратно в `TIME`. Например, два часа – это 7200 секунд (2*60*60), так что следующий запрос добавляет два часа к каждому из значений `t1` таблицы `time_val`:

```
mysql> SELECT t1,
    -> SEC_TO_TIME(TIME_TO_SEC(t1) + 7200) AS 't1 plus 2 hours'
    -> FROM time_val;
+----------+-----------------+
| t1       | t1 plus 2 hours |
+----------+-----------------+
| 15:00:00 | 17:00:00        |
| 05:01:30 | 07:01:30        |
| 12:30:20 | 14:30:20        |
+----------+-----------------+
```

Если сам интервал представлен значением `TIME`, его тоже следует преобразовать в секунды перед сложением. Вычислим сумму двух значений `TIME` из таблицы `time_val`:

```
mysql> SELECT t1, t2,
    -> SEC_TO_TIME(TIME_TO_SEC(t1) + TIME_TO_SEC(t2)) AS 't1 + t2'
    -> FROM time_val;
+----------+----------+----------+
| t1       | t2       | t1 + t2  |
```

```
+----------+----------+----------+
| 15:00:00 | 15:00:00 | 30:00:00 |
| 05:01:30 | 02:30:20 | 07:31:50 |
| 12:30:20 | 17:30:45 | 30:01:05 |
+----------+----------+----------+
```

Необходимо помнить о том, что значения типа TIME в MySQL представляют собой реально прошедшее время, а не время дня, так что они не устанавливаются в 0 после достижения 24 часов. Это видно в первой и третьей строках вывода предыдущего запроса. Чтобы выводить значения времени дня, перед преобразованием секунд обратно в значение TIME выполните операцию деления по модулю. Количество секунд в дне равно 24*60*60, то есть 86 400, поэтому для преобразования любого значения секунд s к 24-часовому диапазону необходимо использовать функцию MOD() или оператор деления по модулю (%):

```
MOD(s,86400)
s % 86400
```

Эти два выражения эквивалентны. Применим первое из них в вычислении времени из предыдущего примера и получим такой результат:

```
mysql> SELECT t1, t2,
    -> SEC_TO_TIME(MOD(TIME_TO_SEC(t1) + TIME_TO_SEC(t2), 86400)) AS 't1 + t2'
    -> FROM time_val;
+----------+----------+----------+
| t1       | t2       | t1 + t2  |
+----------+----------+----------+
| 15:00:00 | 15:00:00 | 06:00:00 |
| 05:01:30 | 02:30:20 | 07:31:50 |
| 12:30:20 | 17:30:45 | 06:01:05 |
+----------+----------+----------+
```

Для значений типа TIME разрешен диапазон от -838:59:59 до 838:59:59 (то есть от -3 020 399 до 3 020 399 секунд). При сложении двух значений можно получить результат, не попадающий в данный диапазон. Если вы попытаетесь сохранить такое значение в столбце TIME, MySQL «обрежет» его до ближайшего граничного значения диапазона.

5.14. Вычисление интервалов между значениями времени

Задача

Вы хотите узнать, сколько времени прошло между двумя значениями времени.

Решение

Преобразуйте время в секунды с помощью функции TIME_TO_SEC() и вычислите разность. Если вам нужен интервал, представленный как время, выполните обратное преобразование, используя функцию SEC_TO_TIME().

Обсуждение

Вычисление интервала между значениями времени похоже на сложение значений времени, только в данном случае вычисляется не сумма, а разность. Например, чтобы вычислить интервал в секундах между парами значений t1 и t2, преобразуйте значения таблицы time_val в секунды при помощи TIME_TO_SEC(), а затем вычислите разность. Чтобы результатом было значение типа TIME, следует передать его в SEC_TO_TIME(). Следующий запрос выводит интервалы в двух представлениях:

```
mysql> SELECT t1, t2,
    -> TIME_TO_SEC(t2) - TIME_TO_SEC(t1) AS 'interval in seconds',
    -> SEC_TO_TIME(TIME_TO_SEC(t2) - TIME_TO_SEC(t1)) AS 'interval as TIME'
    -> FROM time_val;
+----------+----------+---------------------+------------------+
| t1       | t2       | interval in seconds | interval as TIME |
+----------+----------+---------------------+------------------+
| 15:00:00 | 15:00:00 |                   0 | 00:00:00         |
| 05:01:30 | 02:30:20 |               -9070 | -02:31:10        |
| 12:30:20 | 17:30:45 |               18025 | 05:00:25         |
+----------+----------+---------------------+------------------+
```

Обратите внимание на то, что интервалы могут быть и отрицательными, если время t1 наступило позже, чем t2.

5.15. Разбиение интервалов времени на составляющие

Задача

У вас есть интервал времени, представленный как значение времени, но вам хотелось бы разбить его на составляющие.

Решение

Для разбиения интервала на составляющие используйте функции HOUR(), MINUTE() и SECOND(). Если в SQL вычисление получается сложным, а интервал используется в программе, возможно, будет проще выполнить аналогичные вычисления на вашем языке программирования.

Обсуждение

Чтобы представить интервал времени в терминах составляющих его часов, минут и секунд, вычислите компоненты интервала в SQL с помощью функций HOUR(), MINUTE() и SECOND() (не забывайте о том, что интервалы могут быть отрицательными, и это необходимо учитывать). Например, чтобы определить составляющие интервалов между столбцами t1 и t2 таблицы time_val, можно было бы выполнить такое предложение SQL:

```
mysql> SELECT t1, t2,
    -> SEC_TO_TIME(TIME_TO_SEC(t2) - TIME_TO_SEC(t1)) AS 'interval as TIME',
    -> IF(SEC_TO_TIME(TIME_TO_SEC(t2) >= TIME_TO_SEC(t1)),'+','-') AS sign,
    -> HOUR(SEC_TO_TIME(TIME_TO_SEC(t2) - TIME_TO_SEC(t1))) AS hour,
    -> MINUTE(SEC_TO_TIME(TIME_TO_SEC(t2) - TIME_TO_SEC(t1))) AS minute,
    -> SECOND(SEC_TO_TIME(TIME_TO_SEC(t2) - TIME_TO_SEC(t1))) AS second
    -> FROM time_val;
+----------+----------+------------------+------+------+--------+--------+
| t1       | t2       | interval as TIME | sign | hour | minute | second |
+----------+----------+------------------+------+------+--------+--------+
| 15:00:00 | 15:00:00 | 00:00:00         | +    |    0 |      0 |      0 |
| 05:01:30 | 02:30:20 | -02:31:10        | -    |    2 |     31 |     10 |
| 12:30:20 | 17:30:45 | 05:00:25         | +    |    5 |      0 |     25 |
+----------+----------+------------------+------+------+--------+--------+
```

Выглядит беспорядочно и запутанно, а если попытаться сделать то же самое, используя операцию деления по модулю, получится еще хуже. Если запрос вычисления интервала выполняется из программы, всего этого беспорядка можно избежать. Используйте SQL только для вычисления интервалов в секундах, затем используйте язык API для разбиения каждого интервала на составляющие. Необходимо учитывать возможность отрицательного значения и выводить целое значение для каждого компонента. Рассмотрим функцию `time_components()`, написанную на языке Python, которая принимает значение интервала в секундах и возвращает четырехэлементный кортеж, содержащий знак, а также часы, минуты и секунды:

```
def time_components (time_in_secs):
    if time_in_secs < 0:
        sign = "-"
        time_in_secs = -time_in_secs
    else:
        sign = ""
    hours = int (time_in_secs / 3600)
    minutes = int ((time_in_secs / 60)) % 60
    seconds = time_in_secs % 60
    return (sign, hours, minutes, seconds)
```

Можно использовать функцию `time_components()` в программе так:

```
query = "SELECT t1, t2, TIME_TO_SEC(t2) - TIME_TO_SEC(t1) FROM time_val"
cursor = conn.cursor ()
cursor.execute (query)
for (t1, t2, interval) in cursor.fetchall ():
    (sign, hours, minutes, seconds) = time_components (interval)
    print "t1 = %s, t2 = %s, interval = %s%d h, %d m, %d s" \
                    % (t1, t2, sign, hours, minutes, seconds)
cursor.close ()
```

Эта программа формирует такой вывод:

```
t1 = 15:00:00, t2 = 15:00:00, interval = 0 h, 0 m, 0 s
t1 = 05:01:30, t2 = 02:30:20, interval = -2 h, 31 m, 10 s
t1 = 12:30:20, t2 = 17:30:45, interval = 5 h, 0 m, 25 s
```

Предыдущий пример является иллюстрацией общей идеи, которая часто полезна при создании запросов в программе: может оказаться, что проще не связываться со сложными вычислениями в SQL, а лишь выполнить простой запрос, а затем обрабатывать его результаты, используя средства языка API.

5.16. Добавление значения времени к дате

Задача

Вы хотите добавить время к значению даты или к значению дата-и-время.

Решение

Используйте функции DATE_ADD() и DATE_SUB(), специально предназначенные для выполнения арифметических операций с датами. Вы также можете применять TO_DAYS() и FROM_DAYS() или UNIX_TIMESTAMP() и FROM_UNIXTIME().

Обсуждение

Арифметические операции с датами чуть сложнее, чем со временем, из-за переменной длины месяцев и годов. Поэтому MySQL предлагает специальные функции DATE_ADD() и DATE_SUB() для сложения интервалов с датами и вычитания из них.[1] Функции принимают значение даты d и интервал в таком формате:

```
DATE_ADD(d,INTERVAL значение единица)
DATE_SUB(d,INTERVAL значение единица)
```

где *единица* – это единица измерения интервала, а *значение* – выражение, определяющее количество таких единиц. К наиболее распространенным относятся такие спецификаторы единиц, как YEAR, MONTH, DAY, HOUR, MINUTE и SECOND (полный список приведен в руководстве по MySQL). Обратите внимание на то, что единица измерения указывается не во множественном, а в единственном числе.

С помощью функций DATE_ADD() и DATE_SUB() можно выполнять следующие арифметические операции:

- Определить дату, которая наступит через три дня после сегодняшней:

```
mysql> SELECT CURDATE(), DATE_ADD(CURDATE(),INTERVAL 3 DAY);
+------------+-----------------------------------+
| CURDATE()  | DATE_ADD(CURDATE(),INTERVAL 3 DAY) |
+------------+-----------------------------------+
| 2002-07-15 | 2002-07-18                        |
+------------+-----------------------------------+
```

- Узнать дату, наступившую неделю назад (для представления недельного интервала запрос использует значение 7 DAY, так как не существует единица измерения WEEK):

[1] Функции DATE_ADD() и DATE_SUB() появились в MySQL 3.22.4, как и их синонимы ADDDATE() и SUBDATE().

```
mysql> SELECT CURDATE(), DATE_SUB(CURDATE(),INTERVAL 7 DAY);
+------------+-----------------------------------+
| CURDATE()  | DATE_SUB(CURDATE(),INTERVAL 7 DAY) |
+------------+-----------------------------------+
| 2002-07-15 | 2002-07-08                        |
+------------+-----------------------------------+
```

- Если вам необходимо узнать и дату, и время, начните со значения DATETIME или TIMESTAMP. Чтобы ответить на вопрос: «Сколько времени будет через 60 часов?», выполните такой запрос:

```
mysql> SELECT NOW(), DATE_ADD(NOW(),INTERVAL 60 HOUR);
+---------------------+----------------------------------+
| NOW()               | DATE_ADD(NOW(),INTERVAL 60 HOUR) |
+---------------------+----------------------------------+
| 2002-07-15 11:31:17 | 2002-07-17 23:31:17              |
+---------------------+----------------------------------+
```

- Некоторые спецификаторы интервалов включают и дату, и время. Следующий запрос добавляет к текущим дате и времени четырнадцать с половиной часов:

```
mysql> SELECT NOW(), DATE_ADD(NOW(),INTERVAL '14:30' HOUR_MINUTE);
+---------------------+-----------------------------------------------+
| NOW()               | DATE_ADD(NOW(),INTERVAL '14:30' HOUR_MINUTE) |
+---------------------+-----------------------------------------------+
| 2002-07-15 11:31:24 | 2002-07-16 02:01:24                           |
+---------------------+-----------------------------------------------+
```

  Аналогично можно добавить 3 дня и 4 часа:

```
mysql> SELECT NOW(), DATE_ADD(NOW(),INTERVAL '3 4' DAY_HOUR);
+---------------------+---------------------------------------+
| NOW()               | DATE_ADD(NOW(),INTERVAL '3 4' DAY_HOUR) |
+---------------------+---------------------------------------+
| 2002-07-15 11:31:30 | 2002-07-18 15:31:30                   |
+---------------------+---------------------------------------+
```

Функции DATE_ADD() и DATE_SUB() являются взаимозаменяемыми, так как одна из них – то же, что другая с интервалом противоположного знака. Например, два такие вызова эквиваленты для любого значения даты d:

```
DATE_ADD(d,INTERVAL -3 MONTH)
DATE_SUB(d,INTERVAL 3 MONTH)
```

Начиная с MySQL 3.23.4, можно использовать для сложения и вычитания интервалов дат операторы + и -:

```
mysql> SELECT CURDATE(), CURDATE() + INTERVAL 1 YEAR;
+------------+---------------------------+
| CURDATE()  | CURDATE() + INTERVAL 1 YEAR |
+------------+---------------------------+
| 2002-07-15 | 2003-07-15                |
+------------+---------------------------+
```

```
mysql> SELECT NOW(), NOW() - INTERVAL 24 HOUR;
+---------------------+-------------------------+
| NOW()               | NOW() - INTERVAL 24 HOUR |
+---------------------+-------------------------+
| 2002-07-15 11:31:48 | 2002-07-14 11:31:48      |
+---------------------+-------------------------+
```

Для добавления интервалов к дате или значению дата-и-время можно также использовать функции преобразования в базовые единицы и обратно. Например, чтобы сдвинуть дату вперед или назад на неделю (семь дней), используйте функции TO_DAYS() и FROM_DAYS():

```
mysql> SET @d = '2002-01-01';
mysql> SELECT @d AS date,
    -> FROM_DAYS(TO_DAYS(@d) + 7) AS 'date + 1 week',
    -> FROM_DAYS(TO_DAYS(@d) - 7) AS 'date - 1 week';
+------------+---------------+---------------+
| date       | date + 1 week | date - 1 week |
+------------+---------------+---------------+
| 2002-01-01 | 2002-01-08    | 2001-12-25    |
+------------+---------------+---------------+
```

Функция TO_DAYS() умеет вдобавок преобразовывать значения DATETIME и TIMESTAMP в дни, что удобно для тех, кому не нужна часть времени:

```
mysql> SET @dt = '2002-01-01 12:30:45';
mysql> SELECT @dt AS datetime,
    -> FROM_DAYS(TO_DAYS(@dt) + 7) AS 'datetime + 1 week',
    -> FROM_DAYS(TO_DAYS(@dt) - 7) AS 'datetime - 1 week';
+---------------------+-------------------+-------------------+
| datetime            | datetime + 1 week | datetime - 1 week |
+---------------------+-------------------+-------------------+
| 2002-01-01 12:30:45 | 2002-01-08        | 2001-12-25        |
+---------------------+-------------------+-------------------+
```

Для обеспечения точности работы со значениями DATETIME и TIMESTAMP используйте функции UNIX_TIMESTAMP() и FROM_UNIXTIME() вместо рассмотренных ранее. Изменим значение DATETIME на час (3600 секунд) вперед и назад:

```
mysql> SET @dt = '2002-01-01 09:00:00';
mysql> SELECT @dt AS datetime,
    -> FROM_UNIXTIME(UNIX_TIMESTAMP(@dt) + 3600) AS 'datetime + 1 hour',
    -> FROM_UNIXTIME(UNIX_TIMESTAMP(@dt) - 3600) AS 'datetime - 1 hour';
+---------------------+---------------------+---------------------+
| datetime            | datetime + 1 hour   | datetime - 1 hour   |
+---------------------+---------------------+---------------------+
| 2002-01-01 09:00:00 | 2002-01-01 10:00:00 | 2002-01-01 08:00:00 |
+---------------------+---------------------+---------------------+
```

Для реализации последнего приема необходимо, чтобы и исходное, и результирующее значения попадали в разрешенный для значений TIMESTAMP диапазон (с 1970 года где-то по 2037 год).

5.17. Вычисление интервалов между датами

Задача

Вы хотите узнать, на какое время отстоят две даты друг от друга.

Решение

Преобразуйте обе даты в базовые единицы и вычислите разность получившихся значений.

Обсуждение

Общая процедура вычисления интервала между датами заключается в преобразовании обеих дат к общей единице измерения и вычислении разности. Разрешенный диапазон значений определяет возможные виды преобразований. Значения `DATE`, `DATETIME` и `TIMESTAMP`, предшествующие `1970-01-01 00:00:00` по Гринвичу – точке отсчета UNIX, могут быть преобразованы в количество секунд, прошедших с момента начала отсчета. Если обе даты попадают в допустимый диапазон, интервал можно вычислить с точностью до секунды. Старые, но относящиеся к григорианскому календарю (1582) даты можно преобразовывать в дни, и интервалы будут вычисляться в днях. Даты же, предшествующие всем точкам отсчета, представляют некоторую проблему. В таких случаях на помощь может придти ваш язык программирования API – если окажется, что в нем возможны вычисления, которые нельзя или сложно выполнить в SQL. Если это так, подумайте об обработке значений дат непосредственно на вашем языке API. (Например, в библиотеке CPAN доступны модули *Date::Calc* и *Date::Manip*, которые можно использовать в сценариях Perl.)

Чтобы вычислить интервал в днях между датой и значением дата-и-время, преобразуйте их в дни с помощью функции`TO_DAYS()`, а затем вычислите разность:

```
mysql> SELECT TO_DAYS('1884-01-01') - TO_DAYS('1883-06-05') AS days;
+------+
| days |
+------+
|  210 |
+------+
```

Чтобы вычислить интервал в неделях, выполните те же операции и разделите результат на семь:

```
mysql> SELECT (TO_DAYS('1884-01-01') - TO_DAYS('1883-06-05')) / 7 AS weeks;
+-------+
| weeks |
+-------+
| 30.00 |
+-------+
```

Для того чтобы преобразовать дни в месяцы или годы, простого деления недостаточно, так как эти единицы имеют переменную длину. Вычисления с выводом интервалов в этих единицах рассматриваются в рецепте 5.19.

Вам нужен интервал или диапазон?

В результате вычисления разности между двумя датами получается интервал между ними. Если вас интересует диапазон, покрываемый этими двумя датами, следует добавить одну единицу. Например, между 2002-01-01 и 2002-01-04 проходит три дня, но вместе они покрывают промежуток в четыре дня. Если в результате вычисления интервала вы получаете не те результаты, которых ожидали, подумайте, не следует ли сделать поправку на 1.

Для значений, относящихся к периоду после начала 1970 года, вы можете определять интервалы с точностью до секунд с помощью функции UNIX_TIMESTAMP(). Например, количество секунд между датами, отстоящими на две недели, можно вычислить так:

```
mysql> SET @dt1 = '1984-01-01 09:00:00';
mysql> SET @dt2 = '1984-01-15 09:00:00';
mysql> SELECT UNIX_TIMESTAMP(@dt2) - UNIX_TIMESTAMP(@dt1) AS seconds;
+---------+
| seconds |
+---------+
| 1209600 |
+---------+
```

Чтобы преобразовать интервал в секундах в другие единицы, выполните соответствующую арифметическую операцию. Секунды без труда преобразуются в минуты, часы, дни и недели:

```
mysql> SET @interval = UNIX_TIMESTAMP(@dt2) - UNIX_TIMESTAMP(@dt1);
mysql> SELECT @interval AS seconds,
    -> @interval / 60 AS minutes,
    -> @interval / (60 * 60) AS hours,
    -> @interval / (24 * 60 * 60) AS days,
    -> @interval / (7 * 24 * 60 * 60) AS weeks;
+---------+---------+-------+------+-------+
| seconds | minutes | hours | days | weeks |
+---------+---------+-------+------+-------+
| 1209600 |   20160 |   336 |   14 |     2 |
+---------+---------+-------+------+-------+
```

Для значений, не попадающих в диапазон с 1970 по 2037 год, можно использовать более общий (но более запутанный) способ вычисления интервалов:

- Вычислите разность в днях между частями значений, относящимися к датам, и умножьте ее на 24*60*60 для преобразования в секунды.
- Сместите результат на разницу в секундах между частями значений, относящимися к времени.

Рассмотрим пример двух значений дата-и-время, отстоящих на неделю:

```
mysql> SET @dt1 = '1800-02-14 07:30:00';
mysql> SET @dt2 = '1800-02-21 07:30:00';
mysql> SET @interval =
    -> ((TO_DAYS(@dt2) - TO_DAYS(@dt1)) * 24*60*60)
    -> + TIME_TO_SEC(@dt2) - TIME_TO_SEC(@dt1);
mysql> SELECT @interval AS seconds;
+---------+
| seconds |
+---------+
|  604800 |
+---------+
```

Чтобы преобразовать интервал в значение `TIME`, передайте его в `SEC_TO_TIME()`:

```
mysql> SELECT SEC_TO_TIME(@interval) AS TIME;
+-----------+
| TIME      |
+-----------+
| 168:00:00 |
+-----------+
```

Чтобы преобразовать интервал из секунд в другие единицы, выполните соответствующее деление:

```
mysql> SELECT @interval AS seconds,
    -> @interval / 60 AS minutes,
    -> @interval / (60 * 60) AS hours,
    -> @interval / (24 * 60 * 60) AS days,
    -> @interval / (7 * 24 * 60 * 60) AS weeks;
+---------+---------+-------+------+-------+
| seconds | minutes | hours | days | weeks |
+---------+---------+-------+------+-------+
|  604800 |   10080 |   168 |    7 |     1 |
+---------+---------+-------+------+-------+
```

Я немного схитрил, выбрав интервал, дающий прекрасные целые значения для всех операций деления. В общем случае, если у вас появится дробная часть, удобно использовать функцию `FLOOR(`*выражение*`)` для отбрасывания этой дробной части и вывода целого значения.

5.18. Стандартизация не-совсем-ISO-строк

Задача

Формат даты похож на стандарт ISO, но все же не совсем ему соответствует.

Решение

Стандартизируйте дату, передав ее функции, всегда возвращающей результат в формате даты ISO.

Обсуждение

Ранее в этой главе (см. рецепт 5.8) упоминалось, что при формировании дат с помощью функции CONCAT() можно столкнуться со следующей проблемой: полученное значение может не соответствовать формату ISO. Например, рассмотрим запрос, выводящий значения первого числа месяца, в которых месяц может быть представлен всего одним разрядом:

```
mysql> SELECT d, CONCAT(YEAR(d),'-',MONTH(d),'-01') FROM date_val;
+------------+-----------------------------------+
| d          | CONCAT(YEAR(d),'-',MONTH(d),'-01') |
+------------+-----------------------------------+
| 1864-02-28 | 1864-2-01                         |
| 1900-01-15 | 1900-1-01                         |
| 1987-03-05 | 1987-3-01                         |
| 1999-12-31 | 1999-12-01                        |
| 2000-06-04 | 2000-6-01                         |
+------------+-----------------------------------+
```

В рецепте 5.8 было показано, как применять LPAD() для обеспечения двузначности значений месяца:

```
mysql> SELECT d, CONCAT(YEAR(d),'-',LPAD(MONTH(d),2,'0'),'-01') FROM date_val;
+------------+-------------------------------------------------+
| d          | CONCAT(YEAR(d),'-',LPAD(MONTH(d),2,'0'),'-01') |
+------------+-------------------------------------------------+
| 1864-02-28 | 1864-02-01                                      |
| 1900-01-15 | 1900-01-01                                      |
| 1987-03-05 | 1987-03-01                                      |
| 1999-12-31 | 1999-12-01                                      |
| 2000-06-04 | 2000-06-01                                      |
+------------+-------------------------------------------------+
```

Есть еще один способ стандартизации дат, близких к формату ISO, – использование их в выражениях, выводящих ISO-дату. Для даты d подойдет любое из выражений:

```
DATE_ADD(d,INTERVAL 0 DAY)
d + INTERVAL 0 DAY
FROM_DAYS(TO_DAYS(d))
```

Например, преобразуем тремя разными способами в формат ISO не-ISO-результаты выполнения операции CONCAT():

```
mysql> SELECT d,
    -> CONCAT(YEAR(d),'-',MONTH(d),'-01') AS 'non-ISO',
    -> DATE_ADD(CONCAT(YEAR(d),'-',MONTH(d),'-01'),INTERVAL 0 DAY) AS method1,
    -> CONCAT(YEAR(d),'-',MONTH(d),'-01') + INTERVAL 0 DAY AS method2,
    -> FROM_DAYS(TO_DAYS(CONCAT(YEAR(d),'-',MONTH(d),'-01'))) AS method3
    -> FROM date_val;
+------------+------------+------------+------------+------------+
| d          | non-ISO    | method1    | method2    | method3    |
+------------+------------+------------+------------+------------+
```

```
| 1864-02-28 | 1864-2-01  | 1864-02-01 | 1864-02-01 | 1864-02-01 |
| 1900-01-15 | 1900-1-01  | 1900-01-01 | 1900-01-01 | 1900-01-01 |
| 1987-03-05 | 1987-3-01  | 1987-03-01 | 1987-03-01 | 1987-03-01 |
| 1999-12-31 | 1999-12-01 | 1999-12-01 | 1999-12-01 | 1999-12-01 |
| 2000-06-04 | 2000-6-01  | 2000-06-01 | 2000-06-01 | 2000-06-01 |
+------------+------------+------------+------------+------------+
```

5.19. Вычисление возраста

Задача

Вы хотите узнать, сколько кому-то лет.

Решение

Следует решить задачу вычисления интервала между датами, но с небольшой особенностью. При вычислении возраста в годах необходимо учитывать относительное положение начальной и конечной дат в календарном году. Если возраст вычисляется в месяцах, также необходимо учитывать относительный порядок месяцев в году и дней в месяце.

Обсуждение

Определение возраста относится к операциям вычисления интервалов, но в данном случае недостаточно простого вычисления разности дат в днях и деления ее на 365. Нельзя забывать о високосных годах. (Интервал с 1995-03-01 по 1996-02-29 покрывает 365 дней, но в терминах возраста – это не год.) Если выполнять деление на 365.25, вычисление будет более точным, но все же корректным не для всех дат. Вместо этого мы будем определять разность дат в годах, а затем при необходимости корректировать ее, принимая во внимание относительное положение дат внутри календарного года. (Пусть, например, Гретхен Смит родилась 14 апреля 1942 года. Чтобы узнать, сколько ей сейчас лет, необходимо посмотреть, в какую часть календарного года попадает текущая дата: до 13-го апреля включительно Гретхен будет на год моложе, чем начиная с 14-го.) В данном разделе показано, как вычислять возраст в годах и месяцах.

Определение возраста в годах

В общем случае, если вы знаете дату рождения `birth`, то возраст в годах на дату `d` можно вычислить так:

```
if (d occurs earlier in the year than birth)
    age = YEAR(d) - YEAR(birth) - 1
if (d occurs on or later in the year than birth)
    age = YEAR(d) - YEAR(birth)
```

Оба раза выполняется одно и то же вычисление – определение разности дат в годах. Разница лишь в относительном порядке дат внутри календарного года. Этот порядок невозможно установить при помощи функции `DAYOFYEAR()`,

поскольку ее результат имеет смысл только для дат, относящихся к годам с одинаковым количеством дней. Как видно из результатов следующего запроса, в разных годах для разных дат функция DAYOFYEAR() может давать одинаковые значения:

```
mysql> SELECT DAYOFYEAR('1995-03-01'), DAYOFYEAR('1996-02-29');
+-------------------------+-------------------------+
| DAYOFYEAR('1995-03-01') | DAYOFYEAR('1996-02-29') |
+-------------------------+-------------------------+
|                      60 |                      60 |
+-------------------------+-------------------------+
```

Здесь помогает то, что строки дат ISO сравниваются как раз так, как нам нужно. Точнее, используется тот факт, что пять самых правых символов значения даты, представляющие месяц и день, сравниваются соответственно:

```
mysql> SELECT RIGHT('1995-03-01',5), RIGHT('1996-02-29',5);
+-----------------------+-----------------------+
| RIGHT('1995-03-01',5) | RIGHT('1996-02-29',5) |
+-----------------------+-----------------------+
| 03-01                 | 02-29                 |
+-----------------------+-----------------------+
mysql> SELECT IF('02-29' < '03-01','02-29','03-01') AS earliest;
+----------+
| earliest |
+----------+
| 02-29    |
+----------+
```

То есть можно выполнить проверку на наиболее раннюю из дат d1 и d2 так:

```
RIGHT(d2,5) < RIGHT(d1,5)
```

В зависимости от результатов проверки выражение принимает значение 1 или 0, так что мы можем использовать результат сравнения < для вычисления возраста в годах:

```
YEAR(d2) - YEAR(d1) - (RIGHT(d2,5) < RIGHT(d1,5))
```

Чтобы было очевидно, как вычисляется выражение, поместим его внутри функции IF(), которая явно возвращает 1 или 0:

```
YEAR(d2) - YEAR(d1) - IF(RIGHT(d2,5) < RIGHT(d1,5),1,0)
```

Применим формулу для вычисления возраста человека, родившегося 1965-03-01, на начало 1975 года. Будем выводить нескорректированную разность в годах, значение корректировки и итоговый возраст:

```
mysql> SET @birth = '1965-03-01';
mysql> SET @target = '1975-01-01';
mysql> SELECT @birth, @target,
    -> YEAR(@target) - YEAR(@birth) AS 'difference',
    -> IF(RIGHT(@target,5) < RIGHT(@birth,5),1,0) AS 'adjustment',
    -> YEAR(@target) - YEAR(@birth)
```

```
    -> - IF(RIGHT(@target,5) < RIGHT(@birth,5),1,0)
    -> AS 'age';
+------------+------------+------------+------------+------+
| @birth     | @target    | difference | adjustment | age  |
+------------+------------+------------+------------+------+
| 1965-03-01 | 1975-01-01 |         10 |          1 |    9 |
+------------+------------+------------+------------+------+
```

Давайте опробуем методику вычисления возраста в годах на таблице `sibling`, в которой перечислены даты рождения Гретхен Смит (Gretchen Smith) и ее братьев Вильбура (Wilbur) и Франца (Franz):

```
+----------+------------+
| name     | birth      |
+----------+------------+
| Gretchen | 1942-04-14 |
| Wilbur   | 1946-11-28 |
| Franz    | 1953-03-05 |
+----------+------------+
```

Получим ответы на следующие вопросы:

- Сколько лет Смитам на сегодняшний день?

```
mysql> SELECT name, birth, CURDATE() AS today,
    -> YEAR(CURDATE()) - YEAR(birth)
    -> - IF(RIGHT(CURDATE(),5) < RIGHT(birth,5),1,0)
    -> AS 'age in years'
    -> FROM sibling;
+----------+------------+------------+--------------+
| name     | birth      | today      | age in years |
+----------+------------+------------+--------------+
| Gretchen | 1942-04-14 | 2002-07-15 |           60 |
| Wilbur   | 1946-11-28 | 2002-07-15 |           55 |
| Franz    | 1953-03-05 | 2002-07-15 |           49 |
+----------+------------+------------+--------------+
```

- Сколько лет было Гретхен и Вильбуру, когда родился Франц?

```
mysql> SELECT name, birth, '1953-03-05' AS 'Franz'' birthday',
    -> YEAR('1953-03-05') - YEAR(birth)
    -> - IF(RIGHT('1953-03-05',5) < RIGHT(birth,5),1,0)
    -> AS 'age in years'
    -> FROM sibling WHERE name != 'Franz';
+----------+------------+-----------------+--------------+
| name     | birth      | Franz' birthday | age in years |
+----------+------------+-----------------+--------------+
| Gretchen | 1942-04-14 | 1953-03-05      |           10 |
| Wilbur   | 1946-11-28 | 1953-03-05      |            6 |
+----------+------------+-----------------+--------------+
```

При выполнении подобных вычислений не забывайте о том, что сравнения частей вида *MM-DD* строк выдают правильные результаты, только если используются значения ISO, такие как `1987-07-01`, но не значения, близкие-к-

формату-ISO, такие как 1987-7-1. Например, следующее сравнение выводит лексически правильный результат, но некорректный в терминах времени:

```
mysql> SELECT RIGHT('1987-7-1',5) < RIGHT('1987-10-01',5);
+---------------------------------------------+
| RIGHT('1987-7-1',5) < RIGHT('1987-10-01',5) |
+---------------------------------------------+
|                                           0 |
+---------------------------------------------+
```

Отсутствие начальных нулей в значениях дня и месяца приводит к неправильному сравнению строк.

Определение возраста в месяцах

Вычисление возраста в месяцах – это процедура, аналогичная вычислению в годах, только разность в годах умножается на 12, добавляется разность в месяцах и учитывается относительное положение дней двух дат в пределах календарного месяца. В этом случае необходимо по отдельности использовать части месяца и дня каждой даты, так что можно не сравнивать части *MM-DD* строк дат, а извлечь их, используя функции MONTH() и DAYOFMONTH(). Текущий возраст членов семьи Смитов в месяцах можно вычислить так:

```
mysql> SELECT name, birth, CURDATE() AS today,
    -> (YEAR(CURDATE()) - YEAR(birth)) * 12
    -> + (MONTH(CURDATE()) - MONTH(birth))
    -> - IF(DAYOFMONTH(CURDATE()) < DAYOFMONTH(birth),1,0)
    -> AS 'age in months'
    -> FROM sibling;
+----------+------------+------------+---------------+
| name     | birth      | today      | age in months |
+----------+------------+------------+---------------+
| Gretchen | 1942-04-14 | 2002-07-15 |           723 |
| Wilbur   | 1946-11-28 | 2002-07-15 |           667 |
| Franz    | 1953-03-05 | 2002-07-15 |           592 |
+----------+------------+------------+---------------+
```

5.20. Смещение даты на заданную величину

Задача

Вы хотите сдвинуть указанную дату на заданную величину для получения новой даты.

Решение

Используйте функции DATE_ADD() или DATE_SUB().

Обсуждение

Если у вас есть исходная дата и требуется получить из нее другую, отличающуюся на заданный интервал, то обычно задача решается элементарными

арифметическими операциями при помощи функций DATE_ADD() и DATE_SUB(). Примеры: вычисление памятных дат, определение сроков годности или нахождение записей, удовлетворяющих запросам типа «этот день в истории». Некоторые из них будут рассмотрены в данном разделе.

Вычисление памятных дат

Предположим, что ваша свадьба состоялась 6 августа 2003 года, и вы не хотите ждать целый год вашу первую годовщину свадьбы, чтобы выказать преданность своей половине. Вместо этого вы хотели бы приготовить специальные подарки на 1 неделю, 1 месяц, 3 месяца и 6 месяцев со дня свадьбы. Чтобы узнать, когда они наступят, выполните смещение даты вашей свадьбы на указанные интервалы:

```
mysql> SET @d = '2003-08-06';
mysql> SELECT @d AS 'start date',
    -> DATE_ADD(@d,INTERVAL 7 DAY) AS '1 week',
    -> DATE_ADD(@d,INTERVAL 1 MONTH) AS '1 month',
    -> DATE_ADD(@d,INTERVAL 3 MONTH) AS '3 months',
    -> DATE_ADD(@d,INTERVAL 6 MONTH) AS '6 months';
+------------+------------+------------+------------+------------+
| start date | 1 week     | 1 month    | 3 months   | 6 months   |
+------------+------------+------------+------------+------------+
| 2003-08-06 | 2003-08-13 | 2003-09-06 | 2003-11-06 | 2004-02-06 |
+------------+------------+------------+------------+------------+
```

Если вас интересует только часть памятной даты, можно обойтись без арифметических операций для дат. Например, если вы закончили школу 4 июня 2000 года и хотите узнать, в каком году состоятся встречи по случаю 10-й, 20-й и 40-й годовщины окончания, то нет необходимости применять функцию DATE_ADD(). Просто извлеките из исходной даты часть, относящуюся к году, и добавьте к ней 10, 20 и 40 с помощью обычных арифметических операций:

```
mysql> SET @y = YEAR('2000-06-04');
mysql> SELECT @y + 10, @y + 20, @y + 40;
+---------+---------+---------+
| @y + 10 | @y + 20 | @y + 40 |
+---------+---------+---------+
|    2010 |    2020 |    2040 |
+---------+---------+---------+
```

Учет часовых поясов

Сервер MySQL возвращает даты, соответствующие тому часовому поясу, где находится хост, на котором работает сервер. Если клиентская программа запускается в другом часовом поясе, следует изменить значения на локальное время клиента с помощью функции DATE_ADD(). Чтобы преобразовать время для сервера, который на два часа опережает клиента, вычтем два часа:

```
mysql> SELECT dt AS 'server time',
    -> DATE_ADD(dt,INTERVAL -2 HOUR) AS 'client time'
```

```
    -> FROM datetime_val;
+---------------------+---------------------+
| server time         | client time         |
+---------------------+---------------------+
| 1970-01-01 00:00:00 | 1969-12-31 22:00:00 |
| 1987-03-05 12:30:15 | 1987-03-05 10:30:15 |
| 1999-12-31 09:00:00 | 1999-12-31 07:00:00 |
| 2000-06-04 15:45:30 | 2000-06-04 13:45:30 |
+---------------------+---------------------+
```

Имейте в виду, что у сервера нет никакой информации о часовом поясе клиента, так что именно вы должны определить величину смещения времени клиента. В сценарии это можно сделать, получив текущее локальное время и сравнив его с представлением сервера о локальном времени. В Perl для таких случаев есть функция `localtime()`:

```
my ($sec, $min, $hour, $day, $mon, $year) = localtime (time ());
my $now = sprintf ("%04d-%02d-%02d %02d:%02d:%02d",
                   $year + 1900, $mon + 1, $day, $hour, $min, $sec);
my ($server_now, $adjustment) = $dbh->selectrow_array (
        "SELECT NOW(), UNIX_TIMESTAMP(?) - UNIX_TIMESTAMP(NOW())",
        undef, $now);
print "client now: $now\n";
print "server now: $server_now\n";
print "adjustment (secs): $adjustment\n";
```

5.21. Нахождение первого и последнего дней месяца

Задача

У вас есть дата, и необходимо узнать дату первого или последнего дня месяца, к которому она относится, или дату первого или последнего дня месяца *n* месяцев назад.

Решение

Выполните смещение даты.

Обсуждение

Порой требуется вычислить дату, не имеющую фиксированной связи с исходной датой. Например, при нахождении первого дня месяца величина смещения исходной даты зависит от дня месяца даты и от длины месяца.

Чтобы найти первый день того месяца, к которому относится исходная дата, сдвиньте ее назад на количество дней, которое на единицу меньше, чем значение функции `DAYOFMONTH()`:

```
mysql> SELECT d, DATE_SUB(d,INTERVAL DAYOFMONTH(d)-1 DAY) AS '1st of month'
    -> FROM date_val;
```

```
+------------+--------------+
| d          | 1st of month |
+------------+--------------+
| 1864-02-28 | 1864-02-01   |
| 1900-01-15 | 1900-01-01   |
| 1987-03-05 | 1987-03-01   |
| 1999-12-31 | 1999-12-01   |
| 2000-06-04 | 2000-06-01   |
+------------+--------------+
```

В общем случае, для того чтобы найти первый день месяца для любого месяца на *n* месяцев раньше указанной даты, вычислите первый день месяца, к которому относится дата, затем сдвиньте результат на *n* месяцев:

```
DATE_ADD(DATE_SUB(d,INTERVAL DAYOFMONTH(d)-1 DAY),INTERVAL n MONTH)
```

Например, чтобы найти первый день предыдущего и последующего месяцев по отношению к указанной дате, рассматриваем *n* как –1 и 1:

```
mysql> SELECT d,
    -> DATE_ADD(DATE_SUB(d,INTERVAL DAYOFMONTH(d)-1 DAY),INTERVAL -1 MONTH)
    -> AS '1st of previous month',
    -> DATE_ADD(DATE_SUB(d,INTERVAL DAYOFMONTH(d)-1 DAY),INTERVAL 1 MONTH)
    -> AS '1st of following month'
    -> FROM date_val;
+------------+----------------------+-----------------------+
| d          | 1st of previous month | 1st of following month |
+------------+----------------------+-----------------------+
| 1864-02-28 | 1864-01-01           | 1864-03-01            |
| 1900-01-15 | 1899-12-01           | 1900-02-01            |
| 1987-03-05 | 1987-02-01           | 1987-04-01            |
| 1999-12-31 | 1999-11-01           | 2000-01-01            |
| 2000-06-04 | 2000-05-01           | 2000-07-01            |
+------------+----------------------+-----------------------+
```

Найти последний день месяца по указанной исходной дате несколько сложнее, так как месяцы могут иметь различную длину. Однако последний день месяца – это всегда день перед первым днем следующего месяца, который мы уже умеем находить. Тогда общая процедура вычисления последнего дня месяца, на *n* месяцев отстоящего от указанной даты, будет такой:

1. Найти первый день месяца.
2. Сдвинуть результат на *n*+1 месяцев.
3. Сдвинуть на день назад.

Выражение SQL, выполняющее подобную операцию, выглядит так:

```
DATE_SUB(
    DATE_ADD(DATE_SUB(d,INTERVAL DAYOFMONTH(d)-1 DAY),INTERVAL n+1 MONTH),
    INTERVAL 1 DAY)
```

Вычислим, например, последний день для предыдущего, текущего и последующего месяцев для указанной даты (*n* будет равно –1, 0 и 1):

```
mysql> SELECT d,
    -> DATE_SUB(
    -> DATE_ADD(DATE_SUB(d,INTERVAL DAYOFMONTH(d)-1 DAY),INTERVAL 0 MONTH),
    -> INTERVAL 1 DAY)
    -> AS 'last, prev. month',
    -> DATE_SUB(
    -> DATE_ADD(DATE_SUB(d,INTERVAL DAYOFMONTH(d)-1 DAY),INTERVAL 1 MONTH),
    -> INTERVAL 1 DAY)
    -> AS 'last, this month',
    -> DATE_SUB(
    -> DATE_ADD(DATE_SUB(d,INTERVAL DAYOFMONTH(d)-1 DAY),INTERVAL 2 MONTH),
    -> INTERVAL 1 DAY)
    -> AS 'last, following month'
    -> FROM date_val;
+------------+------------------+------------------+-----------------------+
| d          | last, prev. month | last, this month | last, following month |
+------------+------------------+------------------+-----------------------+
| 1864-02-28 | 1864-01-31       | 1864-02-29       | 1864-03-31            |
| 1900-01-15 | 1899-12-31       | 1900-01-31       | 1900-02-28            |
| 1987-03-05 | 1987-02-28       | 1987-03-31       | 1987-04-30            |
| 1999-12-31 | 1999-11-30       | 1999-12-31       | 2000-01-31            |
| 2000-06-04 | 2000-05-31       | 2000-06-30       | 2000-07-31            |
+------------+------------------+------------------+-----------------------+
```

Последний день предыдущего месяца является особым случаем, для которого можно несколько упростить общее выражение:

```
mysql> SELECT d,
    -> DATE_SUB(d,INTERVAL DAYOFMONTH(d) DAY)
    -> AS 'last of previous month'
    -> FROM date_val;
+------------+-----------------------+
| d          | last of previous month |
+------------+-----------------------+
| 1864-02-28 | 1864-01-31            |
| 1900-01-15 | 1899-12-31            |
| 1987-03-05 | 1987-02-28            |
| 1999-12-31 | 1999-11-30            |
| 2000-06-04 | 2000-05-31            |
+------------+-----------------------+
```

Основная особенность общего выражения нахождения последнего дня месяца в том, что оно начинается с вычисления первого дня месяца заданной даты. Это удобно, так как вы всегда можете сдвинуть начало месяца вперед или назад для получения первого числа другого месяца, сдвинув которое на день назад, вы получите последний день предыдущего месяца. Если же определять значения последних дней месяцев, находя последний день месяца для указанной даты, а затем сдвигая его, результат далеко не всегда будет корректным, так как не во всех месяцах одинаковое количество дней. Например, некорректно находить последний день месяца, вычисляя последний день предыдущего месяца и добавляя месяц:

```
mysql> SELECT d,
    -> DATE_ADD(DATE_SUB(d,INTERVAL DAYOFMONTH(d) DAY),INTERVAL 1 MONTH)
    -> AS 'last of month'
    -> FROM date_val;
+------------+---------------+
| d          | last of month |
+------------+---------------+
| 1864-02-28 | 1864-02-29    |
| 1900-01-15 | 1900-01-31    |
| 1987-03-05 | 1987-03-28    |
| 1999-12-31 | 1999-12-30    |
| 2000-06-04 | 2000-06-30    |
+------------+---------------+
```

Этот способ не подходит, так как часть день-месяца результирующего значения даты может быть неправильной. Для строк `1987-03-05` и `1999-12-31` последний день месяца вычислен неправильно. И так будет каждый раз, когда в месяце, предшествующем заданной дате, меньше дней, чем в нужном месяце.

5.22. Вычисление длины месяца

Задача

Вы хотите узнать, сколько дней в месяце.

Решение

Определите последний день месяца и извлеките из результата часть день-месяца.

Обсуждение

Чтобы вычислить количество дней для месяца, к которому относится указанная дата, определите дату последнего дня месяца (см. предыдущий раздел) и извлеките из результата значение `DAYOFMONTH()`:

```
mysql> SELECT d,
    -> DAYOFMONTH(DATE_SUB(
    -> DATE_ADD(DATE_SUB(d,INTERVAL DAYOFMONTH(d)-1 DAY),INTERVAL 1 MONTH),
    -> INTERVAL 1 DAY))
    -> AS 'days in month'
    -> FROM date_val;
+------------+---------------+
| d          | days in month |
+------------+---------------+
| 1864-02-28 |            29 |
| 1900-01-15 |            31 |
| 1987-03-05 |            31 |
| 1999-12-31 |            31 |
| 2000-06-04 |            30 |
+------------+---------------+
```

См. также

В рецепте 5.27 далее в этой главе предложен другой способ вычисления длины месяца. В главе 10 обсуждаются вычисления, связанные с високосными годами, в контексте проверки достоверности данных.

5.23. Получение одной даты из другой заменой подстроки

Задача

У вас есть дата, и вы хотите получить из нее некоторую другую дату, зная при этом, что у них есть общие составляющие.

Решение

Рассматривая значение даты или времени как строку, выполните непосредственную подстановку частей строки.

Обсуждение

В некоторых ситуациях для вычисления даты можно не выполнять никакие арифметические операции, а просто произвести замену подстрок. Например, вы можете применять строковые операции для вывода первого дня месяца, к которому относится дата, заменив составляющую дней на значение `01`. Используйте функцию `DATE_FORMAT()` или `CONCAT()`:

```
mysql> SELECT d,
    -> DATE_FORMAT(d,'%Y-%m-01') AS method1,
    -> CONCAT(YEAR(d),'-',LPAD(MONTH(d),2,'0'),'-01') AS method2
    -> FROM date_val;
+------------+------------+------------+
| d          | method1    | method2    |
+------------+------------+------------+
| 1864-02-28 | 1864-02-01 | 1864-02-01 |
| 1900-01-15 | 1900-01-01 | 1900-01-01 |
| 1987-03-05 | 1987-03-01 | 1987-03-01 |
| 1999-12-31 | 1999-12-01 | 1999-12-01 |
| 2000-06-04 | 2000-06-01 | 2000-06-01 |
+------------+------------+------------+
```

Подстановку строк можно использовать и для получения фиксированной даты календарного года. Чтобы получить день Нового года (1 января), замените месяц и день на `01`:

```
mysql> SELECT d,
    -> DATE_FORMAT(d,'%Y-01-01') AS method1,
    -> CONCAT(YEAR(d),'-01-01') AS method2
    -> FROM date_val;
```

```
+------------+------------+------------+
| d          | method1    | method2    |
+------------+------------+------------+
| 1864-02-28 | 1864-01-01 | 1864-01-01 |
| 1900-01-15 | 1900-01-01 | 1900-01-01 |
| 1987-03-05 | 1987-01-01 | 1987-01-01 |
| 1999-12-31 | 1999-01-01 | 1999-01-01 |
| 2000-06-04 | 2000-01-01 | 2000-01-01 |
+------------+------------+------------+
```

Чтобы вывести дату Рождества, замените месяц и день значениями 12 и 25 соответственно:

```
mysql> SELECT d,
    -> DATE_FORMAT(d,'%Y-12-25') AS method1,
    -> CONCAT(YEAR(d),'-12-25') AS method2
    -> FROM date_val;
+------------+------------+------------+
| d          | method1    | method2    |
+------------+------------+------------+
| 1864-02-28 | 1864-12-25 | 1864-12-25 |
| 1900-01-15 | 1900-12-25 | 1900-12-25 |
| 1987-03-05 | 1987-12-25 | 1987-12-25 |
| 1999-12-31 | 1999-12-25 | 1999-12-25 |
| 2000-06-04 | 2000-12-25 | 2000-12-25 |
+------------+------------+------------+
```

Чтобы выполнить операцию нахождения даты Рождества для других лет, используйте замену строк в сочетании со сдвигом даты. Следующий запрос иллюстрирует две возможности вычисления даты Рождества два года спустя по отношению к сегодняшней дате. Первый способ: сначала находим Рождество текущего года, затем сдвигаем дату на 2 года вперед. Второй способ: смещаем на 2 года текущую дату, затем находим Рождество полученного года:

```
mysql> SELECT CURDATE(),
    -> DATE_ADD(DATE_FORMAT(CURDATE(),'%Y-12-25'),INTERVAL 2 YEAR)
    -> AS method1,
    -> DATE_FORMAT(DATE_ADD(CURDATE(),INTERVAL 2 YEAR),'%Y-12-25')
    -> AS method2;
+------------+------------+------------+
| CURDATE()  | method1    | method2    |
+------------+------------+------------+
| 2002-07-15 | 2004-12-25 | 2004-12-25 |
+------------+------------+------------+
```

5.24. Определение дня недели для даты

Задача

Вы хотите узнать, на какой день недели выпадает указанная дата.

Решение

Используйте функцию DAYNAME().

Обсуждение

Чтобы вывести название дня недели, примените функцию DAYNAME():

```
mysql> SELECT CURDATE(), DAYNAME(CURDATE());
+------------+--------------------+
| CURDATE()  | DAYNAME(CURDATE()) |
+------------+--------------------+
| 2002-07-15 | Monday             |
+------------+--------------------+
```

Функцию DAYNAME() удобно использовать в сочетании с другими методиками работы с датами. Например, чтобы узнать, каким днем недели будет первый день месяца, используйте выражение поиска первого дня месяца, приведенное ранее в этой главе, как аргумент функции DAYNAME():

```
mysql> SET @d = CURDATE();
mysql> SET @first = DATE_SUB(@d,INTERVAL DAYOFMONTH(@d)-1 DAY);
mysql> SELECT @d AS 'starting date',
    -> @first AS '1st of month date',
    -> DAYNAME(@first) AS '1st of month day';
+---------------+-------------------+------------------+
| starting date | 1st of month date | 1st of month day |
+---------------+-------------------+------------------+
| 2002-07-15    | 2002-07-01        | Monday           |
+---------------+-------------------+------------------+
```

5.25. Определение дат для дней текущей недели

Задача

Вы хотите узнать дату какого-то дня текущей недели.

Решение

Вычислите количество дней между началом недели и интересующим вас днем и сдвиньте дату на полученное значение.

Обсуждение

В этом и следующем разделах рассказано о том, как преобразовывать одну дату в другую при условии, что дата, которую нужно получить, указана в терминах дней недели. Например, если вы хотите узнать, каким числом будет вторник на этой неделе, то все зависит от того, какой день недели сегодня. Если сегодня понедельник, следует прибавить день к CURDATE(), если же сегодня среда, необходимо вычесть один день.

MySQL содержит две полезные функции. DAYOFWEEK() считает началом недели воскресенье и возвращает значения от 1 до 7 для дней недели, начиная с воскресенья и заканчивая субботой. Функция WEEKDAY() воспринимает понедельник как начало недели и возвращает значения от 0 до 6 для дней с понедельника по воскресенье (в примерах будет использоваться DAYOFWEEK()). Можно получать не номера дней недели, а их названия, – этим занимается DAYNAME().

Вычисления, определяющие один день недели на основе другого, зависят как от исходного значения, так и от конечного. Мне кажется, что проще всего сначала сместить начальную дату в точку, положение которой относительно начала недели зафиксировано, а затем выполнить обратное смещение:

- Сдвиньте исходную дату назад на количество дней, равное ее значению DAYOFWEEK(), в результате чего вы всегда получите дату субботы предыдущей недели.
- Добавьте один день, чтобы получить дату воскресенья, два дня, чтобы получить дату понедельника и т. д.

В SQL для получения дат дней с воскресенья по субботу для исходной даты d можно выполнить такую операцию (где *n* равно от 1 до 7):

```
DATE_ADD(DATE_SUB(d,INTERVAL DAYOFWEEK(d) DAY),INTERVAL n DAY)
```

Выражение выделяет сдвиг к субботе и обратный сдвиг в отдельные операции, но поскольку интервалы DATE_SUB()и DATE_ADD() измеряются в днях, можно упростить выражение, используя только вызов DATE_ADD():

```
DATE_ADD(d,INTERVAL n-DAYOFWEEK(d) DAY)
```

Применим этот прием к таблице date_val, используя значение *n*, равное 1 для воскресенья и 7 – для субботы, для нахождения первого и последнего дней недели:

```
mysql> SELECT d, DAYNAME(d) AS day,
    -> DATE_ADD(d,INTERVAL 1-DAYOFWEEK(d) DAY) AS Sunday,
    -> DATE_ADD(d,INTERVAL 7-DAYOFWEEK(d) DAY) AS Saturday
    -> FROM date_val;
+------------+----------+------------+------------+
| d          | day      | Sunday     | Saturday   |
+------------+----------+------------+------------+
| 1864-02-28 | Sunday   | 1864-02-28 | 1864-03-05 |
| 1900-01-15 | Monday   | 1900-01-14 | 1900-01-20 |
| 1987-03-05 | Thursday | 1987-03-01 | 1987-03-07 |
| 1999-12-31 | Friday   | 1999-12-26 | 2000-01-01 |
| 2000-06-04 | Sunday   | 2000-06-04 | 2000-06-10 |
+------------+----------+------------+------------+
```

5.26. Определение дат для дней других недель

Задача

Вы хотите вычислить дату некоторого дня некоторой недели (не текущей).

Решение

Определите дату этого дня недели для текущей недели, затем сместите результат в интересующую вас неделю.

Обсуждение

Вычисление даты дня недели какой-то другой недели – это задача, которая разбивается на сдвиг на значение дня недели (см. предыдущий раздел) и сдвиг на недели. Порядок выполнения операций не имеет значения, так как величина сдвига внутри недели не зависит от того, смещена ли исходная дата. Например, чтобы вычислить по приведенной ранее формуле, каким числом будет среда, возьмем *n*, равное 4. Чтобы вычислить дату среды, наступившей две недели назад, вы можете сначала выполнить сдвиг дня недели:

```
mysql> SET @target =
    -> DATE_SUB(DATE_ADD(CURDATE(),INTERVAL 4-DAYOFWEEK(CURDATE()) DAY),
    -> INTERVAL 14 DAY);
mysql> SELECT CURDATE(), @target, DAYNAME(@target);
+------------+------------+------------------+
| CURDATE()  | @target    | DAYNAME(@target) |
+------------+------------+------------------+
| 2002-07-15 | 2002-07-03 | Wednesday        |
+------------+------------+------------------+
```

А можете сначала сдвинуть неделю:

```
mysql> SET @target =
    -> DATE_ADD(DATE_SUB(CURDATE(),INTERVAL 14 DAY),
    -> INTERVAL 4-DAYOFWEEK(CURDATE()) DAY);
mysql> SELECT CURDATE(), @target, DAYNAME(@target);
+------------+------------+------------------+
| CURDATE()  | @target    | DAYNAME(@target) |
+------------+------------+------------------+
| 2002-07-15 | 2002-07-03 | Wednesday        |
+------------+------------+------------------+
```

Некоторым приложениям необходимо знать такие даты, как *n*-е вхождение какого-то дня недели. Например, если вы ведете платежную ведомость, выплаты по которой проводятся во 2-й и 4-й четверг каждого месяца, то вам необходимо знать даты этих дней. Одним из способов выполнения операции для текущего месяца будет нахождение первого дня месяца и его последующий сдвиг. Изменить дату на четверг текущей недели достаточно просто, проблема в том, как узнать, на сколько недель сдвигать результат, чтобы получить именно второй и четвертый четверги. Если первый день месяца выпадает на дни с воскресенья по четверг, то для получения второго четверга следует выполнить сдвиг на одну неделю. Если же первый день месяца выпадает на пятницу или последующие дни, сдвигаем на две недели. Четвертый четверг наступает, естественно, через две недели после второго.

Приведем программу на Perl, вычисляющую все даты выдачи заработной платы в 2002 году. Выполняется цикл, который определяет дату первого дня месяца для всех месяцев года. Для каждого месяца выдается запрос, выводящий даты 2-го и 4-го четвергов:

```
my $year = 2002;
print "MM/CCYY   2nd Thursday   4th Thursday\n";
foreach my $month (1..12)
{
    my $first = sprintf ("%04d-%02d-01", $year, $month);
    my ($thu2, $thu4) = $dbh->selectrow_array (qq{
                SELECT
                    DATE_ADD(
                        DATE_ADD(?,INTERVAL 5-DAYOFWEEK(?) DAY),
                        INTERVAL IF(DAYOFWEEK(?) <= 5, 7, 14) DAY),
                    DATE_ADD(
                        DATE_ADD(?,INTERVAL 5-DAYOFWEEK(?) DAY),
                        INTERVAL IF(DAYOFWEEK(?) <= 5, 21, 28) DAY)
            }, undef, $first, $first, $first, $first, $first, $first);
    printf "%02d/%04d   %s     %s\n", $month, $year, $thu2, $thu4;
}
```

Вывод программы выглядит следующим образом:

```
MM/CCYY   2nd Thursday   4th Thursday
01/2002   2002-01-10     2002-01-24
02/2002   2002-02-14     2002-02-28
03/2002   2002-03-14     2002-03-28
04/2002   2002-04-11     2002-04-25
05/2002   2002-05-09     2002-05-23
06/2002   2002-06-13     2002-06-27
07/2002   2002-07-11     2002-07-25
08/2002   2002-08-08     2002-08-22
09/2002   2002-09-12     2002-09-26
10/2002   2002-10-10     2002-10-24
11/2002   2002-11-14     2002-11-28
12/2002   2002-12-12     2002-12-26
```

5.27. Вычисления для високосных годов

Задача

Вам нужно выполнить вычисление даты, которое принимало бы во внимание наличие високосных годов. Например, длина месяца или года зависит от того, относится ли дата к високосному году.

Решение

Научитесь определять, является ли год високосным, и учтите результат этой проверки при выполнении вычислений.

Обсуждение

Вычисления, связанные с датами, осложняются тем, что не все месяцы содержат одинаковое количество дней, а 29-й день февраля, появляющийся только в високосном году – это вообще отдельная головная боль. В этом разделе рассказано о том, как определить, относится ли указанная дата к високосному году, а также как обрабатывать високосные годы при вычислении длины месяца или года.

Как определить, относится ли дата к високосному году

Чтобы узнать, относится ли дата `d` к високосному году, выделите составляющую года с помощью функции `YEAR()` и проверьте результат. Часто для проверки на високосный год применяют правило делимости на четыре, которое можно реализовать при помощи оператора деления по модулю:

```
YEAR(d) % 4 = 0
```

Однако такая проверка технически некорректна (например, 1900 год делится на четыре, но *не* является високосным). Для того чтобы год был високосным, он должен удовлетворять сразу двум условиям:

- Год должен делиться на четыре.
- Год не должен делиться на 100, если только он не делится и на 400.

Смысл второго ограничения в том, что год начала века не является високосным, за исключением каждого четвертого века. В SQL можно записать эти правила следующим образом:

```
(YEAR(d) % 4 = 0) AND ((YEAR(d) % 100 != 0) OR (YEAR(d) % 400 = 0))
```

Проверим правила високосности для данных таблицы `date_val` и получим такие результаты:

```
mysql> SELECT
    -> d,
    -> YEAR(d) % 4 = 0
    -> AS "rule-of-thumb test",
    -> (YEAR(d) % 4 = 0) AND ((YEAR(d) % 100 != 0) OR (YEAR(d) % 400 = 0))
    -> AS "complete test"
    -> FROM date_val;
+------------+--------------------+---------------+
| d          | rule-of-thumb test | complete test |
+------------+--------------------+---------------+
| 1864-02-28 |                  1 |             1 |
| 1900-01-15 |                  1 |             0 |
| 1987-03-05 |                  0 |             0 |
| 1999-12-31 |                  0 |             0 |
| 2000-06-04 |                  1 |             1 |
+------------+--------------------+---------------+
```

Как видите, полная проверка выводит иные результаты, нежели предыдущая. Первый тест некорректно обрабатывает 1900 год, второй же учитывает ограничения на начало века и работает правильно.

Так как полная проверка на високосный год включает в себя проверку века, ей необходимы четырехзначные значения годов. Для двузначных годов проверка ограничения на начало века невозможна, так как при этом нельзя однозначно определить век.

Если вы работаете со значениями дат в программе, то можете выполнять проверки на високосный год не на уровне SQL, а в своем языке API. Выделите первые четыре разряда строки даты для получения года и проверьте его. Если язык осуществляет автоматическое преобразование значения года из строки в число, все просто. В противном случае придется дополнительно преобразовать значение года в число, прежде чем проверять его.

В Perl и PHP проверка на високосный год имеет такой формат:

```
$year = substr ($date, 0, 4);
$is_leap = ($year % 4 == 0) && ($year % 100 != 0 || $year % 400 == 0);
```

Синтаксис Python аналогичен, но еще требуется операция преобразования типа:

```
year = int (date[0:4])
is_leap = (year % 4 == 0) and (year % 100 != 0 or year % 400 == 0)
```

Преобразование типа необходимо и в Java:

```
int year = Integer.valueOf (date.substring (0, 4)).intValue ();
boolean is_leap = (year % 4 == 0) && (year % 100 != 0 || year % 400 == 0);
```

Использование проверки на високосный год для вычисления длины года

Обычно в году 365 дней, но високосный год имеет один дополнительный день. Чтобы определить длину года, к которому относится указанная дата, можно провести только что описанную проверку на високосный год, чтобы знать, следует ли прибавлять лишний день:

```
$year = substr ($date, 0, 4);
$is_leap = ($year % 4 == 0) && ($year % 100 != 0 || $year % 400 == 0);
$days_in_year = ($is_leap ? 366 : 365);
```

Есть и другой способ вычисления длины года: можно вычислить дату последнего дня года и передать ее в функцию DAYOFYEAR():

```
mysql> SET @d = '2003-04-13';
mysql> SELECT DAYOFYEAR(DATE_FORMAT(@d,'%Y-12-31'));
+--------------------------------------+
| DAYOFYEAR(DATE_FORMAT(@d,'%Y-12-31')) |
+--------------------------------------+
|                                  365 |
+--------------------------------------+
mysql> SET @d = '2004-04-13';
mysql> SELECT DAYOFYEAR(DATE_FORMAT(@d,'%Y-12-31'));
+--------------------------------------+
| DAYOFYEAR(DATE_FORMAT(@d,'%Y-12-31')) |
+--------------------------------------+
|                                  366 |
+--------------------------------------+
```

Использование проверки на високосный год для вычисления длины месяца

В рецепте 5.22 обсуждалось, как определить количество дней в месяце при помощи сдвига даты для нахождения последнего дня месяца. Проверка на високосный год обеспечивает альтернативный способ достижения той же цели. Все месяцы, кроме февраля, имеют фиксированную длину, так что посмотрев на составляющую месяца указанной даты, вы можете сказать, какова длина этого месяца. Если вы знаете, относится ли дата к високосному году, то можете определить и длину февраля.

SQL-выражение, вычисляющее количество дней месяца, можно записать так:

```
mysql> SELECT d,
    -> ELT(MONTH(d),
    -> 31,
    -> IF((YEAR(d)%4 = 0) AND ((YEAR(d)%100 != 0) OR (YEAR(d)%400 = 0)),29,28),
    -> 31,30,31,30,31,31,30,31,30,31)
    -> AS 'days in month'
    -> FROM date_val;
+------------+---------------+
| d          | days in month |
+------------+---------------+
| 1864-02-28 | 29            |
| 1900-01-15 | 31            |
| 1987-03-05 | 31            |
| 1999-12-31 | 31            |
| 2000-06-04 | 30            |
+------------+---------------+
```

Функция `ELT()` оценивает свой первый аргумент для определения значения *n*, затем возвращает *n*-е значение следующих аргументов. Все просто для всех месяцев, кроме февраля, для которого `ELT()` должна вернуть 29 или 28 в зависимости от того, является ли год високосным.

В языке API вы можете написать функцию, которая, получив в качестве аргумента дату в формате ISO, возвращала бы количество дней месяца, к которому относится дата. Приведем версию на Perl:

```
sub days_in_month
{
my $date = shift;
my $year = substr ($date, 0, 4);
my $month = substr ($date, 5, 2);    # month, 1-based
my @days_in_month = (31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31);
my $days = $days_in_month[$month-1];
my $is_leap = ($year % 4 == 0) && ($year % 100 != 0 || $year % 400 == 0);

    $days++ if $month == 2 && $is_leap; # add a day for Feb of leap years
    return ($days);
}
```

5.28. Обработка даты и времени как чисел

Задача

Вы хотите работать с временной строкой как с числом.

Решение

Преобразуйте строку в число.

Обсуждение

MySQL часто разрешает обрабатывать значения даты и времени как числа. Это может быть удобно для выполнения арифметических операций со значениями. Чтобы вызвать преобразование значения времени в число, добавьте к нему ноль или используйте его в числовом контексте:

```
mysql> SELECT t1,
    -> t1+0 AS 't1 as number',
    -> FLOOR(t1) AS 't1 as number',
    -> FLOOR(t1/10000) AS 'hour part'
    -> FROM time_val;
+----------+--------------+--------------+-----------+
| t1       | t1 as number | t1 as number | hour part |
+----------+--------------+--------------+-----------+
| 15:00:00 |       150000 |       150000 |        15 |
| 05:01:30 |        50130 |        50130 |         5 |
| 12:30:20 |       123020 |       123020 |        12 |
+----------+--------------+--------------+-----------+
```

Такое же преобразование можно выполнить и для даты или значения дата-и-время:

```
mysql> SELECT d, d+0 FROM date_val;
+------------+----------+
| d          | d+0      |
+------------+----------+
| 1864-02-28 | 18640228 |
| 1900-01-15 | 19000115 |
| 1987-03-05 | 19870305 |
| 1999-12-31 | 19991231 |
| 2000-06-04 | 20000604 |
+------------+----------+
mysql> SELECT dt, dt+0 FROM datetime_val;
+---------------------+----------------+
| dt                  | dt+0           |
+---------------------+----------------+
| 1970-01-01 00:00:00 | 19700101000000 |
| 1987-03-05 12:30:15 | 19870305123015 |
| 1999-12-31 09:00:00 | 19991231090000 |
| 2000-06-04 15:45:30 | 20000604154530 |
+---------------------+----------------+
```

Значение, порожденное добавлением нуля, – это не то же самое, что значение, полученное в результате преобразования в базовые единицы, такие как секунды или дни. При добавлении ноля вы по сути удаляете разделители из строкового представления исходного значения. Кроме того, преобразование в число возможно только для значений, которые MySQL воспринимает как значения времени. Если вы попытаетесь преобразовать в число литерную строку, добавляя к ней 0, то получите только первую часть значения:

```
mysql> SELECT '1999-01-01'+0, '1999-01-01 12:30:45'+0, '12:30:45'+0;
+----------------+-------------------------+--------------+
| '1999-01-01'+0 | '1999-01-01 12:30:45'+0 | '12:30:45'+0 |
+----------------+-------------------------+--------------+
|           1999 |                    1999 |           12 |
+----------------+-------------------------+--------------+
```

То же самое случится и при использовании функций `DATE_FORMAT()` и `TIME_FORMAT()`, при извлечении частей значений `DATETIME` или `TIMESTAMP` посредством функций `LEFT()` или `RIGHT()`. В контексте прибавления 0 результаты этих функций рассматриваются как строки, а не как значения времени.

5.29. Обработка в MySQL строк как значений времени

Задача

Вы хотите, чтобы строка интерпретировалась как значение времени.

Решение

Используйте строку в контексте времени, чтобы дать MySQL понять, как вы бы хотели ее интерпретировать.

Обсуждение

Если вам нужно заставить MySQL интерпретировать строку как дату или время, используйте ее в выражении, которое, не изменяя значения, создает временной контекст. Например, вы не можете добавить ноль к строковому литералу `TIME`, инициируя преобразование времени в число, но если использовать функции `TIME_TO_SEC()` и `SEC_TO_TIME()`, то такая возможность появится:

```
mysql> SELECT SEC_TO_TIME(TIME_TO_SEC('12:30:45'))+0;
+---------------------------------------+
| SEC_TO_TIME(TIME_TO_SEC('12:30:45'))+0 |
+---------------------------------------+
|                                123045 |
+---------------------------------------+
```

Преобразование значения в секунды и обратно не изменяет его, но заставляет MySQL воспринимать результат как значение типа `TIME`. Для значений дат делаем то же самое, только используем функции `TO_DAYS()` и `FROM_DAYS()`:

```
mysql> SELECT '1999-01-01'+0, FROM_DAYS(TO_DAYS('1999-01-01'))+0;
```

```
+----------------+-------------------------------------+
| '1999-01-01'+0 | FROM_DAYS(TO_DAYS('1999-01-01'))+0 |
+----------------+-------------------------------------+
|           1999 |                            19990101 |
+----------------+-------------------------------------+
```

Для интерпретации строк форматов DATETIME и TIMESTAMP можно прибегнуть к помощи функции DATE_ADD():

```
mysql> SELECT
    -> DATE_ADD('1999-01-01 12:30:45',INTERVAL 0 DAY)+0 AS 'numeric datetime',
    -> DATE_ADD('19990101123045',INTERVAL 0 DAY)+0 AS 'numeric timestamp';
+------------------+-------------------+
| numeric datetime | numeric timestamp |
+------------------+-------------------+
|   19990101123045 |    19990101123045 |
+------------------+-------------------+
```

5.30. Выбор записей по временным характеристикам

Задача

Вы хотите выбирать записи по их временным характеристикам.

Решение

Используйте условие для времени или даты в инструкции WHERE. Можно сравнивать непосредственно значения столбцов с известными значениями. А можно применить к значениям столбцов функцию, чтобы преобразовать их в более удобный для проверок формат, например, можно использовать MONTH() для проверки составляющей месяца дат.

Обсуждение

Большая часть изученных приемов работы со значениями дат иллюстрировалась примерами, выводящими значения даты и времени. Те же самые приемы можно использовать в инструкциях WHERE для наложения временных ограничений на записи, выбираемые запросом. Например, можно выбирать записи, относящиеся к датам, предшествующим заданной, входящим в определенный диапазон или соответствующим определенным значениям месяца или дня.

Сравнение дат друг с другом

Следующие запросы находят записи таблицы date_val, относящиеся к периоду до 1900 года или к 1900-м годам:

```
mysql> SELECT d FROM date_val where d < '1900-01-01';
+------------+
| d          |
+------------+
| 1864-02-28 |
```

```
+------------+
mysql> SELECT d FROM date_val where d BETWEEN '1900-01-01' AND '1999-12-31';
+------------+
| d          |
+------------+
| 1900-01-15 |
| 1987-03-05 |
| 1999-12-31 |
+------------+
```

Если вы работаете с более ранней, чем 3.23.9, версией MySQL, то инструкция BETWEEN может не всегда корректно работать со строковыми литералами дат, имеющими формат не-ISO. Например, может произойти сбой при выполнении такого запроса:

```
SELECT d FROM date_val WHERE d BETWEEN '1960-3-1' AND '1960-3-15';
```

Если это произошло, попробуйте преобразовать даты в формат ISO:

```
SELECT d FROM date_val WHERE d BETWEEN '1960-03-01' AND '1960-03-15';
```

Также можно переписать выражение, используя два явных сравнения:

```
SELECT d FROM date_val WHERE d >= '1960-03-01' AND d <= '1960-03-15';
```

Если нет точной даты, которую хотелось бы использовать в инструкции WHERE, ее можно вычислить при помощи выражения. Например, чтобы найти в таблице history записи о событиях, произошедших ровно 50 лет назад, выполним такой запрос:

```
SELECT * FROM history WHERE d = DATE_SUB(CURDATE(),INTERVAL 50 YEAR);
```

Такие сведения часто публикуются в газетных колонках типа «в этот день много лет назад» (на самом деле, запрос выводит те события, которые отпраздновали свой *n*-й юбилей). Если вы хотите извлечь события, которые случились «в этот день» не в каком-то конкретном году, а в любом, запрос будет немного другим. Необходимо найти записи, соответствующие данному календарному дню, без учета года. Решение такой задачи будет предложено в разделе «Сравнение дат с календарными днями» чуть далее в этом же рецепте.

Вычисленные даты можно использовать для проверки на вхождение в диапазон. Например, для нахождения дат последних шести лет используем функцию DATE_SUB() для вычисления граничной даты:

```
mysql> SELECT d FROM date_val WHERE d >= DATE_SUB(CURDATE(),INTERVAL 6 YEAR);
+------------+
| d          |
+------------+
| 1999-12-31 |
| 2000-06-04 |
+------------+
```

Обратите внимание на то, что в выражении инструкции WHERE столбец даты d расположен обособленно – он один находится слева от оператора сравнения. Если столбец индексирован, то обычно такая форма запроса обрабатывается

в MySQL наиболее эффективно. Можно было бы записать инструкцию WHERE другим способом, который логически эквивалентен предыдущему, но выполняется медленнее:

```
... WHERE DATE_ADD(d,INTERVAL 6 MONTH) >= CURDATE();
```

В данном случае столбец d используется внутри выражения, то есть извлекаться для вычисления и проверки выражения будет *каждая* строка – получается, что индекс не востребован.

Иногда не сразу понятно, как переформулировать сравнение так, чтобы столбец даты оказался изолированным по одну сторону от оператора сравнения. Например, такая инструкция WHERE использует в сравнении только часть столбца дат:

```
... WHERE YEAR(d) >= 1987 AND YEAR(d) <= 1991;
```

Чтобы изменить первое сравнение, уберем вызов YEAR() и заменим правую часть на полное значение даты:

```
... WHERE d >= '1987-01-01' AND YEAR(d) <= 1991;
```

Изменить второе сравнение несколько сложнее. Можно, как и в первом случае, избавиться от YEAR(), но нельзя просто добавить к году в правой части -01-01. Это приведет к следующему некорректному результату:

```
... WHERE d >= '1987-01-01' AND d <= '1991-01-01';
```

Замену нельзя считать успешной, так как новую проверку не пройдут даты с 1991-01-02 по 1991-12-31, удовлетворявшие условиям старого теста. Для того чтобы корректно переписать второе сравнение, нужно использовать один из вариантов:

```
... WHERE d >= '1987-01-01' AND d <= '1991-12-31';
... WHERE d >= '1987-01-01' AND d < '1992-01-01';
```

Еще одной областью применения вычисления дат являются приложения, создающие записи с ограниченным сроком жизни. Такие приложения должны уметь определять, какие записи следует удалить по истечении сроков. К решению задачи можно применить два подхода:

- Хранить для каждой записи дату ее создания (создайте столбец TIMESTAMP или используйте NOW(); подробности приведены в рецепте 5.33). При выполнении в дальнейшем операции удаления устаревших записей вы будете проверять, какие записи имеют слишком старую дату создания по сравнению с текущей датой. Например, запрос, аннулирующий записи, созданные более, чем *n* дней назад, мог бы выглядеть так:

  ```
  DELETE FROM имя_таблицы WHERE create_date < DATE_SUB(NOW(),INTERVAL n DAY);
  ```

- Хранить в каждой записи явную дату истечения срока хранения, вычисляя ее при помощи DATE_ADD() при создании записи. Запись, которая должна быть удалена через *n* дней, могла бы создаваться так:

  ```
  INSERT INTO имя_таблицы (expire_date,...)
  VALUES(DATE_ADD(NOW(),INTERVAL n DAY),...);
  ```

Чтобы выполнить операцию удаления устаревших записей, сравните дату истечения срока хранения записи с текущей датой:

```
DELETE FROM имя_таблицы WHERE expire_date < NOW()
```

Сравнение значений времени

Сравнения, в которых участвуют значения времени, аналогичны сравнениям дат. Например, чтобы найти значения времени, относящиеся к периоду с 9 часов утра до 2 часов дня, используем одно из выражений:

```
... WHERE t1 BETWEEN '09:00:00' AND '14:00:00';
... WHERE HOUR(t1) BETWEEN 9 AND 14;
```

Для индексированного столбца TIME эффективнее первый способ. Второй же хорош тем, что он работает не только для столбцов TIME, но и для DATETIME и TIMESTAMP.

Сравнение дат с календарными днями

Для ответов на вопросы об определенных днях года используйте проверку на календарные дни. Рассмотрим примеры, связанные с поиском дней рождений:

- У кого сегодня день рождения? Необходимо установить соответствие определенному календарному дню, поэтому извлекаем из даты составляющие месяца и дня, игнорируя год при выполнении сравнений:

  ```
  ... WHERE MONTH(d) = MONTH(CURDATE()) AND DAYOFMONTH(d) = DAYOFMONTH(CURDATE());
  ```

  Подобные запросы часто используются для биографических данных, например, для составления списка актеров, политиков, актеров, родившихся в определенный день года.

  Хочется попробовать использовать для решения задачи «в этот день» функцию DAYOFYEAR(), чтобы упростить запрос. Но DAYOFYEAR() плохо работает с високосными годами. Наличие 29 февраля портит значения дней с марта по декабрь.

- У кого день рождения в этом месяце? В данном случае будем проверять только месяц:

  ```
  ... WHERE MONTH(d) = MONTH(CURDATE());
  ```

- У кого будет день рождения в следующем месяце? Хитрость в том, что для получения значения месяца и проверки его на соответствие нельзя просто добавить единичку к текущему месяцу. Если текущий месяц – декабрь, вы получите тринадцатый месяц. Для того чтобы получить январь (первый месяц), воспользуйтесь одним из выражений:

  ```
  ... WHERE MONTH(d) = MONTH(DATE_ADD(CURDATE(),INTERVAL 1 MONTH));
  ... WHERE MONTH(d) = MOD(MONTH(CURDATE()),12)+1;
  ```

5.31. Использование значений TIMESTAMP

Задача

Вы хотите, чтобы при создании или изменении записи автоматически фиксировалось время этого события.

Решение

Используйте столбец типа TIMESTAMP. Но необходимо отметить, что он обладает некоторыми нетривиальными свойствами, так что этот раздел призван познакомить вас с тем, что вы получите. Следующие несколько разделов описывают возможности применения столбцов TIMESTAMP.

Обсуждение

MySQL поддерживает тип столбцов TIMESTAMP, во многом похожий на тип DATETIME. Однако у TIMESTAMP есть и некоторые особенности:

- Первый столбец TIMESTAMP особенным образом ведет себя при создании записи: его значение по умолчанию равно текущей дате и времени. То есть для него не нужно указывать значение в предложении INSERT, если вы хотите, чтобы в столбце хранилось время создания записи. MySQL автоматически инициализирует его. То же самое происходит при установке столбца в NULL при создании записи.
- Первый столбец TIMESTAMP обрабатывается особым образом и при любом изменении текущих значений строк. MySQL автоматически обновляет его значение, присваивая время и дату изменения. Запомните: обновление происходит, только если значение столбца действительно изменилось; если установлено прежнее значение, столбец TIMESTAMP не будет изменен.
- Другим столбцам TIMESTAMP таблицы не присущи свойства, описанные ранее. Их значение по умолчанию равно нулю, а не текущей дате и времени. Их значения не изменяются автоматически при изменении других столбцов, при необходимости вы должны изменять их самостоятельно.
- Столбец TIMESTAMP можно в любой момент установить в текущую дату и время путем установки его в NULL. Это касается всех столбцов TIMESTAMP, а не только первого.

Свойства TIMESTAMP, относящиеся к созданию и изменению записи, делают этот тип чрезвычайно удобным для решения некоторых видов задач, таких как автоматическая регистрация времени вставки или изменения строк таблицы. Но у этого типа есть и свойства, ограничивающие его применение:

- Значения TIMESTAMP имеют формат *CCYYMMDDhhmmss*, который интуитивно не вполне понятен и не очень удобен для восприятия, так что при выводе часто требуется переформатирование.
- Диапазон значений TIMESTAMP начинается в начале 1970 года и продолжается до 2037 года. Если вам необходим более широкий диапазон, используйте значения DATETIME.

В следующих разделах показано, как воспользоваться достоинствами типа TIMESTAMP.

5.32. Регистрация времени последнего изменения строки

Задача

Вы хотели бы автоматически регистрировать время последнего изменения записи.

Решение

Включите в таблицу столбец TIMESTAMP.

Обсуждение

Чтобы создать таблицу, каждая строка которой содержала бы значение, указывающее время последнего обновления записи, включите в нее столбец TIMESTAMP. При создании новой строки столбец будет устанавливаться в текущую дату и время и будет обновляться при каждом изменении значения другого столбца строки. Предположим, что вы создаете таблицу tsdemo1 со столбцом TIMESTAMP:

```
CREATE TABLE tsdemo1
(
    t   TIMESTAMP,
    val INT
);
```

Вставьте в таблицу две записи, а затем извлеките ее содержимое. (Выполните второе предложение INSERT через несколько секунд после первого, чтобы посмотреть, как отличаются временные метки.) В первом предложении INSERT показано, что вы можете установить столбец t в текущую дату и время, явно указав для него значение NULL; а во втором предложении t вообще не участвует в запросе:

```
mysql> INSERT INTO tsdemo1 (t,val) VALUES(NULL,5);
mysql> INSERT INTO tsdemo1 (val) VALUES(10);
mysql> SELECT * FROM tsdemo1;
+----------------+------+
| t              | val  |
+----------------+------+
| 20020715115825 |    5 |
| 20020715115831 |   10 |
+----------------+------+
```

Теперь выдадим запрос, изменяющий значение столбца val, и проверим, как это отразится на содержимом таблицы:

```
mysql> UPDATE tsdemo1 SET val = 6 WHERE val = 5;
```

```
mysql> SELECT * FROM tsdemo1;
+----------------+------+
| t              | val  |
+----------------+------+
| 20020715115915 |    6 |
| 20020715115831 |   10 |
+----------------+------+
```

Как видите, значение TIMESTAMP обновлено только для измененной записи.

Если вы обновляете несколько записей, будут изменены все соответствующие значения TIMESTAMP:

```
mysql> UPDATE tsdemo1 SET val = val + 1;
mysql> SELECT * FROM tsdemo1;
+----------------+------+
| t              | val  |
+----------------+------+
| 20020715115926 |    7 |
| 20020715115926 |   11 |
+----------------+------+
```

Если предложение UPDATE не изменяет значения столбца val, то и значения TIMESTAMP не изменяются. Давайте проверим это, установив каждую запись столбца val в его текущее значение:

```
mysql> UPDATE tsdemo1 SET val = val + 0;
mysql> SELECT * FROM tsdemo1;
+----------------+------+
| t              | val  |
+----------------+------+
| 20020715115926 |    7 |
| 20020715115926 |   11 |
+----------------+------+
```

Альтернативой TIMESTAMP является столбец типа DATETIME, явно устанавливаемый в NOW() при создании и изменении записей. Но в этом случае подобный алгоритм должен быть реализован во всех приложениях, использующих таблицу, иначе ничего не получится.

5.33. Регистрация времени создания записи

Задача

Вы хотите сохранить время создания записи, что можно было бы сделать при помощи TIMESTAMP, но вам нужно, чтобы это значение не менялось при изменении записи, а тип TIMESTAMP не умеет хранить значение.

Решение

На самом деле он все умеет, нужно только включить в таблицу второй столбец TIMESTAMP, имеющий другие свойства.

Обсуждение

Если вам нужен столбец, который бы изначально устанавливался в момент создания записи, а затем больше не менялся, один TIMESTAMP – это не решение, так как он обновляется при изменении других столбцов записи. Создадим два столбца TIMESTAMP и воспользуемся тем, что второй не обладает специальными свойствами первого. Оба столбца можно установить в текущую дату и время в момент создания записи. Далее при каждом изменении других столбцов записи первый столбец TIMESTAMP будет обновляться, а второй будет продолжать хранить время создания записи. Рассмотрим такую таблицу:

```
CREATE TABLE tsdemo2
(
    t_update    TIMESTAMP,  # время последнего изменения записи
    t_create    TIMESTAMP,  # время создания записи
    val INT
);
```

Создадим таблицу, затем вставим в нее запись, в которой оба столбца TIMESTAMP установлены в NULL (чтобы инициализировать их в текущую дату и время):

```
mysql> INSERT INTO tsdemo2 (t_update,t_create,val) VALUES(NULL,NULL,5);
mysql> SELECT * FROM tsdemo2;
+----------------+----------------+------+
| t_update       | t_create       | val  |
+----------------+----------------+------+
| 20020715120003 | 20020715120003 |    5 |
+----------------+----------------+------+
```

После добавления записи изменим столбец val, затем проверим, произошло ли обновление столбца t_update и остался ли неизменным и равным времени создания записи столбец t_create:

```
mysql> UPDATE tsdemo2 SET val = val + 1;
mysql> SELECT * FROM tsdemo2;
+----------------+----------------+------+
| t_update       | t_create       | val  |
+----------------+----------------+------+
| 20020715120012 | 20020715120003 |    6 |
+----------------+----------------+------+
```

Как и в случае с tsdemo1, обновления таблицы tsdemo2, которые в действительности не изменяют значения столбца, не вызывают изменения значений TIMESTAMP:

```
mysql> UPDATE tsdemo2 SET val = val + 0;
mysql> SELECT * FROM tsdemo2;
+----------------+----------------+------+
| t_update       | t_create       | val  |
+----------------+----------------+------+
| 20020715120012 | 20020715120003 |    6 |
+----------------+----------------+------+
```

Можно было использовать для столбцов t_create и t_update и тип DATETIME. При создании записи их оба нужно было бы явно установить в NOW(). При изменении записи t_update устанавливался бы в NOW(), а t_create оставался неизменным.

5.34. Вычисления со значениями TIMESTAMP

Задача

Вы хотите вычислить интервал между значениями TIMESTAMP, найти записи по столбцу TIMESTAMP и тому подобное.

Решение

Для значений TIMESTAMP возможны те же операции, что и для DATETIME: сравнение, сдвиг и извлечение составляющих.

Обсуждение

Для таблицы tsdemo2 из рецепта 5.33 рассмотрим запросы, выполняющие некоторые из операций, допустимых для значений TIMESTAMP:

- Записи, которые не обновлялись с момента создания:

```
SELECT * FROM tsdemo2 WHERE t_create = t_update;
```

- Записи, изменявшиеся в течение последних 12 часов:

```
SELECT * FROM tsdemo2 WHERE t_update >= DATE_SUB(NOW(),INTERVAL 12 HOUR);
```

- Интервал между временем изменения и создания (в часах и секундах):

```
SELECT t_create, t_update,
UNIX_TIMESTAMP(t_update) - UNIX_TIMESTAMP(t_create) AS 'seconds',
(UNIX_TIMESTAMP(t_update) - UNIX_TIMESTAMP(t_create))/(60 * 60) AS 'hours'
FROM tsdemo2;
```

- Записи, созданные с 13 до 16 часов:

```
SELECT * FROM tsdemo2
WHERE HOUR(t_create) BETWEEN 13 AND 16;
```

  или:

```
SELECT * FROM tsdemo2
WHERE DATE_FORMAT(t_create,'%H%i%s') BETWEEN '130000' AND '160000';
```

  Можно даже использовать TIME_TO_SEC() для отбрасывания составляющей даты значений t_create:

```
SELECT * FROM tsdemo2
WHERE TIME_TO_SEC(t_create)
BETWEEN TIME_TO_SEC('13:00:00') AND TIME_TO_SEC('16:00:00');
```

5.35. Вывод значений TIMESTAMP в удобном для чтения виде

Задача

Вам не нравится, как MySQL отображает значения TIMESTAMP.

Решение

Переформатируйте их при помощи функции DATE_FORMAT().

Обсуждение

Столбцы TIMESTAMP обладают рядом полезных свойств, но одно из свойств не так удачно, как другие, – формат вывода (*CCYYMMDDhhmmss*). Длинная непрерывная строка цифр несовместима с форматом DATETIME (*CCYY-MM-DD hh:mm:ss*) и неудобна для восприятия. Чтобы преобразовать значения TIMESTAMP в формат DATETIME, примените функцию DATE_FORMAT(). Следующий пример использует таблицу tsdemo2 из рецепта 5.33:

```
mysql> SELECT t_create, DATE_FORMAT(t_create,'%Y-%m-%d %T') FROM tsdemo2;
+----------------+-------------------------------------+
| t_create       | DATE_FORMAT(t_create,'%Y-%m-%d %T') |
+----------------+-------------------------------------+
| 20020715120003 | 2002-07-15 12:00:03                 |
+----------------+-------------------------------------+
```

Можно выполнить и обратное преобразование (вывести значения DATETIME в формате TIMESTAMP), хотя оно встречается довольно редко. Можно использовать DATE_FORMAT(); а можно просто добавить ноль:

```
mysql> SELECT dt,
    -> DATE_FORMAT(dt,'%Y%m%d%H%i%s'),
    -> dt+0
    -> FROM datetime_val;
+---------------------+--------------------------------+----------------+
| dt                  | DATE_FORMAT(dt,'%Y%m%d%H%i%s') | dt+0           |
+---------------------+--------------------------------+----------------+
| 1970-01-01 00:00:00 | 19700101000000                 | 19700101000000 |
| 1987-03-05 12:30:15 | 19870305123015                 | 19870305123015 |
| 1999-12-31 09:00:00 | 19991231090000                 | 19991231090000 |
| 2000-06-04 15:45:30 | 20000604154530                 | 20000604154530 |
+---------------------+--------------------------------+----------------+
```

Дополнительная информация о преобразовании значений времени в любой удобный формат приведена в рецепте 5.2.

6

Сортировка результатов запроса

6.0. Введение

Эта глава посвящена сортировке – операции, позволяющей управлять порядком вывода результатов предложений SELECT. Сортировка выполняется за счет добавления в запрос инструкции ORDER BY. Если такой инструкции нет, MySQL может возвращать строки в любом порядке; сортировка же помогает навести порядок во всем этом хаосе и упрощает работу с результатами. (При использовании инструкции GROUP BY выполняется неявная сортировка результатов – см. рецепт 6.1.)

Несколько примеров главы работают с таблицей driver_log, которая содержит столбцы для регистрации дневного пробега группы водителей грузовиков:

```
mysql> SELECT * FROM driver_log;
+--------+-------+------------+-------+
| rec_id | name  | trav_date  | miles |
+--------+-------+------------+-------+
|      1 | Ben   | 2001-11-30 |   152 |
|      2 | Suzi  | 2001-11-29 |   391 |
|      3 | Henry | 2001-11-29 |   300 |
|      4 | Henry | 2001-11-27 |    96 |
|      5 | Ben   | 2001-11-29 |   131 |
|      6 | Henry | 2001-11-26 |   115 |
|      7 | Suzi  | 2001-12-02 |   502 |
|      8 | Henry | 2001-12-01 |   197 |
|      9 | Ben   | 2001-12-02 |    79 |
|     10 | Henry | 2001-11-30 |   203 |
+--------+-------+------------+-------+
```

Во многих рецептах использована таблица mail (уже встречавшаяся в предыдущих главах):

```
mysql> SELECT * FROM mail;
+---------------------+---------+---------+---------+---------+---------+
| t                   | srcuser | srchost | dstuser | dsthost | size    |
+---------------------+---------+---------+---------+---------+---------+
| 2001-05-11 10:15:08 | barb    | saturn  | tricia  | mars    |   58274 |
```

```
| 2001-05-12 12:48:13 | tricia  | mars    | gene    | venus   |  194925 |
| 2001-05-12 15:02:49 | phil    | mars    | phil    | saturn  |    1048 |
| 2001-05-13 13:59:18 | barb    | saturn  | tricia  | venus   |     271 |
| 2001-05-14 09:31:37 | gene    | venus   | barb    | mars    |    2291 |
| 2001-05-14 11:52:17 | phil    | mars    | tricia  | saturn  |    5781 |
| 2001-05-14 14:42:21 | barb    | venus   | barb    | venus   |   98151 |
| 2001-05-14 17:03:01 | tricia  | saturn  | phil    | venus   | 2394482 |
| 2001-05-15 07:17:48 | gene    | mars    | gene    | saturn  |    3824 |
| 2001-05-15 08:50:57 | phil    | venus   | phil    | venus   |     978 |
| 2001-05-15 10:25:52 | gene    | mars    | tricia  | saturn  |  998532 |
| 2001-05-15 17:35:31 | gene    | saturn  | gene    | mars    |    3856 |
| 2001-05-16 09:00:28 | gene    | venus   | barb    | mars    |     613 |
| 2001-05-16 23:04:19 | phil    | venus   | barb    | venus   |   10294 |
| 2001-05-17 12:49:23 | phil    | mars    | tricia  | saturn  |     873 |
| 2001-05-19 22:21:51 | gene    | saturn  | gene    | venus   |   23992 |
+---------------------+---------+---------+---------+---------+---------+
```

Могут использоваться и другие таблицы. Большинство из них можно создать при помощи сценариев из каталога *tables* дистрибутива `recipes`. Каталог *baseball1* содержит инструкции по созданию таблиц для примеров, связанных с бейсбольной базой данных *baseball1.com*.

6.1. Использование ORDER BY для сортировки результатов запроса

Задача

Результаты запроса выводятся не в том порядке, в котором хотелось бы.

Решение

Добавьте в запрос инструкцию `ORDER BY`.

Обсуждение

Содержимое таблиц `driver_log` и `mail`, приведенных во введении, не упорядочено и неудобно для восприятия. Исключением являются столбцы `id` и `t`, в которых значения расположены по порядку, но это простое совпадение. Обычно строки возвращаются в том же порядке, что и вставлялись, но только до тех пор, пока к таблице не будет применена операция обновления или удаления. Строки, добавленные последними, вполне могут быть возвращены в середине результирующего множества. Многие пользователи MySQL обращают внимание на такое нарушение порядка извлечения строк и задаются вопросом: «Как хранить строки в таблице так, чтобы они извлекались в определенном порядке?». Но этот вопрос некорректен. Хранением строк занимается сервер, и мы в это не вмешиваемся. (Кроме того, даже если бы вы могли задать порядок хранения, чем бы это вам помогло, если бы нужно было упорядочивать результат каждый раз по-разному?)

Когда вы выбираете записи, сервер извлекает их из базы данных и возвращает в произвольном порядке. Порядок извлечения строк может меняться в зависимости от того, какой индекс сервер использует при выполнении запроса. Даже если ваши строки естественным образом выдаются в нужном порядке, никаких гарантий относительно порядка извлечения строк реляционная база данных не дает, до тех пор, пока вы явно не укажете ей, каким он должен быть. Чтобы расположить строки результата запроса в определенном порядке, отсортируйте их, добавив инструкцию `ORDER BY` в предложение `SELECT`. Если инструкция `ORDER BY` отсутствует, то порядок извлечения строк может измениться при изменении содержимого таблицы. Если же такая инструкция присутствует, MySQL всегда будет выводить строки в заданном порядке.

Инструкция `ORDER BY` имеет следующие характеристики:

- Можно выполнять сортировку по одному или нескольким столбцам значений.
- Любой столбец может быть упорядочен по возрастанию (по умолчанию) или по убыванию.
- На столбцы можно ссылаться по имени, по их позиции в списке вывода или с помощью псевдонимов.

В разделе представлено несколько базовых приемов сортировки. В следующих рецептах будут описаны более сложные случаи. Как ни парадоксально это звучит, можно применять `ORDER BY` и для «разупорядочивания» результирующего множества. Такая возможность используется для расположения строк в случайном порядке или (в сочетании с `LIMIT`) для случайной выборки строки из результирующего множества (см. главу 13).

Указание столбцов и направления сортировки

Рассмотрим ряд примеров, в которых выполняется сортировка по одному и нескольким столбцам, в порядке убывания и возрастания. Примеры выбирают строки из таблицы `driver_log` и сортируют их в разном порядке, что дает возможность сравнить действие различных инструкций `ORDER BY`.

Запрос производит сортировку по одному столбцу – имени водителя:

```
mysql> SELECT * FROM driver_log ORDER BY name;
+--------+-------+------------+-------+
| rec_id | name  | trav_date  | miles |
+--------+-------+------------+-------+
|      1 | Ben   | 2001-11-30 |   152 |
|      5 | Ben   | 2001-11-29 |   131 |
|      9 | Ben   | 2001-12-02 |    79 |
|      3 | Henry | 2001-11-29 |   300 |
|      4 | Henry | 2001-11-27 |    96 |
|      6 | Henry | 2001-11-26 |   115 |
|      8 | Henry | 2001-12-01 |   197 |
|     10 | Henry | 2001-11-30 |   203 |
|      2 | Suzi  | 2001-11-29 |   391 |
|      7 | Suzi  | 2001-12-02 |   502 |
+--------+-------+------------+-------+
```

По умолчанию строки упорядочиваются по возрастанию. Можно задать упорядочивание по возрастанию и явно, добавив ASC после названия сортируемого столбца:

```
SELECT * FROM driver_log ORDER BY name ASC;
```

Возможна сортировка в обратном (или противоположном) по отношению к упорядочиванию по возрастанию порядке – по убыванию, которая задается при помощи добавления DESC после названия сортируемого столбца:

```
mysql> SELECT * FROM driver_log ORDER BY name DESC;
+--------+-------+------------+-------+
| rec_id | name  | trav_date  | miles |
+--------+-------+------------+-------+
|      2 | Suzi  | 2001-11-29 |   391 |
|      7 | Suzi  | 2001-12-02 |   502 |
|      3 | Henry | 2001-11-29 |   300 |
|      4 | Henry | 2001-11-27 |    96 |
|      6 | Henry | 2001-11-26 |   115 |
|      8 | Henry | 2001-12-01 |   197 |
|     10 | Henry | 2001-11-30 |   203 |
|      1 | Ben   | 2001-11-30 |   152 |
|      5 | Ben   | 2001-11-29 |   131 |
|      9 | Ben   | 2001-12-02 |    79 |
+--------+-------+------------+-------+
```

Если вы внимательно посмотрите на вывод только что приведенных запросов, то заметите, что, хотя строки и упорядочены по именам, но если для одного имени есть несколько строк, они никак не упорядочены (например, для Henry и Ben значения trav_date не отсортированы по дате). Все потому, что MySQL выполняет только те сортировки, которые явно указаны:

- Порядок строк в целом не определен до тех пор, пока не добавлена инструкция ORDER BY.
- Аналогично для группы строк, имеющих одно и то же значение столбца сортировки, порядок значений в других столбцах также не определен до тех пор, пока вы не укажете их в инструкции ORDER BY.

Чтобы обеспечить более полное управление выводом, укажите сортировку по нескольким столбцам, перечислив их через запятую. Следующий запрос упорядочивает строки по возрастанию значений столбца name (имя), а для каждого имени – по trav_date (дата):

```
mysql> SELECT * FROM driver_log ORDER BY name, trav_date;
+--------+-------+------------+-------+
| rec_id | name  | trav_date  | miles |
+--------+-------+------------+-------+
|      5 | Ben   | 2001-11-29 |   131 |
|      1 | Ben   | 2001-11-30 |   152 |
|      9 | Ben   | 2001-12-02 |    79 |
|      6 | Henry | 2001-11-26 |   115 |
|      4 | Henry | 2001-11-27 |    96 |
|      3 | Henry | 2001-11-29 |   300 |
```

```
|     10 | Henry | 2001-11-30 |   203 |
|      8 | Henry | 2001-12-01 |   197 |
|      2 | Suzi  | 2001-11-29 |   391 |
|      7 | Suzi  | 2001-12-02 |   502 |
+--------+-------+------------+-------+
```

Можно сортировать несколько столбцов и по убыванию, но тогда необходимо добавлять DESC после имени каждого столбца, который должен быть упорядочен таким образом:

```
mysql> SELECT * FROM driver_log ORDER BY name DESC, trav_date DESC;
+--------+-------+------------+-------+
| rec_id | name  | trav_date  | miles |
+--------+-------+------------+-------+
|      7 | Suzi  | 2001-12-02 |   502 |
|      2 | Suzi  | 2001-11-29 |   391 |
|      8 | Henry | 2001-12-01 |   197 |
|     10 | Henry | 2001-11-30 |   203 |
|      3 | Henry | 2001-11-29 |   300 |
|      4 | Henry | 2001-11-27 |    96 |
|      6 | Henry | 2001-11-26 |   115 |
|      9 | Ben   | 2001-12-02 |    79 |
|      1 | Ben   | 2001-11-30 |   152 |
|      5 | Ben   | 2001-11-29 |   131 |
+--------+-------+------------+-------+
```

При сортировке по нескольким столбцам вы можете задать для одних упорядочивание по возрастанию, а для других – по убыванию. Следующий запрос упорядочивает строки по убыванию значений столбца name, а затем для каждого имени – по возрастанию значений столбца trav_date:

```
mysql> SELECT * FROM driver_log ORDER BY name DESC, trav_date;
+--------+-------+------------+-------+
| rec_id | name  | trav_date  | miles |
+--------+-------+------------+-------+
|      2 | Suzi  | 2001-11-29 |   391 |
|      7 | Suzi  | 2001-12-02 |   502 |
|      6 | Henry | 2001-11-26 |   115 |
|      4 | Henry | 2001-11-27 |    96 |
|      3 | Henry | 2001-11-29 |   300 |
|     10 | Henry | 2001-11-30 |   203 |
|      8 | Henry | 2001-12-01 |   197 |
|      5 | Ben   | 2001-11-29 |   131 |
|      1 | Ben   | 2001-11-30 |   152 |
|      9 | Ben   | 2001-12-02 |    79 |
+--------+-------+------------+-------+
```

Ссылки на столбцы сортировки

Инструкции ORDER BY уже рассмотренных запросов ссылаются на столбцы сортировки по имени. Можно также использовать для ссылок позиции столбцов в списке вывода или псевдонимы (alias). Нумерация позиций начинается с 1. Упорядочим результирующее множество по третьему столбцу miles:

```
mysql> SELECT name, trav_date, miles FROM driver_log ORDER BY 3;
+-------+------------+-------+
| name  | trav_date  | miles |
+-------+------------+-------+
| Ben   | 2001-12-02 |    79 |
| Henry | 2001-11-27 |    96 |
| Henry | 2001-11-26 |   115 |
| Ben   | 2001-11-29 |   131 |
| Ben   | 2001-11-30 |   152 |
| Henry | 2001-12-01 |   197 |
| Henry | 2001-11-30 |   203 |
| Henry | 2001-11-29 |   300 |
| Suzi  | 2001-11-29 |   391 |
| Suzi  | 2001-12-02 |   502 |
+-------+------------+-------+
```

Если столбец вывода имеет псевдоним, вы можете указать его в инструкции ORDER BY:

```
mysql> SELECT name, trav_date, miles AS distance FROM driver_log
    -> ORDER BY distance;
+-------+------------+----------+
| name  | trav_date  | distance |
+-------+------------+----------+
| Ben   | 2001-12-02 |       79 |
| Henry | 2001-11-27 |       96 |
| Henry | 2001-11-26 |      115 |
| Ben   | 2001-11-29 |      131 |
| Ben   | 2001-11-30 |      152 |
| Henry | 2001-12-01 |      197 |
| Henry | 2001-11-30 |      203 |
| Henry | 2001-11-29 |      300 |
| Suzi  | 2001-11-29 |      391 |
| Suzi  | 2001-12-02 |      502 |
+-------+------------+----------+
```

Стоит ли сортировать результаты запроса самостоятельно?

Если вы выполняете запрос SELECT в программе, то можете извлекать неупорядоченный результат в структуру данных, а затем упорядочивать ее с помощью средств языка программирования. Но зачем изобретать велосипед? Сервер MySQL отлично справляется с сортировкой, так что вы вполне можете на него положиться.

Исключением может быть тот случай, когда один набор строк следует упорядочить несколькими разными способами. Тогда вместо того чтобы создавать несколько запросов, отличающихся только инструкциями ORDER BY, вероятно, эффективнее один раз извлечь строки в программе, а затем упорядочивать их как угодно.

Псевдонимы имеют преимущество перед столбцами, указанными номерами позиций в инструкции ORDER BY. Если вы выполняете сортировку, указывая позицию столбца в списке вывода, то при изменении этого списка вам может понадобиться изменить номера позиций столбцов в инструкции ORDER BY. Если вы используете псевдонимы столбцов, в этом нет необходимости. (К сожалению, некоторые процессоры баз данных не поддерживают псевдонимы столбцов в инструкции ORDER BY, так что эта функциональность непереносима.)

Как и столбцы, указанные по имени, столбцы, указанные номерами позиций или псевдонимами, можно упорядочить по убыванию и возрастанию:

```
mysql> SELECT name, trav_date, miles FROM driver_log ORDER BY 3 DESC;
+-------+------------+-------+
| name  | trav_date  | miles |
+-------+------------+-------+
| Suzi  | 2001-12-02 |   502 |
| Suzi  | 2001-11-29 |   391 |
| Henry | 2001-11-29 |   300 |
| Henry | 2001-11-30 |   203 |
| Henry | 2001-12-01 |   197 |
| Ben   | 2001-11-30 |   152 |
| Ben   | 2001-11-29 |   131 |
| Henry | 2001-11-26 |   115 |
| Henry | 2001-11-27 |    96 |
| Ben   | 2001-12-02 |    79 |
+-------+------------+-------+
```

6.2. Сортировка частей таблицы

Задача

Вы хотите упорядочить не всю таблицу, а только ее часть.

Решение

Добавьте инструкцию WHERE, которая будет отбирать только интересующие вас записи.

Обсуждение

Инструкция ORDER BY не учитывает количество строк, она сортирует все, что возвращает запрос. Если вы не хотите упорядочивать всю таблицу, добавьте в запрос инструкцию WHERE, указывающую строки для сортировки. Например, чтобы упорядочить строки только для одного водителя, сделайте, например, так:

```
mysql> SELECT trav_date, miles FROM driver_log WHERE name = 'Henry'
    -> ORDER BY trav_date;
+------------+-------+
| trav_date  | miles |
+------------+-------+
| 2001-11-26 |   115 |
```

```
| 2001-11-27 |    96 |
| 2001-11-29 |   300 |
| 2001-11-30 |   203 |
| 2001-12-01 |   197 |
+------------+-------+
```

Как видно из запроса, столбцы, указанные в инструкции ORDER BY, не обязаны совпадать со столбцами инструкции WHERE. Более того, они даже не обязаны выводиться, но об этом поговорим позже (рецепт 6.4).

6.3. Сортировка результатов выражения

Задача

Вы хотите упорядочить результат запроса на основе вычисления значений некоторых выражений над столбцами.

Решение

Поместите выражение, вычисляющее значение, в инструкцию ORDER BY. Если вы работаете со старой версией MySQL, не поддерживающей выражения в ORDER BY, используйте обходной маневр.

Обсуждение

Один из столбцов таблицы mail содержит величину сообщения в байтах:

```
mysql> SELECT * FROM mail;
+---------------------+---------+---------+---------+---------+---------+
| t                   | srcuser | srchost | dstuser | dsthost | size    |
+---------------------+---------+---------+---------+---------+---------+
| 2001-05-11 10:15:08 | barb    | saturn  | tricia  | mars    |   58274 |
| 2001-05-12 12:48:13 | tricia  | mars    | gene    | venus   |  194925 |
| 2001-05-12 15:02:49 | phil    | mars    | phil    | saturn  |    1048 |
| 2001-05-13 13:59:18 | barb    | saturn  | tricia  | venus   |     271 |
...
```

Предположим, что вы хотите извлекать только «большие» сообщения (например, превышающие 50 000 байт), но при этом выводить и упорядочивать их в терминах килобайтов, а не байтов. Вы можете применить для сортировки результатов выражения инструкцию ORDER BY, если это допускает ваша версия MySQL.

Выражения в инструкции ORDER BY не разрешены в версиях MySQL, предшествующих 3.23.2. Чтобы обойти это ограничение, укажите выражение в списке столбцов вывода и сошлитесь на него либо позиционно, либо используя псевдоним:[1]

[1] Вас удивляет +1023 в выражении FLOOR()? Именно так значения size группируются по ближайшей верхней границе 1024-байтных интервалов. В противном случае (без такой добавки) они группировались бы по нижней границе (например, послание из 2047 байт было бы представлено как имеющее размер 1 Кбайт, а не 2 Кбайт). Подробнее об этой методике будет рассказано в рецепте 7.12.

```
mysql> SELECT t, srcuser, FLOOR((size+1023)/1024)
    -> FROM mail WHERE size > 50000
    -> ORDER BY 3;
+---------------------+---------+-------------------------+
| t                   | srcuser | FLOOR((size+1023)/1024) |
+---------------------+---------+-------------------------+
| 2001-05-11 10:15:08 | barb    |                      57 |
| 2001-05-14 14:42:21 | barb    |                      96 |
| 2001-05-12 12:48:13 | tricia  |                     191 |
| 2001-05-15 10:25:52 | gene    |                     976 |
| 2001-05-14 17:03:01 | tricia  |                    2339 |
+---------------------+---------+-------------------------+
mysql> SELECT t, srcuser, FLOOR((size+1023)/1024) AS kilobytes
    -> FROM mail WHERE size > 50000
    -> ORDER BY kilobytes;
+---------------------+---------+-----------+
| t                   | srcuser | kilobytes |
+---------------------+---------+-----------+
| 2001-05-11 10:15:08 | barb    |        57 |
| 2001-05-14 14:42:21 | barb    |        96 |
| 2001-05-12 12:48:13 | tricia  |       191 |
| 2001-05-15 10:25:52 | gene    |       976 |
| 2001-05-14 17:03:01 | tricia  |      2339 |
+---------------------+---------+-----------+
```

Такой способ будет работать и для версий MySQL 3.23.2 и выше, но в них появляется дополнительная возможность размещения выражения непосредственно в инструкции `ORDER BY`:

```
mysql> SELECT t, srcuser, FLOOR((size+1023)/1024)
    -> FROM mail WHERE size > 50000
    -> ORDER BY FLOOR((size+1023)/1024);
+---------------------+---------+-------------------------+
| t                   | srcuser | FLOOR((size+1023)/1024) |
+---------------------+---------+-------------------------+
| 2001-05-11 10:15:08 | barb    |                      57 |
| 2001-05-14 14:42:21 | barb    |                      96 |
| 2001-05-12 12:48:13 | tricia  |                     191 |
| 2001-05-15 10:25:52 | gene    |                     976 |
| 2001-05-14 17:03:01 | tricia  |                    2339 |
+---------------------+---------+-------------------------+
```

Однако даже если вы помещаете выражение в инструкцию `ORDER BY`, есть по крайней мере две причины для того, чтобы использовать для него псевдоним:

- Легче написать инструкцию `ORDER BY`, используя псевдоним, чем заново вводя выражение (которое может быть весьма громоздким).
- Псевдоним может быть удобен при выводе – столбец получает понятное и осмысленное название.

Такое же ограничение накладывается на выражения в инструкции `GROUP BY` (до которой мы доберемся в главе 7), и обходить его следует так же. Не забудьте

об этом, если ваша версия MySQL старше, чем 3.23.2. Многие запросы далее в книге используют выражения в инструкциях ORDER BY и GROUP BY. Чтобы они заработали на ранних версиях сервера MySQL, вам придется переписать их, применяя только что изученный прием.

6.4. Сортировка одного набора значений и вывод другого

Задача

Вы хотите упорядочить результирующее множество, используя значения, которые запросом не выбираются.

Решение

Никаких проблем. Вы можете использовать в инструкции ORDER BY столбцы, которые не указаны в списке вывода запроса.

Обсуждение

Инструкция ORDER BY может упорядочивать не только столбцы, перечисленные в списке вывода, но и «скрытые» (невыводимые) значения. Такой прием, как правило, используется, когда у вас есть значения, которые могут быть представлены несколькими способами, а вы хотите отображать один тип значений, а сортировать по другому. Например, вы можете захотеть вывести размеры почтовых сообщений не как количество байт (числа), а как строки, то есть 103K для 103 Кбайт. Для преобразования количества байтов в такую строку используйте следующее выражение:

```
CONCAT(FLOOR((size+1023)/1024),'K')
```

Получившиеся значения – это строки, поэтому они упорядочиваются в лексическом, а не в числовом порядке. Если вы выполните для них сортировку, то значение 96K будет стоять после 2339K, несмотря на то, что представляет меньший размер:

```
mysql> SELECT t, srcuser,
    -> CONCAT(FLOOR((size+1023)/1024),'K') AS size_in_K
    -> FROM mail WHERE size > 50000
    -> ORDER BY size_in_K;
+---------------------+---------+-----------+
| t                   | srcuser | size_in_K |
+---------------------+---------+-----------+
| 2001-05-12 12:48:13 | tricia  | 191K      |
| 2001-05-14 17:03:01 | tricia  | 2339K     |
| 2001-05-11 10:15:08 | barb    | 57K       |
| 2001-05-14 14:42:21 | barb    | 96K       |
| 2001-05-15 10:25:52 | gene    | 976K      |
+---------------------+---------+-----------+
```

Чтобы получить тот результат, который нам нужен, будем выводить строку, а для сортировки использовать числовое значение размера:

```
mysql> SELECT t, srcuser,
    -> CONCAT(FLOOR((size+1023)/1024),'K') AS size_in_K
    -> FROM mail WHERE size > 50000
    -> ORDER BY size;
+---------------------+---------+-----------+
| t                   | srcuser | size_in_K |
+---------------------+---------+-----------+
| 2001-05-11 10:15:08 | barb    | 57K       |
| 2001-05-14 14:42:21 | barb    | 96K       |
| 2001-05-12 12:48:13 | tricia  | 191K      |
| 2001-05-15 10:25:52 | gene    | 976K      |
| 2001-05-14 17:03:01 | tricia  | 2339K     |
+---------------------+---------+-----------+
```

Вывод в виде строк значений, отсортированных как числа, может помочь в ряде затруднительных ситуаций. Членам спортивных команд обычно присваивают номер, который присутствует у них на форме. Первое, что приходит в голову, – хранить его в числовом столбце. Но не торопитесь! Некоторым нравится номер ноль (0), а некоторым – двойной ноль (00). Если два таких игрока встретятся в одной команде, вы не сможете хранить их номера в числовом столбце, поскольку значения будут трактоваться как одинаковые. Поэтому следует хранить номера как строки:

```
CREATE TABLE roster
(
    name        CHAR(30),       # имя игрока
    jersey_num  CHAR(3)         # номер на футболке
);
```

Тогда номера будут отображаться так, как вводятся, при этом 0 и 00 будут восприниматься как разные значения. К сожалению, хотя представление чисел в виде строк и решает проблему распознавания 0 и 00, оно вызывает проблемы другого рода. Пусть в команде есть такие игроки:

```
mysql> SELECT name, jersey_num FROM roster;
+-----------+------------+
| name      | jersey_num |
+-----------+------------+
| Lynne     | 29         |
| Ella      | 0          |
| Elizabeth | 100        |
| Nancy     | 00         |
| Jean      | 8          |
| Sherry    | 47         |
+-----------+------------+
```

Проблема возникает, когда вы пытаетесь упорядочить членов команды по номеру. Если номера хранятся как строки, они будут отсортированы в лексическом порядке, который часто отличается от числового. Для игроков нашей команды это так и есть:

```
mysql> SELECT name, jersey_num FROM roster ORDER BY jersey_num;
+-----------+------------+
| name      | jersey_num |
+-----------+------------+
| Ella      | 0          |
| Nancy     | 00         |
| Elizabeth | 100        |
| Lynne     | 29         |
| Sherry    | 47         |
| Jean      | 8          |
+-----------+------------+
```

Значения 100 и 8 стоят явно не на своих местах. Но здесь нет ничего сложного. Выводите строковые значения, но для сортировки используйте числа. Сложите значения jersey_num с нулем, чтобы вызвать преобразование строк в числа:

```
mysql> SELECT name, jersey_num FROM roster ORDER BY jersey_num+0;
+-----------+------------+
| name      | jersey_num |
+-----------+------------+
| Ella      | 0          |
| Nancy     | 00         |
| Jean      | 8          |
| Lynne     | 29         |
| Sherry    | 47         |
| Elizabeth | 100        |
+-----------+------------+
```

Методику вывода одного значения и сортировки по другому также удобно использовать при выводе композитных значений, составленных из нескольких столбцов, которые упорядочиваются не так, как хотелось бы. Например, таблица mail представляет данные об отправителях сообщений в двух отдельных столбцах: srcuser и srchost. Если вы хотите вывести для отправителей адреса в формате srcuser@srchost, то можете получить такие значения, используя выражение:

```
CONCAT(srcuser,'@',srchost)
```

Но эти значения не удобны для сортировки, если название хоста для вас важнее, чем имя пользователя (которое стоит первым). Будем упорядочивать не по составному значению, а по значению базового столбца:

```
mysql> SELECT t, CONCAT(srcuser,'@',srchost) AS sender, size
    -> FROM mail WHERE size > 50000
    -> ORDER BY srchost, srcuser;
+---------------------+---------------+---------+
| t                   | sender        | size    |
+---------------------+---------------+---------+
| 2001-05-15 10:25:52 | gene@mars     |  998532 |
| 2001-05-12 12:48:13 | tricia@mars   |  194925 |
| 2001-05-11 10:15:08 | barb@saturn   |   58274 |
| 2001-05-14 17:03:01 | tricia@saturn | 2394482 |
| 2001-05-14 14:42:21 | barb@venus    |   98151 |
+---------------------+---------------+---------+
```

То же самое часто проделывают с именами людей. Пусть есть таблица `names`, содержащая имена и фамилии. Если значения столбцов выводятся по отдельности, то выполнить сортировку сначала по фамилии, а затем по имени просто:

```
mysql> SELECT last_name, first_name FROM name
    -> ORDER BY last_name, first_name;
+-----------+------------+
| last_name | first_name |
+-----------+------------+
| Blue      | Vida       |
| Brown     | Kevin      |
| Gray      | Pete       |
| White     | Devon      |
| White     | Rondell    |
+-----------+------------+
```

Если же вы хотите выводить для каждого человека строку в виде «имя-пробел-фамилия», то можете начать запрос так:

```
SELECT CONCAT(first_name,' ',last_name) AS full_name FROM name ...
```

Но как теперь упорядочить записи так, чтобы они появлялись в алфавитном порядке фамилий? Будем выводить составные значения, а в инструкции `ORDER BY` ссылаться на составляющие этих значений:

```
mysql> SELECT CONCAT(first_name,' ',last_name) AS full_name
    -> FROM name
    -> ORDER BY last_name, first_name;
+---------------+
| full_name     |
+---------------+
| Vida Blue     |
| Kevin Brown   |
| Pete Gray     |
| Devon White   |
| Rondell White |
+---------------+
```

Если вы хотите, чтобы в запросе осуществлялась сортировка по невыводимым значениям, то можете испытать затруднения в случае, если столбцами сортировки являются выражения, а вы работаете со старой версией MySQL. Дело в том, что выражения в инструкции `ORDER BY` разрешены только начиная с версии MySQL 3.23.2 (см. рецепт 6.3).

Для того чтобы обойти это ограничение, покажем скрытый элемент – выражение, добавив его в список столбцов вывода, и будем ссылаться на него по номеру позиции столбца или при помощи псевдонима. Например, чтобы написать запрос, выводящий имена из таблицы `names`, начиная с самого длинного, в MySQL версии 3.23.2 и выше поступим так:

```
mysql> SELECT CONCAT(first_name,' ',last_name) AS name
    -> FROM names
```

```
    -> ORDER BY LENGTH(CONCAT(first_name,' ',last_name)) DESC;
+---------------+
| name          |
+---------------+
| Rondell White |
| Kevin Brown   |
| Devon White   |
| Vida Blue     |
| Pete Gray     |
+---------------+
```

Чтобы переписать запрос для более ранних версий MySQL, поместим выражение в список столбцов вывода и используем для него псевдоним:

```
mysql> SELECT CONCAT(first_name,' ',last_name) AS name,
    -> LENGTH(CONCAT(first_name,' ',last_name)) AS len
    -> FROM names
    -> ORDER BY len DESC;
+---------------+------+
| name          | len  |
+---------------+------+
| Rondell White |   13 |
| Kevin Brown   |   11 |
| Devon White   |   11 |
| Vida Blue     |    9 |
| Pete Gray     |    9 |
+---------------+------+
```

Или сошлемся на дополнительный столбец вывода по номеру позиции:

```
mysql> SELECT CONCAT(first_name,' ',last_name) AS name,
    -> LENGTH(CONCAT(first_name,' ',last_name)) AS len
    -> FROM names
    -> ORDER BY 2 DESC;
+---------------+------+
| name          | len  |
+---------------+------+
| Rondell White |   13 |
| Kevin Brown   |   11 |
| Devon White   |   11 |
| Vida Blue     |    9 |
| Pete Gray     |    9 |
+---------------+------+
```

Какой бы способ ссылки вы ни выбрали, в выводе будет присутствовать столбец, который нужен там только для обеспечения возможности сортировки, а на самом деле вы абсолютно не заинтересованы в его выводе. Если вы выполняете запрос в программе *mysql*, то, к сожалению, ничего не можете сделать с этим дополнительным столбцом вывода. В ваших же собственных программах этот столбец не создает проблем. Да, он будет возвращен в результирующем множестве, но вы можете его игнорировать. Проиллюстри-

руем вышесказанное примером программы на Python. Выполняется запрос, имена отображаются, а длины имен отбрасываются:

```
cursor = conn.cursor (MySQLdb.cursors.DictCursor)
cursor.execute ("""
        SELECT CONCAT(first_name,' ',last_name) AS full_name,
        LENGTH(CONCAT(first_name,' ',last_name)) AS len
        FROM name
        ORDER BY len DESC
                """)
for row in cursor.fetchall ():
    print row["full_name"]  # вывести имя, игнорировать длину
cursor.close ()
```

6.5. Сортировка и значения NULL

Задача

Вы хотите упорядочить столбец, который может содержать значения NULL.

Решение

Расположение значений NULL в упорядоченном списке менялось с течением времени и зависит от используемой вами версии MySQL. Если значения NULL появляются не там, где хотелось бы, измените их местоположение принудительно.

Обсуждение

Если упорядоченный столбец содержит значения NULL, MySQL группирует их и помещает все вместе. Учитывая то, что в операциях сравнения (как видно из следующего запроса) значения NULL считаются не равными друг другу, такое поведение MySQL может показаться странным:

```
mysql> SELECT NULL = NULL;
+-------------+
| NULL = NULL |
+-------------+
|        NULL |
+-------------+
```

С другой стороны, значения NULL все же больше похожи друг на друга, чем на значения не-NULL, и, в любом случае, хорошего способа различать значения NULL пока не предложено. Итак, значения NULL группируются вместе и могут помещаться или в начало, или в конец отсортированного списка, в зависимости от используемой версии MySQL. До версии 4.0.2 значения NULL располагались в начале списка (или в конце, при сортировке по убыванию). Начиная с версии 4.0.2, MySQL упорядочивает значения NULL согласно спецификации ANSI SQL и всегда помещает их в начало списка, вне зависимости от направления сортировки.

Если вы хотите поместить значения NULL в тот или другой конец упорядоченного вывода, то можете сделать это вне зависимости от того, с какой версией MySQL работаете. Пусть есть таблица t, имеющая такое содержимое:

```
mysql> SELECT val FROM t;
+------+
| val  |
+------+
|    3 |
|  100 |
| NULL |
| NULL |
|    9 |
+------+
```

Обычно сортировка располагает значения NULL в начале списка:

```
mysql> SELECT val FROM t ORDER BY val;
+------+
| val  |
+------+
| NULL |
| NULL |
|    3 |
|    9 |
|  100 |
+------+
```

Чтобы вместо этого выводить их в конце списка, укажите в инструкции ORDER BY дополнительный столбец, который будет сопоставлять значениям NULL значение, превышающее значения не-NULL:

```
mysql> SELECT val FROM t ORDER BY IF(val IS NULL,1,0), val;
+------+
| val  |
+------+
|    3 |
|    9 |
|  100 |
| NULL |
| NULL |
+------+
```

То же самое возможно и для DESC:

```
mysql> SELECT val FROM t ORDER BY IF(val IS NULL,1,0), val DESC;
+------+
| val  |
+------+
|  100 |
|    9 |
|    3 |
| NULL |
| NULL |
+------+
```

Если оказывается, что MySQL помещает значения NULL в конец упорядоченного списка, а вам бы хотелось, чтобы они были в начале, используйте тот же прием, только поменяйте местами второй и третий аргументы функции IF(), чтобы сопоставлять значениям NULL значение меньшее, чем значения не-NULL:

```
IF(val IS NULL,0,1)
```

6.6. Сортировка и чувствительность к регистру

Задача

Сортировка строк чувствительна к регистру тогда, когда вы этого не хотите, и наоборот.

Решение

Измените чувствительность к регистру упорядочиваемых значений.

Обсуждение

В главе 4 рассказывалось о том, что двоичные строки чувствительны к регистру, а недвоичные – не чувствительны. Это свойство переносится и на операцию сортировки строк: ORDER BY выполняет сортировку в лексическом порядке, при этом для двоичных строк операция чувствительна к регистру, а для недвоичных – не чувствительна. Рассмотрим для наглядности таблицу text-blob_val, которая содержит столбец tstr типа TEXT и столбец bstr типа BLOB:

```
mysql> SELECT * FROM textblob_val;
+------+------+
| tstr | bstr |
+------+------+
| aaa  | aaa  |
| AAA  | AAA  |
| bbb  | bbb  |
| BBB  | BBB  |
+------+------+
```

Столбцы содержат одинаковые значения, но упорядочены будут по-разному, поскольку столбцы TEXT не чувствительны к регистру, а столбцы BLOB – чувствительны:

```
mysql> SELECT tstr FROM textblob_val ORDER BY tstr;
+------+
| tstr |
+------+
| aaa  |
| AAA  |
| bbb  |
| BBB  |
+------+
mysql> SELECT bstr FROM textblob_val ORDER BY bstr;
```

```
+------+
| bstr |
+------+
| AAA  |
| BBB  |
| aaa  |
| bbb  |
+------+
```

Чтобы регулировать чувствительность к регистру инструкций ORDER BY, используем приемы, представленные в главе 4 при обсуждении аналогичного воздействия на операции сравнения строк. Чтобы выполнить чувствительное к регистру упорядочивание нечувствительных к регистру строк (в данном случае значений столбца tstr), приведите тип столбца к двоичным строкам, используя ключевое слово BINARY:

```
mysql> SELECT tstr FROM textblob_val ORDER BY BINARY tstr;
+------+
| tstr |
+------+
| AAA  |
| BBB  |
| aaa  |
| bbb  |
+------+
```

Можно также преобразовать столбец вывода в двоичный и отсортировать его:

```
mysql> SELECT BINARY tstr FROM textblob_val ORDER BY 1;
+-------------+
| BINARY tstr |
+-------------+
| AAA         |
| BBB         |
| aaa         |
| bbb         |
+-------------+
```

Начиная с версии MySQL 4.0.2 можно применять специальную функцию приведения типов CAST():

```
mysql> SELECT tstr FROM textblob_val ORDER BY CAST(tstr AS BINARY);
+------+
| tstr |
+------+
| AAA  |
| BBB  |
| aaa  |
| bbb  |
+------+
```

Можно сделать и все наоборот: сортировать двоичные строки без учета регистра. Для этого преобразуйте все значения к верхнему или нижнему регистру с помощью функций UPPER() или LOWER():

```
mysql> SELECT bstr FROM textblob_val ORDER BY UPPER(bstr);
+------+
| bstr |
+------+
| aaa  |
| AAA  |
| bbb  |
| BBB  |
+------+
```

Или можно преобразовать столбец вывода и упорядочивать его, но это может привести к нежелательным изменениям в представлении выводимых значений:

```
mysql> SELECT UPPER(bstr) FROM textblob_val ORDER BY 1;
+-------------+
| UPPER(bstr) |
+-------------+
| AAA         |
| AAA         |
| BBB         |
| BBB         |
+-------------+
```

6.7. Сортировка по дате

Задача

Вы хотите произвести упорядочивание по временному значению.

Решение

Сортируйте по столбцу даты или времени, при необходимости игнорируя ненужные составляющие значений.

Обсуждение

Многие сведения включают в себя информацию о дате или времени, и нередко требуется упорядочить результаты в хронологическом порядке. MySQL умеет упорядочивать временные типы, так что не приходится прибегать к каким-то ухищрениям, чтобы отсортировать значения столбцов `DATE`, `DATETIME`, `TIME` или `TIMESTAMP`. Пусть некоторая таблица содержит столбцы всех этих типов:

```
mysql> SELECT * FROM temporal_val;
+------------+---------------------+----------+----------------+
| d          | dt                  | t        | ts             |
+------------+---------------------+----------+----------------+
| 1970-01-01 | 1884-01-01 12:00:00 | 13:00:00 | 19800101020000 |
| 1999-01-01 | 1860-01-01 12:00:00 | 19:00:00 | 20210101030000 |
| 1981-01-01 | 1871-01-01 12:00:00 | 03:00:00 | 19750101040000 |
| 1964-01-01 | 1899-01-01 12:00:00 | 01:00:00 | 19850101050000 |
+------------+---------------------+----------+----------------+
```

Используем инструкцию `ORDER BY` для упорядочивания каждого из столбцов в нужном направлении:

```
mysql> SELECT * FROM temporal_val ORDER BY d;
+------------+---------------------+----------+----------------+
| d          | dt                  | t        | ts             |
+------------+---------------------+----------+----------------+
| 1964-01-01 | 1899-01-01 12:00:00 | 01:00:00 | 19850101050000 |
| 1970-01-01 | 1884-01-01 12:00:00 | 13:00:00 | 19800101020000 |
| 1981-01-01 | 1871-01-01 12:00:00 | 03:00:00 | 19750101040000 |
| 1999-01-01 | 1860-01-01 12:00:00 | 19:00:00 | 20210101030000 |
+------------+---------------------+----------+----------------+
mysql> SELECT * FROM temporal_val ORDER BY dt;
+------------+---------------------+----------+----------------+
| d          | dt                  | t        | ts             |
+------------+---------------------+----------+----------------+
| 1999-01-01 | 1860-01-01 12:00:00 | 19:00:00 | 20210101030000 |
| 1981-01-01 | 1871-01-01 12:00:00 | 03:00:00 | 19750101040000 |
| 1970-01-01 | 1884-01-01 12:00:00 | 13:00:00 | 19800101020000 |
| 1964-01-01 | 1899-01-01 12:00:00 | 01:00:00 | 19850101050000 |
+------------+---------------------+----------+----------------+
mysql> SELECT * FROM temporal_val ORDER BY t;
+------------+---------------------+----------+----------------+
| d          | dt                  | t        | ts             |
+------------+---------------------+----------+----------------+
| 1964-01-01 | 1899-01-01 12:00:00 | 01:00:00 | 19850101050000 |
| 1981-01-01 | 1871-01-01 12:00:00 | 03:00:00 | 19750101040000 |
| 1970-01-01 | 1884-01-01 12:00:00 | 13:00:00 | 19800101020000 |
| 1999-01-01 | 1860-01-01 12:00:00 | 19:00:00 | 20210101030000 |
+------------+---------------------+----------+----------------+
mysql> SELECT * FROM temporal_val ORDER BY ts;
+------------+---------------------+----------+----------------+
| d          | dt                  | t        | ts             |
+------------+---------------------+----------+----------------+
| 1981-01-01 | 1871-01-01 12:00:00 | 03:00:00 | 19750101040000 |
| 1970-01-01 | 1884-01-01 12:00:00 | 13:00:00 | 19800101020000 |
| 1964-01-01 | 1899-01-01 12:00:00 | 01:00:00 | 19850101050000 |
| 1999-01-01 | 1860-01-01 12:00:00 | 19:00:00 | 20210101030000 |
+------------+---------------------+----------+----------------+
```

Иногда сортировка использует только часть даты или времени. В этом случае можно выделить необходимую составляющую (или составляющие) и работать только с ними (примеры будут приведены в следующих разделах).

6.8. Сортировка по календарному дню

Задача

Вы хотите выполнить сортировку по дню календарного года.

Решение

Используйте для сортировки месяц и день, игнорируя годовую составляющую даты.

Обсуждение

Сортировка в календарном порядке отличается от сортировки по дате. Вы не учитываете год и сортируете только по месяцу и дню, чтобы понять, как даты распределены по календарному году. Предположим, что у вас есть таблица `event`, которая выглядит следующим образом, если даты расположены в хронологическом порядке:

```
mysql> SELECT date, description FROM event ORDER BY date;
+------------+-------------------------------------+
| date       | description                         |
+------------+-------------------------------------+
| 1215-06-15 | Signing of the Magna Carta          |
| 1732-02-22 | George Washington's birthday        |
| 1776-07-14 | Bastille Day                        |
| 1789-07-04 | US Independence Day                 |
| 1809-02-12 | Abraham Lincoln's birthday          |
| 1919-06-28 | Signing of the Treaty of Versailles |
| 1944-06-06 | D-Day at Normandy Beaches           |
| 1957-10-04 | Sputnik launch date                 |
| 1958-01-31 | Explorer 1 launch date              |
| 1989-11-09 | Opening of the Berlin Wall          |
+------------+-------------------------------------+
```

Чтобы расположить их в календарном порядке, отсортируйте их по месяцу, затем по дню месяца:

```
mysql> SELECT date, description FROM event
    -> ORDER BY MONTH(date), DAYOFMONTH(date);
+------------+-------------------------------------+
| date       | description                         |
+------------+-------------------------------------+
| 1958-01-31 | Explorer 1 launch date              |
| 1809-02-12 | Abraham Lincoln's birthday          |
| 1732-02-22 | George Washington's birthday        |
| 1944-06-06 | D-Day at Normandy Beaches           |
| 1215-06-15 | Signing of the Magna Carta          |
| 1919-06-28 | Signing of the Treaty of Versailles |
| 1789-07-04 | US Independence Day                 |
| 1776-07-14 | Bastille Day                        |
| 1957-10-04 | Sputnik launch date                 |
| 1989-11-09 | Opening of the Berlin Wall          |
+------------+-------------------------------------+
```

MySQL включает в себя функцию `DAYOFYEAR()`, которая может быть полезна для упорядочивания по календарному дню:

```
mysql> SELECT date, description FROM event ORDER BY DAYOFYEAR(date);
+------------+-------------------------------------+
| date       | description                         |
+------------+-------------------------------------+
| 1958-01-31 | Explorer 1 launch date              |
| 1809-02-12 | Abraham Lincoln's birthday          |
| 1732-02-22 | George Washington's birthday        |
| 1944-06-06 | D-Day at Normandy Beaches           |
| 1215-06-15 | Signing of the Magna Carta          |
| 1919-06-28 | Signing of the Treaty of Versailles |
| 1789-07-04 | US Independence Day                 |
| 1776-07-14 | Bastille Day                        |
| 1957-10-04 | Sputnik launch date                 |
| 1989-11-09 | Opening of the Berlin Wall          |
+------------+-------------------------------------+
```

Кажется, что все хорошо, но это лишь потому, что в таблице нет записей, представляющих трудности для обработки посредством DAYOFYEAR(): функция может выдавать одно значение для разных календарных дней. Например, 29 февраля високосного года и 1 марта невисокосного года будут считаться одним и тем же днем:

```
mysql> SELECT DAYOFYEAR('1996-02-29'), DAYOFYEAR('1997-03-01');
+-------------------------+-------------------------+
| DAYOFYEAR('1996-02-29') | DAYOFYEAR('1997-03-01') |
+-------------------------+-------------------------+
|                      60 |                      60 |
+-------------------------+-------------------------+
```

Такая особенность функции DAYOFYEAR() означает, что результаты календарной сортировки, выполненной с ее помощью, не всегда будут корректными. Она может сгруппировать вместе даты, на самом деле являющиеся разными календарными днями.

Если даты в таблице представлены в отдельных столбцах для года, месяца и дня, то для календарной сортировки не потребуется извлекать составляющие. Просто извлекайте непосредственно интересующие вас столбцы. Например, в таблице мастеров бейсбола базы данных *baseball1.com* имена и даты рождения приводятся так:

```
mysql> SELECT lastname, firstname, birthyear, birthmonth, birthday
    -> FROM master;
+---------------+-------------+-----------+------------+----------+
| lastname      | firstname   | birthyear | birthmonth | birthday |
+---------------+-------------+-----------+------------+----------+
| AARON         | HANK        |      1934 |          2 |        5 |
| AARON         | TOMMIE      |      1939 |          8 |        5 |
| AASE          | DON         |      1954 |          9 |        8 |
| ABAD          | ANDY        |      1972 |          8 |       25 |
| ABADIE        | JOHN        |      1854 |         11 |        4 |
| ABBATICCHIO   | ED          |      1877 |          4 |       15 |
| ABBEY         | BERT        |      1869 |         11 |       29 |
| ABBEY         | CHARLIE     |      1866 |         10 |       14 |
...
```

Чтобы упорядочить записи в календарном порядке, используем столбцы birthmonth и birthday. Если вы не хотите, чтобы в рамках одного дня записи остались неупорядоченными, то можете добавить дополнительные столбцы сортировки. Следующий запрос выбирает игроков, чьи дни рожденья нам известны, упорядочивает их в календарном порядке и сортирует по имени для каждого календарного дня:

```
mysql> SELECT lastname, firstname, birthyear, birthmonth, birthday
    -> FROM master
    -> WHERE birthmonth IS NOT NULL AND birthday IS NOT NULL
    -> ORDER BY birthmonth, birthday, lastname, firstname;
+----------------+-------------+-----------+------------+----------+
| lastname       | firstname   | birthyear | birthmonth | birthday |
+----------------+-------------+-----------+------------+----------+
| ALLEN          | ETHAN       |      1904 |          1 |        1 |
| BEIRNE         | KEVIN       |      1974 |          1 |        1 |
| BELL           | RUDY        |      1881 |          1 |        1 |
| BERTHRONG      | HARRY       |      1844 |          1 |        1 |
| BETHEA         | BILL        |      1942 |          1 |        1 |
| BISHOP         | CHARLIE     |      1924 |          1 |        1 |
| BOBB           | RANDY       |      1948 |          1 |        1 |
| BRUCKMILLER    | ANDY        |      1882 |          1 |        1 |
...
```

Для больших наборов данных сортировка по отдельным столбцам составляющих даты может оказаться гораздо быстрее сортировки, извлекающей эти составляющие из значений DATE. Не требуется затрат на извлечение части значения, кроме того, что еще важнее, появляется возможность создания отдельных индексов для составляющих даты, что невозможно при работе со значениями типа DATE.

6.9. Сортировка по дню недели

Задача

Вы хотите выполнить сортировку по дню недели.

Решение

Используйте функцию DAYOFWEEK() для преобразования столбца даты в соответствующее числовое значение дня недели.

Обсуждение

Сортировка по дню недели аналогична сортировке по календарному дню, с той лишь разницей, что для вывода упорядоченных значений используется другая функция.

Вы можете получить день недели, применяя DAYNAME(), но тогда на выходе будут строки, упорядоченные в лексическом порядке, а не как дни недели

(воскресенье, понедельник, вторник и т. д.). Тут нам пригодится методика вывода одного значения, а упорядочивания по другому (рецепт 6.4). Будем выводить названия дней при помощи функции DAYNAME(), а сортировать их в порядке дней недели будем посредством функции DAYOFWEEK(), которая возвращает числовые значения от 1 до 7 для дней с воскресенья по субботу:

```
mysql> SELECT DAYNAME(date) AS day, date, description
    -> FROM event
    -> ORDER BY DAYOFWEEK(date);
+----------+------------+-------------------------------------+
| day      | date       | description                         |
+----------+------------+-------------------------------------+
| Sunday   | 1776-07-14 | Bastille Day                        |
| Sunday   | 1809-02-12 | Abraham Lincoln's birthday          |
| Monday   | 1215-06-15 | Signing of the Magna Carta          |
| Tuesday  | 1944-06-06 | D-Day at Normandy Beaches           |
| Thursday | 1989-11-09 | Opening of the Berlin Wall          |
| Friday   | 1957-10-04 | Sputnik launch date                 |
| Friday   | 1958-01-31 | Explorer 1 launch date              |
| Friday   | 1732-02-22 | George Washington's birthday        |
| Saturday | 1789-07-04 | US Independence Day                 |
| Saturday | 1919-06-28 | Signing of the Treaty of Versailles |
+----------+------------+-------------------------------------+
```

Если вы хотите выполнить сортировку по дню недели, но так, чтобы первым днем недели считался понедельник, используйте функцию MOD() для сопоставления понедельнику значения 0, вторнику – 1, ..., воскресенью – 6:

```
mysql> SELECT DAYNAME(date), date, description
    -> FROM event
    -> ORDER BY MOD(DAYOFWEEK(date) + 5, 7);
+---------------+------------+-------------------------------------+
| DAYNAME(date) | date       | description                         |
+---------------+------------+-------------------------------------+
| Monday        | 1215-06-15 | Signing of the Magna Carta          |
| Tuesday       | 1944-06-06 | D-Day at Normandy Beaches           |
| Thursday      | 1989-11-09 | Opening of the Berlin Wall          |
| Friday        | 1957-10-04 | Sputnik launch date                 |
| Friday        | 1958-01-31 | Explorer 1 launch date              |
| Friday        | 1732-02-22 | George Washington's birthday        |
| Saturday      | 1789-07-04 | US Independence Day                 |
| Saturday      | 1919-06-28 | Signing of the Treaty of Versailles |
| Sunday        | 1776-07-14 | Bastille Day                        |
| Sunday        | 1809-02-12 | Abraham Lincoln's birthday          |
+---------------+------------+-------------------------------------+
```

Любой день недели можно сделать первым в упорядоченном списке (табл. 6.1).

Для сортировки по дню недели можно использовать и функцию WEEKDAY(), помня, что она возвращает другой набор значений (0 для понедельника, ..., 6 – для воскресенья).

Таблица 6.1. Перенос первого дня недели

День, который должен выводиться первым	Выражение DAYOFWEEK()
Воскресенье	`DAYOFWEEK(date)`
Понедельник	`MOD(DAYOFWEEK(date) + 5, 7)`
Вторник	`MOD(DAYOFWEEK(date) + 4, 7)`
Среда	`MOD(DAYOFWEEK(date) + 3, 7)`
Четверг	`MOD(DAYOFWEEK(date) + 2, 7)`
Пятница	`MOD(DAYOFWEEK(date) + 1, 7)`
Суббота	`MOD(DAYOFWEEK(date) + 0, 7)`

6.10. Сортировка по времени дня

Задача

Вы хотите упорядочить записи по времени дня.

Решение

Выделите часы, минуты и секунды из столбца, содержащего время, и используйте их для сортировки.

Обсуждение

В зависимости от типа столбца сортировку по времени дня можно проводить разными способами. Если значения хранятся в столбце `TIME`, вы можете непосредственно упорядочивать их. Чтобы расположить по времени дня значения `DATETIME` или `TIMESTAMP`, извлеките из них составляющие времени и отсортируйте их. Например, таблица `mail` содержит значения `DATETIME`, которые могут быть упорядочены по времени дня следующим образом:

```
mysql> SELECT * FROM mail ORDER BY HOUR(t), MINUTE(t), SECOND(t);
+---------------------+---------+---------+---------+---------+---------+
| t                   | srcuser | srchost | dstuser | dsthost | size    |
+---------------------+---------+---------+---------+---------+---------+
| 2001-05-15 07:17:48 | gene    | mars    | gene    | saturn  |    3824 |
| 2001-05-15 08:50:57 | phil    | venus   | phil    | venus   |     978 |
| 2001-05-16 09:00:28 | gene    | venus   | barb    | mars    |     613 |
| 2001-05-14 09:31:37 | gene    | venus   | barb    | mars    |    2291 |
| 2001-05-11 10:15:08 | barb    | saturn  | tricia  | mars    |   58274 |
| 2001-05-15 10:25:52 | gene    | mars    | tricia  | saturn  |  998532 |
| 2001-05-14 11:52:17 | phil    | mars    | tricia  | saturn  |    5781 |
| 2001-05-12 12:48:13 | tricia  | mars    | gene    | venus   |  194925 |
...
```

Вы также можете применить функцию `TIME_TO_SEC()` для отбрасывания составляющей даты и возврата составляющей времени, пересчитанной в секунды:

```
mysql> SELECT * FROM mail ORDER BY TIME_TO_SEC(t);
+---------------------+---------+---------+---------+---------+---------+
| t                   | srcuser | srchost | dstuser | dsthost | size    |
+---------------------+---------+---------+---------+---------+---------+
| 2001-05-15 07:17:48 | gene    | mars    | gene    | saturn  |    3824 |
| 2001-05-15 08:50:57 | phil    | venus   | phil    | venus   |     978 |
| 2001-05-16 09:00:28 | gene    | venus   | barb    | mars    |     613 |
| 2001-05-14 09:31:37 | gene    | venus   | barb    | mars    |    2291 |
| 2001-05-11 10:15:08 | barb    | saturn  | tricia  | mars    |   58274 |
| 2001-05-15 10:25:52 | gene    | mars    | tricia  | saturn  |  998532 |
| 2001-05-14 11:52:17 | phil    | mars    | tricia  | saturn  |    5781 |
| 2001-05-12 12:48:13 | tricia  | mars    | gene    | venus   |  194925 |
...
```

6.11. Сортировка по подстрокам значений столбцов

Задача

Вы хотите упорядочить множество значений, используя одну или несколько подстрок каждого значения.

Решение

Извлеките интересующие вас фрагменты и упорядочивайте их по отдельности.

Обсуждение

Для решения задачи используем сортировку по значению выражения (рецепт 6.3). Если вы хотите упорядочить записи, используя только определенную часть значения столбца, извлеките интересующую вас подстроку и используйте ее в инструкции `ORDER BY`. Проще всего, если подстроки начинаются с фиксированной позиции и имеют фиксированную длину. Подстроки переменной длины или имеющие переменное местоположение можно использовать для сортировки, если имеется надежный способ их идентификации. Несколько последующих рецептов показывают, как применять извлечение подстрок для сортировки в специальном порядке.

6.12. Сортировка по подстрокам фиксированной длины

Задача

Вы хотите выполнить сортировку, используя части столбца, имеющие фиксированное местоположение.

Решение

Выделите интересующие вас части с помощью функций LEFT(), MID() и RIGHT() и упорядочьте их.

Обсуждение

Предположим, что у вас есть таблица housewares, представляющая собой каталог предметов домашней обстановки, элементы которой снабжены 11-символьными идентификаторами, состоящими их трех частей: трехзначное сокращение для категории изделия (например, DIN для «dining room» – столовая, KIT для «kitchen» – кухня), пятизначный серийный номер и двузначный код страны, где было изготовлено изделие:

```
mysql> SELECT * FROM housewares;
+------------+------------------+
| id         | description      |
+------------+------------------+
| DIN40672US | dining table     |
| KIT00372UK | garbage disposal |
| KIT01729JP | microwave oven   |
| BED00038SG | bedside lamp     |
| BTH00485US | shower stall     |
| BTH00415JP | lavatory         |
+------------+------------------+
```

На самом деле такой способ хранения составных значений идентификаторов не очень удачен, и в дальнейшем мы поговорим о том, как записывать их в отдельные столбцы (рецепт 11.13). Но пока что будем считать, что значения должны храниться именно так.

Если вы хотите упорядочить записи таблицы по значениям id, то просто используйте все значение столбца:

```
mysql> SELECT * FROM housewares ORDER BY id;
+------------+------------------+
| id         | description      |
+------------+------------------+
| BED00038SG | bedside lamp     |
| BTH00415JP | lavatory         |
| BTH00485US | shower stall     |
| DIN40672US | dining table     |
| KIT00372UK | garbage disposal |
| KIT01729JP | microwave oven   |
+------------+------------------+
```

Но что делать, если потребуется упорядочить записи по одной из трех составляющих, например по стране-изготовителю? Для таких операций удобно применять функции, выделяющие части столбца, такие как LEFT(), MID() и RIGHT(). Эти функции можно использовать для разбиения значений id на три части:

```
mysql> SELECT id,
    -> LEFT(id,3) AS category,
    -> MID(id,4,5) AS serial,
    -> RIGHT(id,2) AS country
    -> FROM housewares;
+------------+----------+--------+---------+
| id         | category | serial | country |
+------------+----------+--------+---------+
| DIN40672US | DIN      | 40672  | US      |
| KIT00372UK | KIT      | 00372  | UK      |
| KIT01729JP | KIT      | 01729  | JP      |
| BED00038SG | BED      | 00038  | SG      |
| BTH00485US | BTH      | 00485  | US      |
| BTH00415JP | BTH      | 00415  | JP      |
+------------+----------+--------+---------+
```

Любую из этих трех подстрок фиксированной длины можно использовать для сортировки как отдельно, так и в сочетании с другими. Чтобы упорядочить изделия по категории, извлеките составляющую категории и поместите ее в инструкцию `ORDER BY`:

```
mysql> SELECT * FROM housewares ORDER BY LEFT(id,3);
+------------+------------------+
| id         | description      |
+------------+------------------+
| BED00038SG | bedside lamp     |
| BTH00485US | shower stall     |
| BTH00415JP | lavatory         |
| DIN40672US | dining table     |
| KIT00372UK | garbage disposal |
| KIT01729JP | microwave oven   |
+------------+------------------+
```

Чтобы выполнить сортировку по серийному номеру изделия, используйте `MID()` для извлечения из значений `id` пяти центральных символов начиная с четвертого:

```
mysql> SELECT * FROM housewares ORDER BY MID(id,4,5);
+------------+------------------+
| id         | description      |
+------------+------------------+
| BED00038SG | bedside lamp     |
| KIT00372UK | garbage disposal |
| BTH00415JP | lavatory         |
| BTH00485US | shower stall     |
| KIT01729JP | microwave oven   |
| DIN40672US | dining table     |
+------------+------------------+
```

Похоже на числовую сортировку, но на самом деле она строковая, так как `MID()` возвращает строки. В данном случае благодаря тому, что у «чисел» имеются начальные нули (обеспечивающие им одинаковую длину), лексический и числовой порядки сортировки совпадают.

Для упорядочивания по коду страны используйте два самых правых символа значений `id`:

```
mysql> SELECT * FROM housewares ORDER BY RIGHT(id,2);
+------------+------------------+
| id         | description      |
+------------+------------------+
| KIT01729JP | microwave oven   |
| BTH00415JP | lavatory         |
| BED00038SG | bedside lamp     |
| KIT00372UK | garbage disposal |
| DIN40672US | dining table     |
| BTH00485US | shower stall     |
+------------+------------------+
```

Можно выполнить сортировку и по комбинации подстрок. Например, чтобы отсортировать по коду страны и серийному номеру, создайте такой запрос:

```
mysql> SELECT * FROM housewares ORDER BY RIGHT(id,2), MID(id,4,5);
+------------+------------------+
| id         | description      |
+------------+------------------+
| BTH00415JP | lavatory         |
| KIT01729JP | microwave oven   |
| BED00038SG | bedside lamp     |
| KIT00372UK | garbage disposal |
| BTH00485US | shower stall     |
| DIN40672US | dining table     |
+------------+------------------+
```

6.13. Сортировка по подстрокам переменной длины

Задача

Вы хотите выполнить сортировку, используя части столбца, *не* имеющие фиксированного местоположения внутри столбца.

Решение

Придумайте какой-нибудь способ идентификации тех частей, которые вы хотите извлечь, иначе у вас ничего не получится.

Обсуждение

Если подстрока, которую вы хотели бы использовать для сортировки, имеет переменную длину, то необходим надежный способ извлечения именно этой части строки. Чтобы посмотреть, что имеется в виду, создадим таблицу `housewares2`, похожую на таблицу `housewares` из предыдущего раздела, но без начальных нулей в серийных номерах, входящих в состав значений `id`:

```
mysql> SELECT * FROM housewares2;
+------------+------------------+
| id         | description      |
+------------+------------------+
| DIN40672US | dining table     |
| KIT372UK   | garbage disposal |
| KIT1729JP  | microwave oven   |
| BED38SG    | bedside lamp     |
| BTH485US   | shower stall     |
| BTH415JP   | lavatory         |
+------------+------------------+
```

Составляющие категории и страны значений `id` можно извлечь и упорядочить с помощью функций `LEFT()` и `RIGHT()`, как и в таблице `housewares`. Но числовая часть значения теперь имеет переменную длину и уже не может быть извлечена и упорядочена посредством вызова `MID()`. Применим функцию `SUBSTRING()` для того, чтобы пропустить три первых символа и вернуть оставшуюся часть значения, начиная с четвертого символа (первой цифры):

```
mysql> SELECT id, SUBSTRING(id,4) FROM housewares2;
+------------+-----------------+
| id         | SUBSTRING(id,4) |
+------------+-----------------+
| DIN40672US | 40672US         |
| KIT372UK   | 372UK           |
| KIT1729JP  | 1729JP          |
| BED38SG    | 38SG            |
| BTH485US   | 485US           |
| BTH415JP   | 415JP           |
+------------+-----------------+
```

Затем возьмем все, кроме двух самых правых символов. Сделаем это, например, так:

```
mysql> SELECT id, LEFT(SUBSTRING(id,4),LENGTH(SUBSTRING(id,4)-2))
    -> FROM housewares2;
+------------+-------------------------------------------------+
| id         | LEFT(SUBSTRING(id,4),LENGTH(SUBSTRING(id,4)-2)) |
+------------+-------------------------------------------------+
| DIN40672US | 40672                                           |
| KIT372UK   | 372                                             |
| KIT1729JP  | 1729                                            |
| BED38SG    | 38                                              |
| BTH485US   | 485                                             |
| BTH415JP   | 415                                             |
+------------+-------------------------------------------------+
```

Но можно поступить и проще. Функция `SUBSTRING()` принимает третий необязательный аргумент, указывающий длину строки, а мы знаем, что длина центральной части значения равна длине строки минус пять (три символа в начале и два символа в конце). Следующий запрос показывает, как получить числовую центральную часть значения, отбрасывая самую правую часть идентификатора изделия:

```
mysql> SELECT id, SUBSTRING(id,4), SUBSTRING(id,4,LENGTH(id)-5)
    -> FROM housewares2;
+------------+-----------------+-----------------------------+
| id         | SUBSTRING(id,4) | SUBSTRING(id,4,LENGTH(id)-5) |
+------------+-----------------+-----------------------------+
| DIN40672US | 40672US         | 40672                       |
| KIT372UK   | 372UK           | 372                         |
| KIT1729JP  | 1729JP          | 1729                        |
| BED38SG    | 38SG            | 38                          |
| BTH485US   | 485US           | 485                         |
| BTH415JP   | 415JP           | 415                         |
+------------+-----------------+-----------------------------+
```

К сожалению, несмотря на то что итоговое выражения правильно извлекает числовую часть идентификатора, результирующие значения являются строками. Поэтому они сортируются в лексическом, а не в числовом порядке:

```
mysql> SELECT * FROM housewares2
    -> ORDER BY SUBSTRING(id,4,LENGTH(id)-5);
+------------+------------------+
| id         | description      |
+------------+------------------+
| KIT1729JP  | microwave oven   |
| KIT372UK   | garbage disposal |
| BED38SG    | bedside lamp     |
| DIN40672US | dining table     |
| BTH415JP   | lavatory         |
| BTH485US   | shower stall     |
+------------+------------------+
```

Что же делать? Можно добавить к ним ноль, чтобы сообщить MySQL о необходимости преобразования строки в число, в результате чего серийные номера будут упорядочены как числа:

```
mysql> SELECT * FROM housewares2
    -> ORDER BY SUBSTRING(id,4,LENGTH(id)-5)+0;
+------------+------------------+
| id         | description      |
+------------+------------------+
| BED38SG    | bedside lamp     |
| KIT372UK   | garbage disposal |
| BTH415JP   | lavatory         |
| BTH485US   | shower stall     |
| KIT1729JP  | microwave oven   |
| DIN40672US | dining table     |
+------------+------------------+
```

Но в нашем конкретном случае есть более простое решение. Нет необходимости в вычислении длины числовой части строки, так как операция преобразования строки в число отбрасывает замыкающие нечисловые символы и предоставляет значения, которые и требуются для сортировки по серийному номеру – составляющей значений `id`, имеющей переменную длину. То есть третий аргумент функции `SUBSTRING()` не нужен:

```
mysql> SELECT * FROM housewares2
    -> ORDER BY SUBSTRING(id,4)+0;
+------------+------------------+
| id         | description      |
+------------+------------------+
| BED38SG    | bedside lamp     |
| KIT372UK   | garbage disposal |
| BTH415JP   | lavatory         |
| BTH485US   | shower stall     |
| KIT1729JP  | microwave oven   |
| DIN40672US | dining table     |
+------------+------------------+
```

В предыдущем примере для извлечения подстроки переменной длины использовалось то, что в середине значений `id` стояли символы, отличные от краевых символов (цифры и не-цифры). Возможны случаи, когда значение столбца разбивается на части символами-разделителями. Пусть таблица `housewares3` содержит такие значения `id`:

```
mysql> SELECT * FROM housewares3;
+---------------+------------------+
| id            | description      |
+---------------+------------------+
| 13-478-92-2   | dining table     |
| 873-48-649-63 | garbage disposal |
| 8-4-2-1       | microwave oven   |
| 97-681-37-66  | bedside lamp     |
| 27-48-534-2   | shower stall     |
| 5764-56-89-72 | lavatory         |
+---------------+------------------+
```

Для того чтобы извлекать части таких значений, используйте `SUBSTRING_INDEX(`*str*`,`*c*`,`*n*`)`. Выполняется просмотр строки *str* в поиске *n*-го вхождения указанного символа *c* и возвращается все, что расположено слева от этого символа. Например, следующий вызов возвращает `13-478`:

```
SUBSTRING_INDEX('13-478-92-2','-',2)
```

Если значение *n* отрицательное, то просмотр *c* начинается с правого конца и возвращает все, что расположено слева от найденного символа. Такой вызов вернет `478-92-2`:

```
SUBSTRING_INDEX('13-478-92-2','-',-3)
```

Используя вызовы `SUBSTRING_INDEX()` с отрицательными и положительными аргументами, можно последовательно извлекать части из значений `id`. Можно сначала извлечь первые *n* сегментов значения, затем отбросить самый правый. Изменяя *n* от 1 до 4, получим последовательные отрезки значения `id`, расположенные слева направо:

```
SUBSTRING_INDEX(SUBSTRING_INDEX(id,'-',1),'-',-1)
SUBSTRING_INDEX(SUBSTRING_INDEX(id,'-',2),'-',-1)
SUBSTRING_INDEX(SUBSTRING_INDEX(id,'-',3),'-',-1)
SUBSTRING_INDEX(SUBSTRING_INDEX(id,'-',4),'-',-1)
```

Первое из этих выражений можно оптимизировать, так как внутренний вызов SUBSTRING_INDEX() возвращает односегментную строку, и этого достаточно для получения самой левой составляющей id:

```
SUBSTRING_INDEX(id,'-',1)
```

Есть и другой способ получения подстрок – извлекаем *n* сегментов справа и отбрасываем первый. Будем менять *n* от –4 до –1:

```
SUBSTRING_INDEX(SUBSTRING_INDEX(id,'-',-4),'-',1)
SUBSTRING_INDEX(SUBSTRING_INDEX(id,'-',-3),'-',1)
SUBSTRING_INDEX(SUBSTRING_INDEX(id,'-',-2),'-',1)
SUBSTRING_INDEX(SUBSTRING_INDEX(id,'-',-1),'-',1)
```

Опять-таки, возможна оптимизация. В четвертом выражении внутреннего вызова SUBSTRING_INDEX() достаточно для возврата итоговой подстроки:

```
SUBSTRING_INDEX(id,'-',-1)
```

Возможно, эти выражения нелегко прочитать и понять, так что, вероятно, вам стоит немного потренироваться, чтобы посмотреть, как они работают. Рассмотрим пример, показывающий, как получить второй и четвертый сегменты значений id:

```
mysql> SELECT
    -> id,
    -> SUBSTRING_INDEX(SUBSTRING_INDEX(id,'-',2),'-',-1) AS segment2,
    -> SUBSTRING_INDEX(SUBSTRING_INDEX(id,'-',4),'-',-1) AS segment4
    -> FROM housewares3;
+---------------+----------+----------+
| id            | segment2 | segment4 |
+---------------+----------+----------+
| 13-478-92-2   | 478      | 2        |
| 873-48-649-63 | 48       | 63       |
| 8-4-2-1       | 4        | 1        |
| 97-681-37-66  | 681      | 66       |
| 27-48-534-2   | 48       | 2        |
| 5764-56-89-72 | 56       | 72       |
+---------------+----------+----------+
```

Чтобы использовать подстроки для сортировки, поместите соответствующие выражения в инструкцию ORDER BY. (Не забудьте выполнить преобразование из строки в число, если вас интересует не лексическая, а числовая сортировка.) Приведем два запроса, упорядочивающие результаты по второму сегменту значения id. Первый запрос выполняет лексическую сортировку, а второй – числовую:

```
mysql> SELECT * FROM housewares3
    -> ORDER BY SUBSTRING_INDEX(SUBSTRING_INDEX(id,'-',2),'-',-1);
+---------------+------------------+
| id            | description      |
+---------------+------------------+
| 8-4-2-1       | microwave oven   |
| 13-478-92-2   | dining table     |
| 873-48-649-63 | garbage disposal |
```

```
| 27-48-534-2   | shower stall     |
| 5764-56-89-72 | lavatory         |
| 97-681-37-66  | bedside lamp     |
+---------------+------------------+
mysql> SELECT * FROM housewares3
    -> ORDER BY SUBSTRING_INDEX(SUBSTRING_INDEX(id,'-',2),'-',-1)+0;
+---------------+------------------+
| id            | description      |
+---------------+------------------+
| 8-4-2-1       | microwave oven   |
| 873-48-649-63 | garbage disposal |
| 27-48-534-2   | shower stall     |
| 5764-56-89-72 | lavatory         |
| 13-478-92-2   | dining table     |
| 97-681-37-66  | bedside lamp     |
+---------------+------------------+
```

Выражения для извлечения строк выглядят достаточно запутанно, но по крайней мере в значениях столбцов, к которым мы их применяем, количество сегментов постоянно. Если речь идет о сортировке значений, число сегментов которых переменно, задача усложняется. Поговорим об этом в следующем разделе.

6.14. Сортировка имен хостов по доменам

Задача

Вы хотите упорядочить имена хостов по доменам, при этом правая часть имени является более важной, чем левая.

Решение

Разбейте имена на части и сортируйте части справа налево.

Обсуждение

Имена хостов – это символьные строки, поэтому их естественным порядком является лексический. Однако часто требуется упорядочить имена хостов по доменам, при этом самые правые сегменты значений являются более значимыми, чем самые левые. Предположим, что у вас есть таблица `hostname` с именами хостов:

```
mysql> SELECT name FROM hostname ORDER BY name;
+--------------------+
| name               |
+--------------------+
| cvs.php.net        |
| dbi.perl.org       |
| jakarta.apache.org |
| lists.mysql.com    |
| mysql.com          |
| www.kitebird.com   |
+--------------------+
```

Результат запроса отсортирован в естественном лексическом порядке значений name, который отличается от сортировки по домену (табл. 6.2):

Таблица 6.2. Сортировка строк

В лексическом порядке	По домену
cvs.php.net	www.kitebird.com
dbi.perl.org	mysql.com
jakarta.apache.org	lists.mysql.com
lists.mysql.com	cvs.php.net
mysql.com	jakarta.apache.org
www.kitebird.com	dbi.perl.org

Формирование вывода, отсортированного по домену, – это задача сортировки по подстроке. Необходимо извлечь все сегменты имен, чтобы отсортировать их справа налево. Появляется и дополнительная сложность: значения могут содержать разное количество сегментов, как имена хостов из примера (большинство имен состоит из трех частей, но mysql.com – только из двух).

Чтобы извлечь части имен хостов, для начала используем функцию SUBSTRING_INDEX() так же, как и в рецепте 6.13. Значения имен хостов состоят максимум из трех сегментов, так что слева направо можно извлечь их следующим образом:

```
SUBSTRING_INDEX(SUBSTRING_INDEX(name,'.',-3),'.',1)
SUBSTRING_INDEX(SUBSTRING_INDEX(name,'.',-2),'.',1)
SUBSTRING_INDEX(name,'.',-1)
```

Эти выражения будут работать правильно, если все имена хостов будут состоять ровно из трех частей. Но если имя содержит меньше трех компонентов, корректный результат не получается:

```
mysql> SELECT name,
    -> SUBSTRING_INDEX(SUBSTRING_INDEX(name,'.',-3),'.',1) AS leftmost,
    -> SUBSTRING_INDEX(SUBSTRING_INDEX(name,'.',-2),'.',1) AS middle,
    -> SUBSTRING_INDEX(name,'.',-1) AS rightmost
    -> FROM hostname;
+--------------------+----------+----------+-----------+
| name               | leftmost | middle   | rightmost |
+--------------------+----------+----------+-----------+
| cvs.php.net        | cvs      | php      | net       |
| dbi.perl.org       | dbi      | perl     | org       |
| lists.mysql.com    | lists    | mysql    | com       |
| mysql.com          | mysql    | mysql    | com       |
| jakarta.apache.org | jakarta  | apache   | org       |
| www.kitebird.com   | www      | kitebird | com       |
+--------------------+----------+----------+-----------+
```

Обратите внимание на вывод для строки mysql.com; в столбце leftmost присутствует значение mysql, хотя там должна бы быть пустая строка. Выражения

извлечения сегментов действуют так: они отбрасывают *n* сегментов справа и возвращают самый левый сегмент результата. Источник проблемы с mysql.com в том, что если сегментов меньше, чем *n*, то выражение просто возвращает самый левый сегмент. Чтобы исправить ошибку, добавим в начало имени хоста достаточное количество точек, чтобы гарантировать наличие необходимого количества сегментов:

```
mysql> SELECT name,
    -> SUBSTRING_INDEX(SUBSTRING_INDEX(CONCAT('..',name),'.',-3),'.',1)
    -> AS leftmost,
    -> SUBSTRING_INDEX(SUBSTRING_INDEX(CONCAT('.',name),'.',-2),'.',1)
    -> AS middle,
    -> SUBSTRING_INDEX(name,'.',-1) AS rightmost
    -> FROM hostname;
+--------------------+----------+----------+-----------+
| name               | leftmost | middle   | rightmost |
+--------------------+----------+----------+-----------+
| cvs.php.net        | cvs      | php      | net       |
| dbi.perl.org       | dbi      | perl     | org       |
| lists.mysql.com    | lists    | mysql    | com       |
| mysql.com          |          | mysql    | com       |
| jakarta.apache.org | jakarta  | apache   | org       |
| www.kitebird.com   | www      | kitebird | com       |
+--------------------+----------+----------+-----------+
```

Выражения выглядят отвратительно, но обеспечивают извлечение подстрок, необходимых для корректной сортировки значений имен хостов справа налево:

```
mysql> SELECT name FROM hostname
    -> ORDER BY
    -> SUBSTRING_INDEX(name,'.',-1),
    -> SUBSTRING_INDEX(SUBSTRING_INDEX(CONCAT('.',name),'.',-2),'.',1),
    -> SUBSTRING_INDEX(SUBSTRING_INDEX(CONCAT('..',name),'.',-3),'.',1);
+--------------------+
| name               |
+--------------------+
| www.kitebird.com   |
| mysql.com          |
| lists.mysql.com    |
| cvs.php.net        |
| jakarta.apache.org |
| dbi.perl.org       |
+--------------------+
```

Если максимальное количество сегментов имени хоста равно не трем, а четырем, то необходимо добавить в инструкцию ORDER BY еще одно выражение SUBSTRING_INDEX(), которое помещает в начало значения три точки.

6.15. Сортировка IP-адресов в числовом порядке

Задача

Вы хотите упорядочить строки, представляющие IP-адреса (четверки чисел, разделенные точками) как числа.

Решение

Разбейте строки на составляющие и сортируйте их как числа. Или просто используйте функцию `INET_ATON()`.

Обсуждение

Если таблица содержит IP-адреса, представленные строками в виде четверок чисел, разделенных точками (например, `111.122.133.144`), они будут упорядочиваться лексически. Чтобы сформировать вывод, отсортированный в числовом порядке, можно упорядочивать адреса как значения, состоящие из четырех частей, каждая из которых сортируется как число. Применим прием, наподобие используемого для сортировки имен хостов, но имеющий следующий отличия:

- Четверки чисел, разделенные точками, всегда состоят из четырех сегментов, так что не нужно добавлять точки в начало значения при извлечении подстрок.
- Значения сортируются слева направо, поэтому порядок, в котором подстроки указываются в инструкции `ORDER BY`, противоположен использованному для сортировки имен хостов.
- Сегменты значений являются числами, поэтому необходимо добавлять к каждой подстроке ноль, чтобы указать MySQL на необходимость выполнения числовой, а не лексической сортировки.

Предположим, что у вас есть таблица `hostip`, содержащая строковый столбец `ip` с IP-адресами:

```
mysql> SELECT ip FROM hostip ORDER BY ip;
+-----------------+
| ip              |
+-----------------+
| 127.0.0.1       |
| 192.168.0.10    |
| 192.168.0.2     |
| 192.168.1.10    |
| 192.168.1.2     |
| 21.0.0.1        |
| 255.255.255.255 |
+-----------------+
```

Результат этого запроса отсортирован в лексическом порядке. Чтобы изменить порядок на числовой, можно извлечь каждый сегмент, добавить ноль для преобразования его в число и поместить в инструкцию `ORDER BY`:

```
mysql> SELECT ip FROM hostip
    -> ORDER BY
    -> SUBSTRING_INDEX(ip,'.',1)+0,
    -> SUBSTRING_INDEX(SUBSTRING_INDEX(ip,'.',-3),'.',1)+0,
    -> SUBSTRING_INDEX(SUBSTRING_INDEX(ip,'.',-2),'.',1)+0,
    -> SUBSTRING_INDEX(ip,'.',-1)+0;
+-----------------+
| ip              |
+-----------------+
| 21.0.0.1        |
| 127.0.0.1       |
| 192.168.0.2     |
| 192.168.0.10    |
| 192.168.1.2     |
| 192.168.1.10    |
| 255.255.255.255 |
+-----------------+
```

Если вы работаете с MySQL версии 3.23.15 или выше, то можете решить задачу проще. Отсортируйте IP-адреса с помощью функции `INET_ATON()`, которая преобразует сетевой адрес непосредственно в соответствующее число:

```
mysql> SELECT ip FROM hostip ORDER BY INET_ATON(ip);
+-----------------+
| ip              |
+-----------------+
| 21.0.0.1        |
| 127.0.0.1       |
| 192.168.0.2     |
| 192.168.0.10    |
| 192.168.1.2     |
| 192.168.1.10    |
| 255.255.255.255 |
+-----------------+
```

Если вы надеетесь выполнить сортировку, просто добавив ноль к значению `ip` и применив `ORDER BY` к результату, посмотрите, какие значения на самом деле будут получены при преобразовании строк в числа:

```
mysql> SELECT ip, ip+0 FROM hostip;
+-----------------+---------+
| ip              | ip+0    |
+-----------------+---------+
| 127.0.0.1       |     127 |
| 192.168.0.2     | 192.168 |
| 192.168.0.10    | 192.168 |
| 192.168.1.2     | 192.168 |
| 192.168.1.10    | 192.168 |
| 255.255.255.255 | 255.255 |
| 21.0.0.1        |      21 |
+-----------------+---------+
```

Преобразование оставляет только ту часть каждого значения, которую можно интерпретировать как число. Оставшаяся часть будет недоступна для сортировки, а для вывода корректного результата нужны все составляющие.

6.16. Размещение некоторых значений в начале или конце упорядоченного списка

Задача

Вы хотите, чтобы столбец был отсортирован обычным образом, за исключением нескольких значений, которые вы хотите видеть на определенном месте.

Решение

Добавьте в инструкцию `ORDER BY` еще один столбец сортировки, который поместит эти несколько значений туда, куда нужно. Остальные столбцы сортировки будут оказывать свое обычное воздействие на другие значения.

Обсуждение

Если вы хотите, чтобы результирующее множество было упорядочено обычным образом, *за исключением того*, что в его начало должны быть помещены определенные значения, создайте дополнительный столбец, который содержит 0 для указанных значений и 1 для остальных. Будем использовать прием, который уже применялся для помещения значений `NULL` в начало упорядоченного списка (рецепт 6.5). Предположим, что вы хотите отсортировать таблицу сообщений `mail` по отправителям/получателям, при этом первыми должны выводиться сообщения пользователя `phil`. Сделаем так:

```
mysql> SELECT t, srcuser, dstuser, size
    -> FROM mail
    -> ORDER BY IF(srcuser='phil',0,1), srcuser, dstuser;
+---------------------+---------+---------+---------+
| t                   | srcuser | dstuser | size    |
+---------------------+---------+---------+---------+
| 2001-05-16 23:04:19 | phil    | barb    |   10294 |
| 2001-05-12 15:02:49 | phil    | phil    |    1048 |
| 2001-05-15 08:50:57 | phil    | phil    |     978 |
| 2001-05-14 11:52:17 | phil    | tricia  |    5781 |
| 2001-05-17 12:49:23 | phil    | tricia  |     873 |
| 2001-05-14 14:42:21 | barb    | barb    |   98151 |
| 2001-05-11 10:15:08 | barb    | tricia  |   58274 |
| 2001-05-13 13:59:18 | barb    | tricia  |     271 |
| 2001-05-14 09:31:37 | gene    | barb    |    2291 |
| 2001-05-16 09:00:28 | gene    | barb    |     613 |
| 2001-05-15 07:17:48 | gene    | gene    |    3824 |
| 2001-05-15 17:35:31 | gene    | gene    |    3856 |
| 2001-05-19 22:21:51 | gene    | gene    |   23992 |
| 2001-05-15 10:25:52 | gene    | tricia  |  998532 |
```

```
| 2001-05-12 12:48:13 | tricia  | gene    |  194925 |
| 2001-05-14 17:03:01 | tricia  | phil    | 2394482 |
+---------------------+---------+---------+---------+
```

Значение дополнительного столбца сортировки равно 0 для строк, в которых значение srcuser равно phil, и 1 – для остальных строк. Теперь записи для сообщений, отправленных пользователем phil, «всплывут» на самый верх вывода. (Чтобы вместо этого «утопить» их, то есть спустить в самый низ вывода, задайте обратное направление сортировки при помощи DESC или поменяйте местами второй и третий аргументы функции IF().)

Перемещать можно не только конкретные значения, но и записи, для которых выполнены какие-то условия. Чтобы вывести первыми записи для случаев, когда люди писали письма сами себе, выполним такой запрос:

```
mysql> SELECT t, srcuser, dstuser, size
    -> FROM mail
    -> ORDER BY IF(srcuser=dstuser,0,1), srcuser, dstuser;
+---------------------+---------+---------+---------+
| t                   | srcuser | dstuser | size    |
+---------------------+---------+---------+---------+
| 2001-05-14 14:42:21 | barb    | barb    |   98151 |
| 2001-05-15 07:17:48 | gene    | gene    |    3824 |
| 2001-05-15 17:35:31 | gene    | gene    |    3856 |
| 2001-05-19 22:21:51 | gene    | gene    |   23992 |
| 2001-05-12 15:02:49 | phil    | phil    |    1048 |
| 2001-05-15 08:50:57 | phil    | phil    |     978 |
| 2001-05-11 10:15:08 | barb    | tricia  |   58274 |
| 2001-05-13 13:59:18 | barb    | tricia  |     271 |
| 2001-05-14 09:31:37 | gene    | barb    |    2291 |
| 2001-05-16 09:00:28 | gene    | barb    |     613 |
| 2001-05-15 10:25:52 | gene    | tricia  |  998532 |
| 2001-05-16 23:04:19 | phil    | barb    |   10294 |
| 2001-05-14 11:52:17 | phil    | tricia  |    5781 |
| 2001-05-17 12:49:23 | phil    | tricia  |     873 |
| 2001-05-12 12:48:13 | tricia  | gene    |  194925 |
| 2001-05-14 17:03:01 | tricia  | phil    | 2394482 |
+---------------------+---------+---------+---------+
```

Если вы хорошо представляете себе содержимое вашей таблицы, то, возможно, сумеете обойтись без дополнительного столбца сортировки. Например, в таблице mail столбец srcuser никогда не содержит NULL, поэтому предыдущий запрос можно переписать, используя в инструкции ORDER BY на один столбец меньше (считаем, что значения NULL при сортировке располагаются перед всеми значениями не-NULL):

```
mysql> SELECT t, srcuser, dstuser, size
    -> FROM mail
    -> ORDER BY IF(srcuser=dstuser,NULL,srcuser), dstuser;
+---------------------+---------+---------+---------+
| t                   | srcuser | dstuser | size    |
+---------------------+---------+---------+---------+
| 2001-05-14 14:42:21 | barb    | barb    |   98151 |
```

```
| 2001-05-15 07:17:48 | gene    | gene    |    3824 |
| 2001-05-15 17:35:31 | gene    | gene    |    3856 |
| 2001-05-19 22:21:51 | gene    | gene    |   23992 |
| 2001-05-12 15:02:49 | phil    | phil    |    1048 |
| 2001-05-15 08:50:57 | phil    | phil    |     978 |
| 2001-05-11 10:15:08 | barb    | tricia  |   58274 |
| 2001-05-13 13:59:18 | barb    | tricia  |     271 |
| 2001-05-14 09:31:37 | gene    | barb    |    2291 |
| 2001-05-16 09:00:28 | gene    | barb    |     613 |
| 2001-05-15 10:25:52 | gene    | tricia  |  998532 |
| 2001-05-16 23:04:19 | phil    | barb    |   10294 |
| 2001-05-14 11:52:17 | phil    | tricia  |    5781 |
| 2001-05-17 12:49:23 | phil    | tricia  |     873 |
| 2001-05-12 12:48:13 | tricia  | gene    |  194925 |
| 2001-05-14 17:03:01 | tricia  | phil    | 2394482 |
+---------------------+---------+---------+---------+
```

6.17. Сортировка в порядке, определенном пользователем

Задача

Вы хотите определить порядок сортировки для всех значений столбца.

Решение

Используйте функцию `FIELD()` для отображения значений столбцов на последовательность, располагающую значения в нужном порядке.

Обсуждение

В предыдущем разделе было показано, как переместить указанную группу строк в начало упорядоченного списка. Если вы хотите определить специальный порядок для *всех* значений столбца, примените функцию `FIELD()`, которая сопоставит их списку числовых значений, и используйте сортировку для чисел. `FIELD()` сравнивает свой первый аргумент со следующими и возвращает число, показывающее, с которым из них он совпадает. Следующий вызов `FIELD()` сравнивает аргумент *value* с аргументами *str1*, *str2*, *str3* и *str4* и возвращает 1, 2, 3 или 4 в зависимости от того, какому из аргументов равно значение *value*:

```
FIELD(value,str1,str2,str3,str4)
```

Значений для сравнения не обязательно должно быть четыре; `FIELD()` принимает список аргументов переменной длины. Если *value* – это `NULL` или совпадения нет, `FIELD()` возвращает 0.

Функцию `FIELD()` можно использовать для сортировки произвольного множества значений в любом порядке. Например, чтобы вывести записи табли-

цы driver_log для имен Henry, Suzi и Ben именно в таком порядке, выполните запрос:

```
mysql> SELECT * FROM driver_log
    -> ORDER BY FIELD(name,'Henry','Suzi','Ben');
+--------+-------+------------+-------+
| rec_id | name  | trav_date  | miles |
+--------+-------+------------+-------+
|      3 | Henry | 2001-11-29 |   300 |
|      4 | Henry | 2001-11-27 |    96 |
|      6 | Henry | 2001-11-26 |   115 |
|      8 | Henry | 2001-12-01 |   197 |
|     10 | Henry | 2001-11-30 |   203 |
|      2 | Suzi  | 2001-11-29 |   391 |
|      7 | Suzi  | 2001-12-02 |   502 |
|      1 | Ben   | 2001-11-30 |   152 |
|      5 | Ben   | 2001-11-29 |   131 |
|      9 | Ben   | 2001-12-02 |    79 |
+--------+-------+------------+-------+
```

FIELD() можно использовать и для подстрок столбцов. Чтобы упорядочить элементы таблицы housewares по стране изготовления в порядке US, UK, JP, SG, сделайте так:

```
mysql> SELECT id, description FROM housewares
    -> ORDER BY FIELD(RIGHT(id,2),'US','UK','JP','SG');
+------------+-----------------+
| id         | description     |
+------------+-----------------+
| DIN40672US | dining table    |
| BTH00485US | shower stall    |
| KIT00372UK | garbage disposal |
| KIT01729JP | microwave oven  |
| BTH00415JP | lavatory        |
| BED00038SG | bedside lamp    |
+------------+-----------------+
```

Если обобщить, FIELD() может применяться для сортировки любых значений в произвольном порядке, если естественный порядок этих значений не представляет интереса.

6.18. Сортировка значений ENUM

Задача

Значения ENUM не сортируются так, как другие строковые столбцы.

Решение

Поймите, как они работают, и используйте их возможности себе во благо.

Обсуждение

ENUM считается строковым типом, но значения ENUM обладают особым свойством: они хранятся в числовом формате, причем числовые значения отсортированы именно в том порядке, в каком они перечислены в определении таблицы. Эти числовые значения определяют порядок сортировки вывода перечислимых типов, что может быть полезным. Предположим, что у вас есть таблица weekday, содержащая перечислимый столбец day, который включает в себя названия дней недели:

```
CREATE TABLE weekday
(
   day ENUM('Sunday','Monday','Tuesday','Wednesday','Thursday','Friday','Saturday')
);
```

Внутренний формат хранения значений таков: MySQL сопоставляет перечисленным значениям от Sunday до Saturday числовые значения от 1 до 7. Чтобы убедиться в этом, создайте таблицу, используя только что рассмотренное определение, и вставьте в нее запись для каждого дня недели. Чтобы увидеть эффект сортировки, сделаем порядок ввода отличным от порядка значений в списке, вводя дни произвольным образом:

```
mysql> INSERT INTO weekday (day) VALUES('Monday'),('Friday'),
    -> ('Tuesday'), ('Sunday'), ('Thursday'), ('Saturday'), ('Wednesday');
```

Затем выведем значения и как строки, и как внутренние числовые значения (чтобы получить последние, добавьте к строкам 0, инициируя тем самым преобразование строки в число):

```
mysql> SELECT day, day+0 FROM weekday;
+-----------+-------+
| day       | day+0 |
+-----------+-------+
| Monday    |     2 |
| Friday    |     6 |
| Tuesday   |     3 |
| Sunday    |     1 |
| Thursday  |     5 |
| Saturday  |     7 |
| Wednesday |     4 |
+-----------+-------+
```

Поскольку запрос не содержит инструкцию ORDER BY, записи выводятся неупорядоченными. Если добавить инструкцию ORDER BY day, будет очевидно, что MySQL выполняет сортировку по внутренним числовым значениям:

```
mysql> SELECT day, day+0 FROM weekday ORDER BY day;
+-----------+-------+
| day       | day+0 |
+-----------+-------+
| Sunday    |     1 |
| Monday    |     2 |
| Tuesday   |     3 |
```

```
| Wednesday |     4 |
| Thursday  |     5 |
| Friday    |     6 |
| Saturday  |     7 |
+-----------+-------+
```

Что тогда делать, если требуется вывести значения ENUM в лексическом порядке? Используйте функцию CONCAT(), чтобы заставить MySQL воспринимать их как строки. Обычно CONCAT() принимает несколько аргументов и объединяет их в одну строку. Но функцию можно использовать и с одним аргументом, что удобно в тех случаях, когда единственное, что вам нужно от CONCAT(), это ее способность выводить строковый результат:

```
mysql> SELECT day, day+0 FROM weekday ORDER BY CONCAT(day);
+-----------+-------+
| day       | day+0 |
+-----------+-------+
| Friday    |     6 |
| Monday    |     2 |
| Saturday  |     7 |
| Sunday    |     1 |
| Thursday  |     5 |
| Tuesday   |     3 |
| Wednesday |     4 |
+-----------+-------+
```

Если вы всегда (или почти всегда) сортируете неперечислимый столбец в определенном нелексическом порядке, подумайте о том, чтобы изменить его тип на ENUM, перечислив его значения в соответствующем порядке. Создадим таблицу color, содержащую строковый столбец, и заполним ее тестовыми данными:

```
mysql> CREATE TABLE color (name CHAR(10));
mysql> INSERT INTO color (name) VALUES ('blue'),('green'),
    -> ('indigo'),('orange'),('red'),('violet'),('yellow');
```

Если сейчас выполнить сортировку по столбцу name, ее порядок будет лексическим, так как столбец содержит значения типа CHAR:

```
mysql> SELECT name FROM color ORDER BY name;
+--------+
| name   |
+--------+
| blue   |
| green  |
| indigo |
| orange |
| red    |
| violet |
| yellow |
+--------+
```

Теперь предположим, что вы хотите упорядочить столбец по цветам в том порядке, в котором они расположены в радуге. (Этот порядок задается именем «Roy G. Biv», последовательность букв которого совпадает с первыми буквами цветов радуги.) В качестве одного из способов можно предложить использовать функцию FIELD():

```
mysql> SELECT name FROM color
    -> ORDER BY
    -> FIELD(name,'red','orange','yellow','green','blue','indigo','violet');
+--------+
| name   |
+--------+
| red    |
| orange |
| yellow |
| green  |
| blue   |
| indigo |
| violet |
+--------+
```

Чтобы сделать то же самое, не прибегая к помощи FIELD(), используйте предложение ALTER TABLE для преобразования столбца name в тип ENUM, в котором цвета перечислены в нужном порядке:

```
mysql> ALTER TABLE color
    -> MODIFY name
    -> ENUM('red','orange','yellow','green','blue','indigo','violet');
```

После преобразования таблицы сортировка по столбцу name приводит к «радужному» упорядочиванию без каких бы то ни было дополнительных действий:

```
mysql> SELECT name FROM color ORDER BY name;
+--------+
| name   |
+--------+
| red    |
| orange |
| yellow |
| green  |
| blue   |
| indigo |
| violet |
+--------+
```

7

Формирование итогов

7.0. Введение

Системы управления базами данных удобны для хранения и извлечения записей; кроме того, они могут суммировать информацию, представляя ее в более сжатом виде. Итоговая информация необходима, когда вы хотите получить представление о чем-то в целом, не вдаваясь в детали. Суммарная информация обычно проще для восприятия, чем длинные списки записей. Вы можете получать ответы на вопросы типа «Сколько?», «Какова общая сумма?» или «Каков диапазон значений?» Если вы занимаетесь коммерческой деятельностью, вам может понадобиться информация о том, сколько у вас клиентов в каждом штате или каков ежемесячный объем продаж. Вы могли бы получить данные по штатам, выведя список записей о клиентах и подсчитав их самостоятельно, но это не имеет смысла, раз MySQL может все сделать сама. Не очень удобно для вычисления объема продаж вручную складывать суммы заказов, беря их из соответствующих строк. Положитесь на MySQL.

Приведенные примеры требуют формирования двух наиболее распространенных видов итоговой информации. В первом примере (количество клиентов по штатам) используется просто счетчик. Содержимое каждой записи требуется только для внесения ее в соответствующую группу или категорию для подсчета количества. Такие итоги, по сути, являются гистограммами: записи разбиваются на несколько групп, вычисляется количество элементов в каждой из них. Второй пример (объем продаж за месяц) – это вычисляемая итоговая информация, зависящая от содержания записи: итоги продаж получаются сложением значений отдельных записей.

Суммарная информация может представлять собой не только счетчик или сумму, но и просто список уникальных значений. Такой список необходим тогда, когда вас не интересует, сколько присутствует экземпляров каждого значения, важно лишь то, *какие* значения имеются. Например, если вы хотите узнать, в каких штатах у вас есть клиенты, вам нужен список неповторяющихся названий штатов из записей о клиентах, а не список, включающий

значение штата каждой записи. Иногда одни суммарные операции могут применяться к результатам других: например, чтобы определить, в скольких штатах живут ваши клиенты, сначала сформируйте список уникальных штатов, зачем подсчитайте их количество.

Возможность формирования суммарной информации определяется типом обрабатываемых данных. Подсчет количества может выполняться для любых значений, будь то строки символов, числа или даты. Суммы и средние значения могут быть получены только для чисел. Вы можете сосчитать количество штатов, в которых у вас есть клиенты, для подготовки демографического анализа, но не можете складывать и усреднять названия штатов, в этом нет смысла.

В операциях формирования итогов задействованы следующие конструкции SQL:

- Чтобы вычислить итоговое значение для набора отдельных величин, используйте одну из так называемых агрегирующих функций. Название функций объясняется тем, что они работают с агрегатами (группами) значений. К агрегирующим функциям относятся: `COUNT()`, вычисляющая количество записей или значений в результате запроса; `MIN()` и `MAX()`, определяющие наименьшее и наибольшее значения; `SUM()` и `AVG()`, подсчитывающие суммы и средние значения. Эти функции могут использоваться для вычисления значения для всего множества, или же вы можете (при помощи инструкции `GROUP BY`) разбить строки на подмножества и вычислять итоговые значения для каждого из них.
- Чтобы получить список уникальных значений, используйте `SELECT DISTINCT` вместо `SELECT`.
- Для подсчета количества уникальных значений используйте `COUNT(DISTINCT)` вместо `COUNT()`.

Рецепты этой главы сначала познакомят вас с элементарными приемами получения итоговой информации, затем будет показано, как выполнять более сложные операции. Примеры формирования итогов, связанные с выполнением соединений и статистическими операциями, встретятся вам и позже (см. главы 12 и 13).

В примерах используются таблицы `driver_log` и `mail`. Вы уже много раз встречались с ними в главе 6, так что должны быть хорошо с ними знакомы. Кроме того, в главе часто используется и третья таблица, `states`, строки которой содержат информацию о каждом из штатов США:

```
mysql> SELECT * FROM states ORDER BY name;
+---------------+--------+------------+----------+
| name          | abbrev | statehood  | pop      |
+---------------+--------+------------+----------+
| Alabama       | AL     | 1819-12-14 |  4040587 |
| Alaska        | AK     | 1959-01-03 |   550043 |
| Arizona       | AZ     | 1912-02-14 |  3665228 |
| Arkansas      | AR     | 1836-06-15 |  2350725 |
| California    | CA     | 1850-09-09 | 29760021 |
```

```
| Colorado       | CO     | 1876-08-01 |  3294394 |
| Connecticut    | CT     | 1788-01-09 |  3287116 |
...
```

В столбцах `name` и `abbrev` приведены полное название штата и соответствующая аббревиатура. Столбец `statehood` хранит дату вступления штата в США. Столбец `pop` представляет население штата на 1 апреля 1990 года по данным переписи.

Иногда будут использоваться и другие таблицы. Большинство из них можно создать при помощи сценариев каталога *tables* дистрибутива `recipes`. Для создания таблиц, содержащих информацию бейсбольной базы данных *baseball1.com*, следуйте инструкциям из каталога *baseball1*. Таблица `kjv` была описана в рецепте 4.11.

7.1. Суммирование с помощью функции COUNT()

Задача

Вы хотите вычислить количество строк таблицы, количество строк, удовлетворяющих некоторому условию, или узнать, сколько раз встречается определенное значение.

Решение

Используйте функцию `COUNT()`.

Обсуждение

Чтобы вычислить количество строк во всей таблице или количество строк, удовлетворяющих условию, используйте функцию `COUNT()`. Например, чтобы вывести содержимое записей таблицы, вы выполняете запрос `SELECT *`, а для того чтобы вместо этого сосчитать их количество, создайте запрос `SELECT COUNT(*)`. Если в запросе нет инструкции `WHERE`, то будут сосчитаны все записи таблицы, как в следующем запросе, выводящем количество строк таблицы `driver_log`:

```
mysql> SELECT COUNT(*) FROM driver_log;
+----------+
| COUNT(*) |
+----------+
|       10 |
+----------+
```

Если вы не знаете, сколько штатов в США, вам поможет такой запрос:

```
mysql> SELECT COUNT(*) FROM states;
+----------+
| COUNT(*) |
+----------+
|       50 |
+----------+
```

COUNT(*) без инструкции WHERE очень быстро выполняется для таблиц ISAM или MyISAM. А вот для таблиц BDB или InnoDB функцию лучше не использовать; запрос проводит полный просмотр таблиц этих типов, поэтому для больших таблиц операция может быть медленной. Если вас интересует только приблизительное количество строк таблицы, и вы работаете с версией MySQL 3.23 или выше, то можете избежать полного просмотра, используя предложение SHOW TABLE STATUS и исследуя значение Rows вывода. Если бы таблица states относилась к типу InnoDB, результат запроса выглядел бы как-то так:

```
mysql> SHOW TABLE STATUS FROM cookbook LIKE 'states'\G
*************************** 1. row ***************************
           Name: states
           Type: InnoDB
     Row_format: Dynamic
           Rows: 50
 Avg_row_length: 327
    Data_length: 16384
Max_data_length: NULL
   Index_length: 0
      Data_free: 0
 Auto_increment: NULL
    Create_time: NULL
    Update_time: NULL
     Check_time: NULL
 Create_options:
        Comment: InnoDB free: 479232 kB
```

Чтобы сосчитать количество строк, удовлетворяющих некоторым условиям, укажите эти условия в инструкции WHERE. Условия могут быть любыми, благодаря чему COUNT() умеет отвечать на множество разных вопросов:

- Сколько раз водитель проезжал более 200 миль за день?

```
mysql> SELECT COUNT(*) FROM driver_log WHERE miles > 200;
+----------+
| COUNT(*) |
+----------+
|        4 |
+----------+
```

- Сколько дней была за рулем Suzi?

```
mysql> SELECT COUNT(*) FROM driver_log WHERE name = 'Suzi';
+----------+
| COUNT(*) |
+----------+
|        2 |
+----------+
```

- Сколько штатов входило в США в начале XX века?

```
mysql> SELECT COUNT(*) FROM states WHERE statehood < '1900-01-01';
```

```
+----------+
| COUNT(*) |
+----------+
|       45 |
+----------+
```

- Сколько из этих штатов вступило в Союз в XIX веке?

```
mysql> SELECT COUNT(*) FROM states
    -> WHERE statehood BETWEEN '1800-01-01' AND '1899-12-31';
+----------+
| COUNT(*) |
+----------+
|       29 |
+----------+
```

На самом деле функцию COUNT() можно применять в двух формах. Использованная нами ранее (*) считает строки. Вторая же форма, COUNT(*выражение*), принимает как аргумент имя столбца или выражение и вычисляет количество значений не-NULL. Следующий запрос показывает, как вывести для таблицы и счетчик строк, и количество значений не-NULL одного из ее столбцов:

```
SELECT COUNT(*), COUNT(mycol) FROM mytbl;
```

Тот факт, что COUNT(*выражение*) не учитывает значения NULL, можно использовать при получении нескольких счетчиков для одного набора данных. Чтобы в одном запросе вычислить количество поездок в выходные дни (субботу и воскресенье) в таблице driver_log, сделайте следующее:

```
mysql> SELECT
    -> COUNT(IF(DAYOFWEEK(trav_date)=7,1,NULL)) AS 'Saturday trips',
    -> COUNT(IF(DAYOFWEEK(trav_date)=1,1,NULL)) AS 'Sunday trips'
    -> FROM driver_log;
+----------------+--------------+
| Saturday trips | Sunday trips |
+----------------+--------------+
|              1 |            2 |
+----------------+--------------+
```

Чтобы отдельно сосчитать количество поездок в рабочие и выходные дни, выполните такой запрос:

```
mysql> SELECT
    -> COUNT(IF(DAYOFWEEK(trav_date) IN (1,7),1,NULL)) AS 'weekend trips',
    -> COUNT(IF(DAYOFWEEK(trav_date) IN (1,7),NULL,1)) AS 'weekday trips'
    -> FROM driver_log;
+---------------+---------------+
| weekend trips | weekday trips |
+---------------+---------------+
|             3 |             7 |
+---------------+---------------+
```

Выражения IF() определяют, должно ли быть сосчитано значение каждого столбца. Если да, то выражение оценивается в 1, и COUNT() включает его в итог. Если же нет, то выражение оценивается в 0, и COUNT() его игнорирует.

Получается, что вычисляется количество значений, удовлетворяющих условию, заданному в первом аргументе IF().

См. также

Различие между COUNT(*) и COUNT(*выражение*) будет рассматриваться далее в разделе 7.8 «Итоги и значения NULL».

7.2. Суммирование при помощи функций MIN() и MAX()

Задача

Вам нужно определить наименьшее или наибольшее значения множества данных.

Решение

Используйте функцию MIN() для нахождения минимального значения, а функцию MAX() – для нахождения максимального значения.

Обсуждение

Нахождение наибольшего и наименьшего значений похоже на сортировку, только вместо того, чтобы выводить все упорядоченное множество, выбираем всего одно значение с одного или другого конца отсортированного списка. Такие операции выполняются для получения ответов на вопросы о наибольшем, наименьшем, самом молодом и старом, наиболее дорогом и дешевом и т. д. Одним из способов нахождения подобных значений является использование функций MIN() и MAX(). (Кроме того, на эти вопросы можно ответить при помощи инструкции LIMIT; см. рецепты 3.16 и 3.18.)

Поскольку функции MIN() и MAX() определяют экстремальные значения множества, их можно использовать для определения диапазонов:

- Какой диапазон дат представлен строками таблицы mail? Каковы размеры самого короткого и самого длинного сообщений?

```
mysql> SELECT
    -> MIN(t) AS earliest, MAX(t) AS latest,
    -> MIN(size) AS smallest, MAX(size) AS largest
    -> FROM mail;
+---------------------+---------------------+----------+---------+
| earliest            | latest              | smallest | largest |
+---------------------+---------------------+----------+---------+
| 2001-05-11 10:15:08 | 2001-05-19 22:21:51 |      271 | 2394482 |
+---------------------+---------------------+----------+---------+
```

- Какая поездка из таблицы driver_log была самой короткой, а какая самой длинной?

```
mysql> SELECT MIN(miles) AS shortest, MAX(miles) AS longest
    -> FROM driver_log;
+----------+---------+
| shortest | longest |
+----------+---------+
|       79 |     502 |
+----------+---------+
```

- Каково наибольшее и наименьшее население штата США?

```
mysql> SELECT MIN(pop) AS 'fewest people', MAX(pop) AS 'most people'
    -> FROM states;
+---------------+-------------+
| fewest people | most people |
+---------------+-------------+
|        453588 |    29760021 |
+---------------+-------------+
```

- Название какого штата является первым в лексическом порядке, а какого – последним?

```
mysql> SELECT MIN(name), MAX(name) FROM states;
+-----------+-----------+
| MIN(name) | MAX(name) |
+-----------+-----------+
| Alabama   | Wyoming   |
+-----------+-----------+
```

Функции `MIN()` и `MAX()` необязательно должны применяться непосредственно к значениям столбцов. Они работают и с выражениями, и со значениями, полученными из значений столбцов. Например, чтобы определить длины самого короткого и самого длинного названий штата, выполните следующий запрос:

```
mysql> SELECT MIN(LENGTH(name)) AS shortest, MAX(LENGTH(name)) AS longest
    -> FROM states;
+----------+---------+
| shortest | longest |
+----------+---------+
|        4 |      14 |
+----------+---------+
```

7.3. Суммирование при помощи функций SUM() и AVG()

Задача

Вам нужно найти сумму чисел или вычислить их среднее значение.

Решение

Используйте функцию `SUM()` или `AVG()`.

Обсуждение

Функции SUM() и AVG() выводят сумму и среднее значение для набора данных. Используйте их, чтобы получить ответы на вопросы:

- Каковы общий объем почтового трафика и средний размер сообщения?

```
mysql> SELECT SUM(size) AS 'total traffic',
    -> AVG(size) AS 'average message size'
    -> FROM mail;
+---------------+----------------------+
| total traffic | average message size |
+---------------+----------------------+
|       3798185 |          237386.5625 |
+---------------+----------------------+
```

- Сколько всего миль проехали водители из таблицы driver_log? Каким был средний дневной пробег?

```
mysql> SELECT SUM(miles) AS 'total miles',
    -> AVG(miles) AS 'average miles/day'
    -> FROM driver_log;
+-------------+-------------------+
| total miles | average miles/day |
+-------------+-------------------+
|        2166 |          216.6000 |
+-------------+-------------------+
```

- Каково общее население США?

```
mysql> SELECT SUM(pop) FROM states;
+-----------+
| SUM(pop)  |
+-----------+
| 248102973 |
+-----------+
```

  (Значение представляет население США на 1 апреля 1990 года. Приведенные цифры отличаются от показателей переписи населения, так как таблица states не содержит данных о Вашингтоне, округ Колумбия.)

Функции SUM() и AVG() являются строго числовыми, то есть не могут применяться к строкам или значениям времени. Правда, в некоторых случаях можно преобразовать нечисловые значения в числовые. Предположим, что таблица хранит значения TIME, представляющие затраченное время:

```
mysql> SELECT t1 FROM time_val;
+----------+
| t1       |
+----------+
| 15:00:00 |
| 05:01:30 |
| 12:30:20 |
+----------+
```

Чтобы вычислить общее потраченное время, сначала используем функцию TIME_TO_SEC() для преобразования значений в секунды. Результат также будет выражен в секундах, так что если вы предпочитаете формат TIME, передайте результат в SEC_TO_TIME():

```
mysql> SELECT SUM(TIME_TO_SEC(t1)) AS 'total seconds',
    -> SEC_TO_TIME(SUM(TIME_TO_SEC(t1))) AS 'total time'
    -> FROM time_val;
+---------------+------------+
| total seconds | total time |
+---------------+------------+
|        117110 | 32:31:50   |
+---------------+------------+
```

См. также

Функции SUM() и AVG() особенно полезны в приложениях, занимающихся вычислением статистических характеристик. Мы еще поговорим о них в главе 13, где они будут использоваться совместно с STD() – функцией, вычисляющей стандартные флуктуации.

7.4. Использование ключевого слова DISTINCT для удаления дубликатов

Задача

Вы хотите узнать, какие значения входят в набор данных, не перечисляя повторяющиеся значения многократно. Или хотите узнать, сколько уникальных значений входит в множество.

Решение

Используйте ключевое слово DISTINCT для выбора уникальных значений и конструкцию COUNT(DISTINCT) – для подсчета их количества.

Обсуждение

Удаление дубликатов для получения информации о том, какие значения или строки входят в набор данных, – это суммарная операция, не требующая использования агрегирующей функции. Используем ключевое слово DISTINCT (или его синоним DISTINCTROW). DISTINCT сокращает результат запроса и часто используется в сочетании с ORDER BY для расположения значений в более удобном порядке. Например, если вы хотите узнать имена водителей из таблицы driver_log, выполните такой запрос:

```
mysql> SELECT DISTINCT name FROM driver_log ORDER BY name;
+-------+
| name  |
+-------+
```

```
| Ben   |
| Henry |
| Suzi  |
+-------+
```

Запрос без использования ключевого слова DISTINCT выводит те же самые имена, но в менее удобной форме:

```
mysql> SELECT name FROM driver_log;
+-------+
| name  |
+-------+
| Ben   |
| Suzi  |
| Henry |
| Henry |
| Ben   |
| Henry |
| Suzi  |
| Henry |
| Ben   |
| Henry |
+-------+
```

Если вы хотите узнать, сколько разных водителей включено в таблицу, используйте конструкцию COUNT(DISTINCT):

```
mysql> SELECT COUNT(DISTINCT name) FROM driver_log;
+----------------------+
| COUNT(DISTINCT name) |
+----------------------+
|                    3 |
+----------------------+
```

Конструкция COUNT(DISTINCT) игнорирует значения NULL. Если вы хотите сосчитать и значения NULL (если они присутствуют), сделайте следующее:

```
COUNT(DISTINCT знач) + IF(COUNT(IF(знач IS NULL,1,NULL))=0,0,1)
```

Того же эффекта можно достичь при помощи таких выражений:

```
COUNT(DISTINCT знач) + IF(SUM(ISNULL(знач))=0,0,1)
COUNT(DISTINCT знач) + (SUM(ISNULL(знач))!=0)
```

Конструкция COUNT(DISTINCT) доступна, начиная с MySQL версии 3.23.2. Если вы работаете с более ранней версией, необходимо использовать какой-то обходной маневр для вычисления количества строк запроса SELECT DISTINCT. Можно, например, выбрать уникальные значения в отдельную таблицу, а затем применить COUNT(*) для подсчета количества строк новой таблицы.

Запросы DISTINCT часто используются в сочетании с агрегирующими функциями для получения более полного описания имеющихся данных. Например, если применить COUNT(*) к таблице customer, вы узнаете количество своих клиентов, применив DISTINCT к значениям таблицы state, вы узнаете,

в каких штатах живут ваши клиенты. Если же выполнить COUNT(DISTINCT) к значениям state, то будет выведено количество штатов, в которых есть ваши клиенты.

Если использовать DISTINCT для нескольких столбцов, то будут выведены уникальные комбинации значений этих столбцов, а COUNT(DISTINCT) вычислит количество таких комбинаций. Следующий запрос находит в таблице mail различные пары отправитель-получатель (srcuser-dstuser) и считает их количество:

```
mysql> SELECT DISTINCT srcuser, dstuser FROM mail
    -> ORDER BY srcuser, dstuser;
+---------+---------+
| srcuser | dstuser |
+---------+---------+
| barb    | barb    |
| barb    | tricia  |
| gene    | barb    |
| gene    | gene    |
| gene    | tricia  |
| phil    | barb    |
| phil    | phil    |
| phil    | tricia  |
| tricia  | gene    |
| tricia  | phil    |
+---------+---------+
mysql> SELECT COUNT(DISTINCT srcuser, dstuser) FROM mail;
+----------------------------------+
| COUNT(DISTINCT srcuser, dstuser) |
+----------------------------------+
|                               10 |
+----------------------------------+
```

Можно применять DISTINCT не только к столбцам, но и к выражениям. Чтобы вычислить количество часов дня, в которые отправлялись сообщения таблицы mail, будем учитывать неповторяющиеся значения HOUR():

```
mysql> SELECT COUNT(DISTINCT HOUR(t)) FROM mail;
+-------------------------+
| COUNT(DISTINCT HOUR(t)) |
+-------------------------+
|                      12 |
+-------------------------+
```

Чтобы узнать, какие именно это были часы, выведем их список:

```
mysql> SELECT DISTINCT HOUR(t) FROM mail ORDER BY 1;
+---------+
| HOUR(t) |
+---------+
|       7 |
|       8 |
|       9 |
```

```
|      10 |
|      11 |
|      12 |
|      13 |
|      14 |
|      15 |
|      17 |
|      22 |
|      23 |
+---------+
```

Обратите внимание на то, что запрос не сообщает нам, сколько сообщений было отправлено в каждый из часов. Об этом рассказано в рецепте 7.15.

7.5. Поиск значений, связанных с минимальным и максимальным значениями

Задача

Вы хотите узнать значения других столбцов строки, содержащей минимальное или максимальное значение.

Решение

Используйте два запроса и переменную SQL. Или «прием MAX-CONCAT». Или соединение (join).

Обсуждение

Функции `MIN()` и `MAX()` находят границы диапазона значений. Бывают ситуации, когда после нахождения минимального или максимального значения вам нужно получить и другие значения той строки, в которой оно встретилось. Например, наибольшее население штата можно получить так:

```
mysql> SELECT MAX(pop) FROM states;
+----------+
| MAX(pop) |
+----------+
| 29760021 |
+----------+
```

Но при этом вы не знаете, какой именно штат имеет такое население. Кажется, что можно вывести необходимую информацию так:

```
mysql> SELECT name, MAX(pop) FROM states WHERE pop = MAX(pop);
ERROR 1111 at line 1: Invalid use of group function
```

Практически каждый рано или поздно пробует сделать нечто подобное, но ничего не получается, поскольку агрегирующие функции, такие как `MIN()` и `MAX()`, нельзя использовать в инструкциях `WHERE`. В приведенном выше предложении мы пытались сначала определить, какая запись содержит макси-

мальное значение для численности населения, а затем вывести название соответствующего штата. Проблема в том, что мы-то хорошо понимаем, что имели в виду, а вот для MySQL все это не имеет никакого смысла. Запрос не удается выполнить, так как MySQL использует инструкцию WHERE для того, чтобы определить, какие записи выбрать, но при этом узнаёт значение агрегирующей функции только *после* того, как выбраны записи, по которым определяется значение функции! То есть предложение является внутренне противоречивым. Можно было бы справиться с этим, используя подзапрос, однако MySQL будет поддерживать их только начиная с версии 4.1. Между тем, можно разбить решение задачи на два этапа: первый запрос будет извлекать максимальное значение в переменную SQL, а второй – ссылаться на эту переменную в своей инструкции WHERE:

```
mysql> SELECT @max := MAX(pop) FROM states;
mysql> SELECT @max AS 'highest population', name FROM states WHERE pop = @max;
+--------------------+------------+
| highest population | name       |
+--------------------+------------+
| 29760021           | California |
+--------------------+------------+
```

Этот прием работает даже тогда, когда минимальное или максимальное значение не содержится непосредственно в строке, а получается из нее. Если вы хотите узнать длину самого короткого стиха в King James Version, ее легко можно вычислить так:

```
mysql> SELECT MIN(LENGTH(vtext)) FROM kjv;
+--------------------+
| MIN(LENGTH(vtext)) |
+--------------------+
|                 11 |
+--------------------+
```

Если же вы хотите получить ответ на вопрос: «Что это за стих?», выполните следующий запрос:

```
mysql> SELECT @min := MIN(LENGTH(vtext)) FROM kjv;
mysql> SELECT bname, cnum, vnum, vtext FROM kjv WHERE LENGTH(vtext) = @min;
+-------+------+------+-------------+
| bname | cnum | vnum | vtext       |
+-------+------+------+-------------+
| John  |   11 |   35 | Jesus wept. |
+-------+------+------+-------------+
```

Есть и другой способ, который можно использовать для нахождения значений, связанных с минимумом или максимумом. В справочном руководстве по MySQL он называется «прием MAX-CONCAT» («MAX-CONCAT trick»). Способ не очень изящен, но может оказаться полезным тем, кто работает с версиями MySQL, не допускающими использования переменных SQL. Технология такова: используя CONCAT(), добавляем столбец к столбцу суммирования, находим максимум получившихся значений при помощи MAX() и из-

влекаем несуммируемую часть значения результата. Например, чтобы вывести название штата с наибольшей численностью населения, можно выбрать максимальное составное значение столбцов pop и name, а затем извлечь из него составляющую name. Давайте будем действовать последовательно. Сначала определим максимальную численность населения, чтобы узнать размер этого значения:

```
mysql> SELECT MAX(pop) FROM states;
+----------+
| MAX(pop) |
+----------+
| 29760021 |
+----------+
```

Восемь символов. Такая информация необходима для того, чтобы соответствующим образом разместить составляющую названия штата в комбинированном значении население-штат, обеспечив тем самым его корректное извлечение. Составляющая названия должна начинаться с фиксированной позиции. Зная, что максимальное значение численности населения состоит из восьми символов, дополним столбец pop пробелами до восьми символов, тогда значения name всегда будут начинаться с девятой позиции.

Но будьте внимательны при дополнении значений численности населения пробелами. Значения, порождаемые функцией CONCAT(), являются строками, поэтому функция MAX() при сортировке будет рассматривать значения население-штат как строки. Если выровнять значения pop по левому краю, дополнив их пробелами справа с помощью RPAD(), то составные значения будут такими:

```
mysql> SELECT CONCAT(RPAD(pop,8,' '),name) FROM states;
+------------------------------+
| CONCAT(RPAD(pop,8,' '),name) |
+------------------------------+
| 4040587 Alabama              |
| 550043  Alaska               |
| 3665228 Arizona              |
| 2350725 Arkansas             |
...
```

Эти значения будут упорядочиваться в лексическом порядке, что удобно для нахождения наибольшего строкового значения с помощью функции MAX(). Но значения pop – это числа, и хотелось бы, чтобы значения упорядочивались как числа. Чтобы лексическое упорядочивание соответствовало числовому, необходимо выровнять значения численности населения вправо, дополнив их пробелами слева при помощи LPAD():

```
mysql> SELECT CONCAT(LPAD(pop,8,' '),name) FROM states;
+------------------------------+
| CONCAT(LPAD(pop,8,' '),name) |
+------------------------------+
|  4040587Alabama              |
```

```
|   550043Alaska              |
|  3665228Arizona             |
|  2350725Arkansas            |
...
```

Теперь используем выражение `CONCAT()` в сочетании с `MAX()` для нахождения значения с наибольшей составляющей численности населения:

```
mysql> SELECT MAX(CONCAT(LPAD(pop,8,' '),name)) FROM states;
+-----------------------------------+
| MAX(CONCAT(LPAD(pop,8,' '),name)) |
+-----------------------------------+
| 29760021California                |
+-----------------------------------+
```

Чтобы получить конечный результат (название штата с наибольшим населением), извлеките из максимального составного значения подстроку, начинающуюся с девятого символа:

```
mysql> SELECT SUBSTRING(MAX(CONCAT(LPAD(pop,8,' '),name)),9) FROM states;
+-----------------------------------------------+
| SUBSTRING(MAX(CONCAT(LPAD(pop,8,' '),name)),9) |
+-----------------------------------------------+
| California                                    |
+-----------------------------------------------+
```

Естественно, для хранения промежуточного результата гораздо удобнее использовать переменную SQL. Это и более эффективно, так как не приходится заниматься объединением значений столбцов для сортировки и разбиением результата на составляющие для вывода.

Существует и еще один способ выбора столбцов из строк, содержащих минимальное или максимальное значение – использование соединения. Извлеките значение в другую таблицу и соедините его с исходной таблицей для выбора соответствующей строки. Чтобы найти запись о штате с максимальной численностью населения, примените такое соединение:

```
mysql> CREATE TEMPORARY TABLE t
    -> SELECT MAX(pop) as maxpop FROM states;
mysql> SELECT states.* FROM states, t WHERE states.pop = t.maxpop;
+------------+--------+------------+----------+
| name       | abbrev | statehood  | pop      |
+------------+--------+------------+----------+
| California | CA     | 1850-09-09 | 29760021 |
+------------+--------+------------+----------+
```

См. также

Дополнительная информация о соединениях приведена в главе 12.

7.6. Управление чувствительностью к регистру функций MIN() и MAX()

Задача

Функции MIN() и MAX() выбирают строки, учитывая их регистр, а вам бы этого не хотелось, или наоборот.

Решение

Измените чувствительность строк к регистру.

Обсуждение

Когда функции MIN() и MAX() применяются к строковым значениям, они выводят результаты, руководствуясь правилами лексического упорядочивания. Чувствительность к регистру влияет на сортировку, а значит, и на функции MIN() и MAX(). В главе 6 использовалась таблица textblob_val, содержащая два столбца, казалось бы, одинаковых значений:

```
mysql> SELECT tstr, bstr FROM textblob_val;
+------+------+
| tstr | bstr |
+------+------+
| aaa  | aaa  |
| AAA  | AAA  |
| bbb  | bbb  |
| BBB  | BBB  |
+------+------+
```

Но столбцы только выглядят одинаково, а ведут себя по-разному. Столбец bstr относится к типу BLOB и чувствителен к регистру. Столбец tstr имеет тип TEXT и нечувствителен к регистру. Поэтому функции MIN() и MAX() необязательно получат одинаковые результаты для двух столбцов:

```
mysql> SELECT MIN(tstr), MIN(bstr) FROM textblob_val;
+-----------+-----------+
| MIN(tstr) | MIN(bstr) |
+-----------+-----------+
| aaa       | AAA       |
+-----------+-----------+
```

Чтобы сделать столбец tstr чувствительным к регистру, применим BINARY:

```
mysql> SELECT MIN(BINARY tstr) FROM textblob_val;
+------------------+
| MIN(BINARY tstr) |
+------------------+
| AAA              |
+------------------+
```

Чтобы сделать столбец bstr не чувствительным к регистру, можно преобразовать все его значения к одному регистру:

```
mysql> SELECT MIN(LOWER(bstr)) FROM textblob_val;
+------------------+
| MIN(LOWER(bstr)) |
+------------------+
| aaa              |
+------------------+
```

К сожалению, в результате выполнения такой операции изменится и отображаемое значение. Если это важно, используйте вместо только что предложенного такой прием (он может выводить немного другой результат):

```
mysql> SELECT @min := MIN(LOWER(bstr)) FROM textblob_val;
mysql> SELECT bstr FROM textblob_val WHERE LOWER(bstr) = @min;
+------+
| bstr |
+------+
| aaa  |
| AAA  |
+------+
```

7.7. Разбиение итогов на подгруппы

Задача

Вы хотите получить итоговую информацию для каждой подгруппы множества строк, а не одно общее итоговое значение.

Решение

Используйте инструкцию GROUP BY для распределения строк по группам.

Обсуждение

Пока что рассматривались запросы, формирующие суммарные значения для всех строк результирующего множества. Например, следующий запрос вычисляет количество ежедневных записей таблицы driver_log, то есть общее количество дней, когда водители были в дороге:

```
mysql> SELECT COUNT(*) FROM driver_log;
+----------+
| COUNT(*) |
+----------+
|       10 |
+----------+
```

Но иногда необходимо разбить множество строк на подгруппы и проводить суммирование в каждой группе. Для этого используются агрегирующие функции в сочетании с инструкцией GROUP BY. Чтобы вычислить количество

дней, проведенных за рулем каждым водителем, сгруппируйте строки по имени водителя, сосчитайте количество строк в каждой группе и выведите имена вместе со счетчиками:

```
mysql> SELECT name, COUNT(name) FROM driver_log GROUP BY name;
+-------+-------------+
| name  | COUNT(name) |
+-------+-------------+
| Ben   |           3 |
| Henry |           5 |
| Suzi  |           2 |
+-------+-------------+
```

Данный запрос суммирует тот же столбец, который группируется (`name`), но это совсем необязательно. Предположим, вам нужно быстро извлечь из таблицы `driver_log` следующую информацию: общее количество пройденных миль и средний дневной пробег для каждого водителя. Для группировки строк снова будем использовать столбец `name`, но суммирующие функции теперь будут работать со значениями `miles`:

```
mysql> SELECT name,
    -> SUM(miles) AS 'total miles',
    -> AVG(miles) AS 'miles per day'
    -> FROM driver_log GROUP BY name;
+-------+-------------+---------------+
| name  | total miles | miles per day |
+-------+-------------+---------------+
| Ben   |         362 |      120.6667 |
| Henry |         911 |      182.2000 |
| Suzi  |         893 |      446.5000 |
+-------+-------------+---------------+
```

Используйте столько столбцов группировки, сколько требуется для достижения необходимой детальности итоговой информации. Следующий запрос выводит грубую оценку, вычисляя, сколько сообщений было отправлено каждым пользователем из таблицы `mail`:

```
mysql> SELECT srcuser, COUNT(*) FROM mail
    -> GROUP BY srcuser;
+---------+----------+
| srcuser | COUNT(*) |
+---------+----------+
| barb    |        3 |
| gene    |        6 |
| phil    |        5 |
| tricia  |        2 |
+---------+----------+
```

Чтобы получить более конкретные данные и узнать, сколько сообщений каждый пользователь отправил с каждого хоста, используйте группировку по двум столбцам. В результате получатся вложенные группы (группы внутри групп):

Получение неповторяющихся значений без помощи DISTINCT

Если вы используете инструкцию GROUP BY, не выбирая значение какой бы то ни было агрегирующей функции, то тем самым добиваетесь того же эффекта, что и при использовании DISTINCT (хотя явно DISTINCT не используется):

```
mysql> SELECT name FROM driver_log GROUP BY name;
+-------+
| name  |
+-------+
| Ben   |
| Henry |
| Suzi  |
+-------+
```

Обычно в запросах такого типа выбирается какое-то суммарное значение (например, вызывается COUNT(name) для подсчета количества вхождений каждого имени), но этого можно и не делать. В результате формируется список уникальных сгруппированных значений. Я лично предпочитаю использовать DISTINCT, так как тогда более понятна цель выполнения запроса. (На самом деле MySQL внутренне отображает DISTINCT-форму запроса на GROUP BY-форму.)

```
mysql> SELECT srcuser, srchost, COUNT(*) FROM mail
    -> GROUP BY srcuser, srchost;
+---------+---------+----------+
| srcuser | srchost | COUNT(*) |
+---------+---------+----------+
| barb    | saturn  |        2 |
| barb    | venus   |        1 |
| gene    | mars    |        2 |
| gene    | saturn  |        2 |
| gene    | venus   |        2 |
| phil    | mars    |        3 |
| phil    | venus   |        2 |
| tricia  | mars    |        1 |
| tricia  | saturn  |        1 |
+---------+---------+----------+
```

В примерах данного раздела для получения групповых итогов использовались функции COUNT(), SUM() и AVG(). Можно применять и функции MIN() и MAX(). Будучи использованными в инструкции GROUP BY, они выводят наименьшее и наибольшее значения в группе. Сгруппируем строки таблицы mail по отправителям сообщений и будем выводить для каждой из них размер максимального сообщения и дату последнего сообщения:

```
mysql> SELECT srcuser, MAX(size), MAX(t) FROM mail GROUP BY srcuser;
```

```
+---------+-----------+---------------------+
| srcuser | MAX(size) | MAX(t)              |
+---------+-----------+---------------------+
| barb    |     98151 | 2001-05-14 14:42:21 |
| gene    |    998532 | 2001-05-19 22:21:51 |
| phil    |     10294 | 2001-05-17 12:49:23 |
| tricia  |   2394482 | 2001-05-14 17:03:01 |
+---------+-----------+---------------------+
```

Можно выполнить группировку по нескольким столбцам и вывести наибольшее значение для каждой комбинации значений этих столбцов. Следующий запрос находит размер самого длинного сообщения среди сообщений каждой пары отправитель-получатель (`srcuser-dstuser`) из таблицы `mail`:

```
mysql> SELECT srcuser, dstuser, MAX(size) FROM mail GROUP BY srcuser, dstuser;
+---------+---------+-----------+
| srcuser | dstuser | MAX(size) |
+---------+---------+-----------+
| barb    | barb    |     98151 |
| barb    | tricia  |     58274 |
| gene    | barb    |      2291 |
| gene    | gene    |     23992 |
| gene    | tricia  |    998532 |
| phil    | barb    |     10294 |
| phil    | phil    |      1048 |
| phil    | tricia  |      5781 |
| tricia  | gene    |    194925 |
| tricia  | phil    |   2394482 |
+---------+---------+-----------+
```

Используя агрегирующие функции при получении итоговых значений для групп, можно попасть в ловушку. Предположим, вы хотите найти самую длинную поездку каждого водителя в таблице `driver_log`. Выполняем такой запрос:

```
mysql> SELECT name, MAX(miles) AS 'longest trip'
    -> FROM driver_log GROUP BY name;
+-------+--------------+
| name  | longest trip |
+-------+--------------+
| Ben   |          152 |
| Henry |          300 |
| Suzi  |          502 |
+-------+--------------+
```

Но что делать, если вы хотите вывести и дату совершения этой поездки? Можно ли просто добавить столбец `trav_date` в список вывода? К сожалению, нет:

```
mysql> SELECT name, trav_date, MAX(miles) AS 'longest trip'
    -> FROM driver_log GROUP BY name;
+-------+------------+--------------+
| name  | trav_date  | longest trip |
+-------+------------+--------------+
| Ben   | 2001-11-30 |          152 |
```

```
| Henry | 2001-11-29 |          300 |
| Suzi  | 2001-11-29 |          502 |
+-------+------------+--------------+
```

Запрос возвращает результат, но если вы сравните его с данными таблицы (приведенными ниже), то обнаружите, что даты для имен Ben и Henry корректны, а для Suzi – нет:

```
+--------+-------+------------+-------+
| rec_id | name  | trav_date  | miles |
+--------+-------+------------+-------+
|      1 | Ben   | 2001-11-30 |   152 |    ← самая длинная поездка Ben'a
|      2 | Suzi  | 2001-11-29 |   391 |
|      3 | Henry | 2001-11-29 |   300 |    ← самая длинная поездка Henry
|      4 | Henry | 2001-11-27 |    96 |
|      5 | Ben   | 2001-11-29 |   131 |
|      6 | Henry | 2001-11-26 |   115 |
|      7 | Suzi  | 2001-12-02 |   502 |    ← самая длинная поездка Suzi
|      8 | Henry | 2001-12-01 |   197 |
|      9 | Ben   | 2001-12-02 |    79 |
|     10 | Henry | 2001-11-30 |   203 |
+--------+-------+------------+-------+
```

В чем же причина? Почему запрос выводит неправильный результат? Дело в том, что когда вы включаете в запрос инструкцию `GROUP BY`, выбирать можно только значения группируемых столбцов или вычисленные для них итоговые значения. Если вы выводите какие-то дополнительные столбцы, они никак не привязаны к группируемым столбцам, и выводимые для них значения никак не задаются. (Похоже, что в только что приведенном запросе СУБД MySQL просто взяла первую дату для каждого водителя, не заботясь о том, является ли она датой самой длинной поездки.)

Общим решением проблем вывода содержимого строк, связанных с минимальным или максимальным значением, является использование соединения (см. главу 12). Если вы не хотите забегать вперед или не хотите использовать другую таблицу, то можете обратиться к описанному ранее приему MAX-CONCAT. Запрос получается не очень красивым, но выводит корректный результат:

```
mysql> SELECT name,
    -> SUBSTRING(MAX(CONCAT(LPAD(miles,3,' '), trav_date)),4) AS date,
    -> LEFT(MAX(CONCAT(LPAD(miles,3,' '), trav_date)),3) AS 'longest trip'
    -> FROM driver_log GROUP BY name;
+-------+------------+--------------+
| name  | date       | longest trip |
+-------+------------+--------------+
| Ben   | 2001-11-30 | 152          |
| Henry | 2001-11-29 | 300          |
| Suzi  | 2001-12-02 | 502          |
+-------+------------+--------------+
```

7.8. Итоги и значения NULL

Задача

Суммируя набор значений, среди которых могут быть и значения NULL, вы хотите понять, как интерпретировать полученные результаты.

Решение

Осознайте, каким образом агрегирующие функции обрабатывают значения NULL.

Обсуждение

Большинство агрегирующих функций игнорируют значения NULL. Предположим, у вас есть таблица expt, в которую записываются результаты тестов для испытуемых (subject), каждому из которых нужно пройти четыре теста (test); при этом указывается значение NULL, если результат (score) еще не получен:

```
mysql> SELECT subject, test, score FROM expt ORDER BY subject, test;
+---------+------+-------+
| subject | test | score |
+---------+------+-------+
| Jane    | A    |    47 |
| Jane    | B    |    50 |
| Jane    | C    |  NULL |
| Jane    | D    |  NULL |
| Marvin  | A    |    52 |
| Marvin  | B    |    45 |
| Marvin  | C    |    53 |
| Marvin  | D    |  NULL |
+---------+------+-------+
```

Если использовать инструкцию GROUP BY для группировки строк по имени испытуемых, то можно вычислить количество тестов, пройденных каждым из них, а также общий, средний, максимальный и минимальный результаты следующим образом:

```
mysql> SELECT subject,
    -> COUNT(score) AS n,
    -> SUM(score) AS total,
    -> AVG(score) AS average,
    -> MIN(score) AS lowest,
    -> MAX(score) AS highest
    -> FROM expt GROUP BY subject;
+---------+---+-------+---------+--------+---------+
| subject | n | total | average | lowest | highest |
+---------+---+-------+---------+--------+---------+
| Jane    | 2 |    97 | 48.5000 |     47 |      50 |
| Marvin  | 3 |   150 | 50.0000 |     45 |      53 |
+---------+---+-------+---------+--------+---------+
```

Из результатов в столбце n (количество тестов) видно, что запрос обработал только пять значений. Почему? Потому что в этом столбце выводятся только значения, соответствующие не-NULL-результатам тестов каждого испытуемого. Значения, представленные в других итоговых столбцах, также вычислены на основе только не-NULL-результатов тестов.

Вполне логично, что агрегирующие функции игнорируют значения NULL. Если бы они следовали обычным арифметическим правилам SQL, то в результате прибавления NULL к любому другому значению получалось бы значение NULL. Работать с агрегирующими функциями стало бы очень тяжело, поскольку вам приходилось бы каждый раз самостоятельно отфильтровывать значения NULL перед выполнением суммирования, чтобы избежать получения NULL-результата. Да, игнорируя NULL, агрегирующие функции становятся гораздо более привлекательными для использования.

Однако помните, что несмотря на то, что агрегирующие функции игнорируют значения NULL, некоторые из них все же могут получать это значение как результат. Так бывает, если нечего суммировать. Чуть-чуть изменим предыдущий запрос: теперь он выбирает только NULL-результаты тестов, так что агрегирующим функциям не с чем работать:

```
mysql> SELECT subject,
    -> COUNT(score) AS n,
    -> SUM(score) AS total,
    -> AVG(score) AS average,
    -> MIN(score) AS lowest,
    -> MAX(score) AS highest
    -> FROM expt WHERE score IS NULL GROUP BY subject;
+---------+---+-------+---------+--------+---------+
| subject | n | total | average | lowest | highest |
+---------+---+-------+---------+--------+---------+
| Jane    | 0 |     0 |    NULL |   NULL |    NULL |
| Marvin  | 0 |     0 |    NULL |   NULL |    NULL |
+---------+---+-------+---------+--------+---------+
```

Даже в таких условиях агрегирующие функции возвращают наиболее разумное значение. Количество тестов, пройденных каждым испытуемым, и его общий результат равны нулю, нули и выводятся. А вот AVG() возвращает NULL. Среднее значение – это отношение суммы значений к их количеству. Если складывать нечего, вы имеете дело с отношением 0/0, которое не определено. Поэтому для AVG() разумнее всего вернуть NULL. Аналогично, функциям MIN() и MAX() не с чем работать, поэтому они возвращают NULL. Если вы не хотите, чтобы эти функции выводили NULL, используйте IFNULL() для сопоставления им соответствующих значений:

```
mysql> SELECT subject,
    -> COUNT(score) AS n,
    -> SUM(score) AS total,
    -> IFNULL(AVG(score),0) AS average,
    -> IFNULL(MIN(score),'Unknown') AS lowest,
```

```
    -> IFNULL(MAX(score),'Unknown') AS highest
    -> FROM expt WHERE score IS NULL GROUP BY subject;
+---------+---+-------+---------+---------+---------+
| subject | n | total | average | lowest  | highest |
+---------+---+-------+---------+---------+---------+
| Jane    | 0 |     0 |       0 | Unknown | Unknown |
| Marvin  | 0 |     0 |       0 | Unknown | Unknown |
+---------+---+-------+---------+---------+---------+
```

Функция `COUNT()` несколько отличается в своей трактовке значений `NULL` от остальных агрегирующих функций. Как и другие агрегирующие функции, `COUNT(`*выражение*`)` считает только значения не-`NULL`, а `COUNT(*)` считает все строки независимо от их содержимого. Продемонстрируем разницу между двумя формами `COUNT()`:

```
mysql> SELECT COUNT(*), COUNT(score) FROM expt;
+----------+--------------+
| COUNT(*) | COUNT(score) |
+----------+--------------+
|        8 |            5 |
+----------+--------------+
```

Теперь вы знаете, что в таблице `expt` восемь строк, но лишь в пяти из них заполнены значения `score`. Две формы `COUNT()` очень удобно использовать для подсчета недостающих значений – просто найдите разность значений:

```
mysql> SELECT COUNT(*) - COUNT(score) AS missing FROM expt;
+---------+
| missing |
+---------+
|       3 |
+---------+
```

Можно проводить подсчет недостающих и имеющихся значений и для подгрупп. Следующий запрос выполняет операцию для каждого испытуемого. Так вы можете оценить, как далеко продвинулся эксперимент:

```
mysql> SELECT subject,
    -> COUNT(*) AS total,
    -> COUNT(score) AS 'non-missing',
    -> COUNT(*) - COUNT(score) AS missing
    -> FROM expt GROUP BY subject;
+---------+-------+-------------+---------+
| subject | total | non-missing | missing |
+---------+-------+-------------+---------+
| Jane    |     4 |           2 |       2 |
| Marvin  |     4 |           3 |       1 |
+---------+-------+-------------+---------+
```

7.9. Выбор групп только с определенными характеристиками

Задача

Вы хотите вычислить итоги для групп, но результаты вывести только для групп, отвечающих определенным требованиям.

Решение

Используйте инструкцию HAVING.

Обсуждение

Вы уже умеете применять инструкцию WHERE для задания условий, которым должны удовлетворять отдельные записи, выбираемые запросом. Кажется естественным использовать WHERE и для написания условий для итоговых значений. Но ничего не выйдет. Если вы захотите узнать, какой водитель из таблицы driver_log работал больше трех дней, то, вероятно, начнете с такого запроса:

```
mysql> SELECT COUNT(*), name
    -> FROM driver_log
    -> WHERE COUNT(*) > 3
    -> GROUP BY name;
ERROR 1111 at line 1: Invalid use of group function
```

Дело в том, что WHERE указывает на исходные ограничения, определяющие, какие строки следует выбирать, но значение функции COUNT() может быть получено лишь после того, как строки выбраны. Необходимо поместить выражение COUNT() в инструкцию HAVING, аналогичную WHERE, но применяемую к групповым характеристикам, а не отдельным записям. То есть HAVING работает с уже выбранным и сгруппированным набором строк, применяя к нему дополнительные условия, сформированные на основе результатов агрегирующей функции, которые не были известны в момент первичной выборки. Предыдущий запрос можно переписать так:

```
mysql> SELECT COUNT(*), name
    -> FROM driver_log
    -> GROUP BY name
    -> HAVING COUNT(*) > 3;
+----------+-------+
| COUNT(*) | name  |
+----------+-------+
|        5 | Henry |
+----------+-------+
```

Если вы используете инструкцию HAVING, то можете включить в запрос и инструкцию WHERE, но только для выбора строк, не для проверки итоговых значений.

Инструкция HAVING может ссылаться на псевдонимы:

```
mysql> SELECT COUNT(*) AS count, name
    -> FROM driver_log
    -> GROUP BY name
    -> HAVING count > 3;
+-------+-------+
| count | name  |
+-------+-------+
|     5 | Henry |
+-------+-------+
```

7.10. Устанавливаем уникальность значения

Задача

Вы хотите знать, уникальны ли значения таблицы.

Решение

Используйте инструкцию HAVING в сочетании с функцией COUNT().

Обсуждение

Вы можете использовать инструкцию HAVING для нахождения уникальных значений в тех случаях, где ключевое слово DISTINCT неприменимо. DISTINCT удаляет дубликаты, но не сообщает о том, какие значения повторялись в исходном множестве. HAVING же может указать, какие значения уникальны, а какие – нет.

Следующий запрос выводит дни, в которые работал только один водитель, и дни, в которые работало несколько водителей. HAVING и COUNT() используются для определения того, являются ли значения trav_date уникальными:

```
mysql> SELECT trav_date, COUNT(trav_date)
    -> FROM driver_log
    -> GROUP BY trav_date
    -> HAVING COUNT(trav_date) = 1;
+------------+------------------+
| trav_date  | COUNT(trav_date) |
+------------+------------------+
| 2001-11-26 |                1 |
| 2001-11-27 |                1 |
| 2001-12-01 |                1 |
+------------+------------------+
mysql> SELECT trav_date, COUNT(trav_date)
    -> FROM driver_log
    -> GROUP BY trav_date
    -> HAVING COUNT(trav_date) > 1;
+------------+------------------+
| trav_date  | COUNT(trav_date) |
```

```
+------------+------------------+
| 2001-11-29 |                3 |
| 2001-11-30 |                2 |
| 2001-12-02 |                2 |
+------------+------------------+
```

Такой прием можно применять и к комбинациям значений. Например, чтобы найти количество пар отправитель-получатель (`srcuser-dstuser`), которые обменялись всего одним письмом, будем искать комбинации, встречающиеся в таблице `mail` только один раз:

```
mysql> SELECT srcuser, dstuser
    -> FROM mail
    -> GROUP BY srcuser, dstuser
    -> HAVING COUNT(*) = 1;
+---------+---------+
| srcuser | dstuser |
+---------+---------+
| barb    | barb    |
| gene    | tricia  |
| phil    | barb    |
| tricia  | gene    |
| tricia  | phil    |
+---------+---------+
```

Обратите внимание, что запрос не выводит счетчик, как это делали предыдущие запросы с целью показать правильность подсчета. Вы можете использовать счетчик в инструкции `HAVING`, не включая его в список столбцов вывода.

7.11. Группирование по результатам выражения

Задача

Вы хотите объединить строки в подгруппу на основе значений, полученных в результате вычисления выражения.

Решение

Поместите выражение в инструкцию `GROUP BY`. В ранних версиях MySQL, не поддерживающих выражения в `GROUP BY`, используйте обходной маневр.

Обсуждение

Как и `ORDER BY`, инструкция `GROUP BY` может ссылаться на выражения начиная с версии MySQL 3.23.2. То есть можно использовать вычисления как основу для группирования. Например, чтобы вывести распределение длин названий штатов, выполните группирование по `LENGTH(name)`:

```
mysql> SELECT LENGTH(name), COUNT(*)
    -> FROM states GROUP BY LENGTH(name);
```

```
+--------------+----------+
| LENGTH(name) | COUNT(*) |
+--------------+----------+
|            4 |        3 |
|            5 |        3 |
|            6 |        5 |
|            7 |        8 |
|            8 |       12 |
|            9 |        4 |
|           10 |        4 |
|           11 |        2 |
|           12 |        4 |
|           13 |        3 |
|           14 |        2 |
+--------------+----------+
```

В версиях до MySQL 3.23.2 выражения в инструкции GROUP BY не поддерживались, так что такой запрос не выполнился бы. В рецепте 6.3 было показано, как обойти это ограничение для ORDER BY; то же самое можно сделать и для GROUP BY. Можно указать псевдоним выражения в списке столбцов вывода и сослаться в инструкции GROUP BY на этот псевдоним:

```
mysql> SELECT LENGTH(name) AS len, COUNT(*)
    -> FROM states GROUP BY len;
+------+----------+
| len  | COUNT(*) |
+------+----------+
|    4 |        3 |
|    5 |        3 |
|    6 |        5 |
|    7 |        8 |
|    8 |       12 |
|    9 |        4 |
|   10 |        4 |
|   11 |        2 |
|   12 |        4 |
|   13 |        3 |
|   14 |        2 |
+------+----------+
```

Можно переписать инструкцию GROUP BY так, чтобы она ссылалась на столбец по его позиции в списке вывода:

```
mysql> SELECT LENGTH(name), COUNT(*)
    -> FROM states GROUP BY 1;
+--------------+----------+
| LENGTH(name) | COUNT(*) |
+--------------+----------+
|            4 |        3 |
|            5 |        3 |
|            6 |        5 |
|            7 |        8 |
```

```
|            8 |       12 |
|            9 |        4 |
|           10 |        4 |
|           11 |        2 |
|           12 |        4 |
|           13 |        3 |
|           14 |        2 |
+--------------+----------+
```

Естественно, способы, предложенные в качестве альтернативы использованию выражений, будут работать и в MySQL версии 3.23.2 и выше, и некоторые пользователи считают их наиболее удачными.

При желании вы можете выполнять группирование по нескольким выражениям. Чтобы найти те дни года, в которые в Союз вступило более одного штата, группируйте строки по месяцу и дню статуса штата, а затем примените функции `HAVING` и `COUNT()` для поиска неуникальных комбинаций:

```
mysql> SELECT MONTHNAME(statehood), DAYOFMONTH(statehood), COUNT(*)
    -> FROM states GROUP BY 1, 2 HAVING COUNT(*) > 1;
+----------------------+-----------------------+----------+
| MONTHNAME(statehood) | DAYOFMONTH(statehood) | COUNT(*) |
+----------------------+-----------------------+----------+
| February             |                    14 |        2 |
| June                 |                     1 |        2 |
| March                |                     1 |        2 |
| May                  |                    29 |        2 |
| November             |                     2 |        2 |
+----------------------+-----------------------+----------+
```

7.12. Классификация некатегориальных данных

Задача

Вам необходимо получить итоговую информацию для набора значений, большая часть которых уникальна и не разбивается естественным образом на категории.

Решение

Для разбиения значений на категории используйте выражения.

Обсуждение

Группирование строк по результатам вычисления выражения часто используется для классификации значений, которые сами по себе не образуют никаких категорий. Такая возможность очень важна, так как инструкция `GROUP BY` отлично работает для столбцов с повторяющими значениями. Но, например, вам может понадобиться выполнить анализ численности населения, группируя записи таблицы `states` по значениям столбца `pop`. Получится

не очень хорошо, так как в столбце много разных значений. На самом деле они вообще *все* разные, как видно из запроса:

```
mysql> SELECT COUNT(pop), COUNT(DISTINCT pop) FROM states;
+------------+---------------------+
| COUNT(pop) | COUNT(DISTINCT pop) |
+------------+---------------------+
|         50 |                  50 |
+------------+---------------------+
```

В подобных ситуациях, когда не удается сгруппировать значения в небольшое количество подмножеств, можно использовать преобразование, которое вызовет разбиение на категории. Сначала определим диапазон значений численности населения:

```
mysql> SELECT MIN(pop), MAX(pop) FROM states;
+----------+----------+
| MIN(pop) | MAX(pop) |
+----------+----------+
|   453588 | 29760021 |
+----------+----------+
```

Из результата видно, что если разделить значения `pop` на пять миллионов, можно получить шесть категорий – вполне разумное количество. (Категории будут иметь диапазоны от 1 до 5 000 000; от 5 000 001 до 10 000 000; и т. д.) Чтобы определить категорию каждого значения, выполним деление на пять миллионов и воспользуемся целой частью результата:

```
mysql> SELECT FLOOR(pop/5000000) AS 'population (millions)',
    -> COUNT(*) AS 'number of states'
    -> FROM states GROUP BY 1;
+----------------------+------------------+
| population (millions) | number of states |
+----------------------+------------------+
|                    0 |               35 |
|                    1 |                8 |
|                    2 |                4 |
|                    3 |                2 |
|                    5 |                1 |
+----------------------+------------------+
```

Что-то не так... Выражение группирует значения численности населения в небольшое количество категорий, но почему-то неправильно определяет значение соответствующей категории. Давайте попробуем умножить результаты функции `FLOOR()` на пять:

```
mysql> SELECT FLOOR(pop/5000000)*5 AS 'population (millions)',
    -> COUNT(*) AS 'number of states'
    -> FROM states GROUP BY 1;
+----------------------+------------------+
| population (millions) | number of states |
+----------------------+------------------+
|                    0 |               35 |
```

```
|                    5 |                8 |
|                   10 |                4 |
|                   15 |                2 |
|                   25 |                1 |
+----------------------+------------------+
```

Все равно неправильно! Максимальное население штата равнялось 29 760 021, такое значение должно было попасть в категорию для 30, а не 25 миллионов. Проблема в том, что выражение, формирующее категории, группирует значения по нижней границе каждого интервала. Чтобы добиться группирования по верхней границе, применим один прием: для сопоставления значению *x* соответствующей категории длины *n* воспользуйтесь выражением:

```
FLOOR((x+(n-1))/n)
```

В итоге запрос будет таким:

```
mysql> SELECT FLOOR((pop+4999999)/5000000)*5 AS 'population (millions)',
    -> COUNT(*) AS 'number of states'
    -> FROM states GROUP BY 1;
+----------------------+------------------+
| population (millions) | number of states |
+----------------------+------------------+
|                    5 |               35 |
|                   10 |                8 |
|                   15 |                4 |
|                   20 |                2 |
|                   30 |                1 |
+----------------------+------------------+
```

Как видите, население большей части штатов США не превышает пяти миллионов.

Данный метод можно применять к любым видам числовых значений. Например, можно группировать записи таблицы `mail` в категории по 100 000 каждая:

```
mysql> SELECT FLOOR((size+99999)/100000) AS 'size (100KB)',
    -> COUNT(*) AS 'number of messages'
    -> FROM mail GROUP BY 1;
+--------------+--------------------+
| size (100KB) | number of messages |
+--------------+--------------------+
|            1 |                 13 |
|            2 |                  1 |
|           10 |                  1 |
|           24 |                  1 |
+--------------+--------------------+
```

В некоторых ситуациях удобнее создавать группы, используя логарифмическую шкалу. Например, численность населения штатов можно было интерпретировать так:

```
mysql> SELECT FLOOR(LOG10(pop)) AS 'log10(population)',
    -> COUNT(*) AS 'number of states'
```

```
    -> FROM states GROUP BY 1;
+------------------+------------------+
| log10(population) | number of states |
+------------------+------------------+
|                5 |                7 |
|                6 |               36 |
|                7 |                7 |
+------------------+------------------+
```

Насколько повторяющимся является множество значений?

Чтобы оценить, насколько много в множестве повторяющихся значений, используйте отношение `COUNT(DISTINCT)` к `COUNT()`. Если все значения уникальны, оба счетчика будет одинаковыми и отношение будет равно 1. Именно так дело обстоит со значениями `t` таблицы `mail` и значениями `pop` таблицы `states`:

```
mysql> SELECT COUNT(DISTINCT t) / COUNT(t) FROM mail;
+------------------------------+
| COUNT(DISTINCT t) / COUNT(t) |
+------------------------------+
|                         1.00 |
+------------------------------+
mysql> SELECT COUNT(DISTINCT pop) / COUNT(pop) FROM states;
+----------------------------------+
| COUNT(DISTINCT pop) / COUNT(pop) |
+----------------------------------+
|                             1.00 |
+----------------------------------+
```

В множествах с повторяющимися значениями `COUNT(DISTINCT)` будет меньше `COUNT()`, и отношение будет меньше единицы:

```
mysql> SELECT COUNT(DISTINCT name) / COUNT(name) FROM driver_log;
+-----------------------------------+
| COUNT(DISTINCT name) / COUNT(name) |
+-----------------------------------+
|                              0.30 |
+-----------------------------------+
```

Какова практическая польза этого отношения? Близкий к нулю результат показывает, что значения естественным образом делятся на небольшое количество групп. Единица же или результат, близкий к ней, показывает, что в множестве много уникальных значений, так что не будет особой пользы от применения инструкции `GROUP BY`. (То есть будет получено слишком большое количество категорий, сопоставимое с количеством значений.) Тогда для получения итоговой информации вам, вероятно, понадобится создать искусственные категории значений, используя приемы, предложенные в данном разделе.

7.13. Управление порядком вывода итоговой информации

Задача

Вы хотите упорядочить результат запроса, выводящего итоговую информацию.

Решение

Если желаемый порядок недостижим при помощи инструкции `GROUP BY`, используйте инструкцию `ORDER BY`.

Обсуждение

В MySQL `GROUP BY` не только группирует, но и сортирует. Поэтому часто нет необходимости дополнительно использовать инструкцию `ORDER BY` в запросе, выводящем итоговую информацию. Если же вы хотите изменить порядок, задаваемый инструкцией `GROUP BY` по умолчанию, используйте `ORDER BY`. Например, чтобы вычислить количество дней за рулем и общее количество миль для каждого водителя из таблицы `driver_log`, выполним такой запрос:

```
mysql> SELECT name, COUNT(*) AS days, SUM(miles) AS mileage
    -> FROM driver_log GROUP BY name;
+-------+------+-------------+
| name  | days | total miles |
+-------+------+-------------+
| Ben   |    3 |         362 |
| Henry |    5 |         911 |
| Suzi  |    2 |         893 |
+-------+------+-------------+
```

Результат упорядочен по именам. Если же вы хотите выводить водителей в порядке убывания проделанных миль или количества рабочих дней, добавьте в запрос соответствующую инструкцию `ORDER BY`:

```
mysql> SELECT name, COUNT(*) AS days, SUM(miles) AS mileage
    -> FROM driver_log GROUP BY name
    -> ORDER BY days DESC;
+-------+------+---------+
| name  | days | mileage |
+-------+------+---------+
| Henry |    5 |     911 |
| Ben   |    3 |     362 |
| Suzi  |    2 |     893 |
+-------+------+---------+
mysql> SELECT name, COUNT(*) AS days, SUM(miles) AS mileage
    -> FROM driver_log GROUP BY name
    -> ORDER BY mileage DESC;
```

```
+-------+------+---------+
| name  | days | mileage |
+-------+------+---------+
| Henry |    5 |     911 |
| Suzi  |    2 |     893 |
| Ben   |    3 |     362 |
+-------+------+---------+
```

Для ссылки в инструкции ORDER BY на итоговое значение необходимо использовать псевдоним или позицию столбца в списке вывода. Это касается даже версий MySQL 3.23.2 и выше, которые обычно разрешают применять выражения в инструкции ORDER BY; но эти выражения должны ссылаться на индивидуальные значения, а не на вычисленные.

Иногда можно изменить порядок вывода итоговой информации, не прибегая к инструкции ORDER BY, за счет выбора подходящего выражения для инструкции GROUP BY. Например, если вы хотите узнать, сколько штатов вступило в Союз в каждый из дней недели, и группируете результаты по названию дня недели, то они будут выведены в лексическом порядке:

```
mysql> SELECT DAYNAME(statehood), COUNT(*) FROM states
    -> GROUP BY DAYNAME(statehood);
+--------------------+----------+
| DAYNAME(statehood) | COUNT(*) |
+--------------------+----------+
| Friday             |        8 |
| Monday             |        9 |
| Saturday           |       11 |
| Thursday           |        5 |
| Tuesday            |        6 |
| Wednesday          |       11 |
+--------------------+----------+
```

Из списка, конечно, можно извлечь информацию о том, что ни один из штатов не вступил в Союз в воскресенье, но для этого требуется провести некоторый анализ результата. Вывод был бы более понятным, если бы применялось упорядочивание по дням недели. Можно добавить инструкцию ORDER BY для сортировки по числовому значению дня недели, но есть и другой способ достижения того же результата, причем без помощи ORDER BY: выполним группировку по DAYOFWEEK(), а не по DAYNAME():

```
mysql> SELECT DAYNAME(statehood), COUNT(*)
    -> FROM states GROUP BY DAYOFWEEK(statehood);
+--------------------+----------+
| DAYNAME(statehood) | COUNT(*) |
+--------------------+----------+
| Monday             |        9 |
| Tuesday            |        6 |
| Wednesday          |       11 |
| Thursday           |        5 |
| Friday             |        8 |
| Saturday           |       11 |
+--------------------+----------+
```

Инструкцию GROUP BY нельзя использовать для сортировки вывода в других СУБД. Чтобы повысить вероятность того, что для других баз данных запросы MySQL не придется переписывать, добавляйте во всех случаях явную инструкцию ORDER BY.

7.14. Нахождение наибольшего и наименьшего из итоговых значений

Задача

Вы хотите вычислить итоговые значения для групп и вывести только наибольшее или наименьшее из них.

Решение

Добавьте в запрос инструкцию LIMIT.

Обсуждение

Функции MIN() и MAX() находят граничные значения диапазона, но в данном случае нам нужны экстремумы множества итоговых значений, и эти функции уже не годятся. Аргументами MIN() и MAX() не могут быть другие агрегирующие функции. Например, вы без труда можете вычислить общее количество миль, проделанных каждым водителем:

```
mysql> SELECT name, SUM(miles)
    -> FROM driver_log
    -> GROUP BY name;
+-------+------------+
| name  | SUM(miles) |
+-------+------------+
| Ben   |        362 |
| Henry |        911 |
| Suzi  |        893 |
+-------+------------+
```

Но выбрать только запись для водителя с наибольшим количеством миль так не удастся:

```
mysql> SELECT name, SUM(miles)
    -> FROM driver_log
    -> GROUP BY name
    -> HAVING SUM(miles) = MAX(SUM(name));
ERROR 1111 at line 1: Invalid use of group function
```

Вместо этого упорядочим строки, сделав первым наибольшее значение SUM(), и применим инструкцию LIMIT для выбора первой записи:

```
mysql> SELECT name, SUM(miles) AS 'total miles'
    -> FROM driver_log
    -> GROUP BY name
    -> ORDER BY 'total miles' DESC LIMIT 1;
```

```
+-------+-------------+
| name  | total miles |
+-------+-------------+
| Henry |         911 |
+-------+-------------+
```

В инструкции ORDER BY использован псевдоним, так как она не может ссылаться непосредственно на агрегирующие функции (см. рецепт 7.13).

Обратите внимание, что если существует несколько строк с одинаковым наибольшим итоговым значением, рассмотренный запрос не сообщит вам об этом. Например, можно попробовать установить, какая буква чаще всего является первой в названии штата:

```
mysql> SELECT LEFT(name,1) AS letter, COUNT(*) AS count FROM states
    -> GROUP BY letter ORDER BY count DESC LIMIT 1;
+--------+-------+
| letter | count |
+--------+-------+
| M      |     8 |
+--------+-------+
```

Но и с буквы N начинаются названия восьми штатов. Если нужны все наиболее часто встречающиеся значения, которых может быть несколько, следует использовать два запроса:

```
mysql> SELECT LEFT(name,1) AS letter, @max:=COUNT(*) AS count FROM states
    -> GROUP BY letter ORDER BY count DESC LIMIT 1;
mysql> SELECT LEFT(name,1) AS letter, COUNT(*) AS count FROM states
    -> GROUP BY letter HAVING count = @max;
+--------+-------+
| letter | count |
+--------+-------+
| M      |     8 |
| N      |     8 |
+--------+-------+
```

7.15. Итоги по датам

Задача

Вы хотите выводить итоговую информацию для значений даты или времени.

Решение

Используйте инструкцию GROUP BY для разбиения значений времени на диапазоны нужного размера. Часто для извлечения значащих составляющих даты или времени требуется использовать выражения.

Обсуждение

Чтобы разместить записи во временном порядке, используйте инструкцию ORDER BY для сортировки столбца временного типа. Если вместо этого вы хотите

выводить итоговую информацию для записей, группируя их по интервалам времени, необходимо определить, как сопоставить запись интервалу, и использовать для группирования записей инструкцию GROUP BY.

Если временные значения разбиваются на категории естественным образом, можно работать с ними напрямую. Часто так обрабатываются таблицы, в которых составляющие даты или времени представлены в отдельных столбцах. Например, таблица лучших игроков *baseball1.com* приводит даты рождения, используя разные столбцы для года, месяца и дня. Чтобы узнать, сколько игроков родилось в каждый день года, подсчитайте записи по календарной дате, используя значения месяца и дня, но игнорируя год:

```
mysql> SELECT birthmonth, birthday, COUNT(*)
    -> FROM master
    -> WHERE birthmonth IS NOT NULL AND birthday IS NOT NULL
    -> GROUP BY birthmonth, birthday;
+------------+----------+----------+
| birthmonth | birthday | COUNT(*) |
+------------+----------+----------+
|          1 |        1 |       47 |
|          1 |        2 |       40 |
|          1 |        3 |       50 |
|          1 |        4 |       38 |
...
|         12 |       28 |       33 |
|         12 |       29 |       32 |
|         12 |       30 |       32 |
|         12 |       31 |       27 |
+------------+----------+----------+
```

Менее подробные итоги могут быть получены за счет использования только значений месяца:

```
mysql> SELECT birthmonth, COUNT(*)
    -> FROM master
    -> WHERE birthmonth IS NOT NULL
    -> GROUP BY birthmonth;
+------------+----------+
| birthmonth | COUNT(*) |
+------------+----------+
|          1 |     1311 |
|          2 |     1144 |
|          3 |     1243 |
|          4 |     1179 |
|          5 |     1118 |
|          6 |     1105 |
|          7 |     1244 |
|          8 |     1438 |
|          9 |     1314 |
|         10 |     1438 |
|         11 |     1314 |
|         12 |     1269 |
+------------+----------+
```

Иногда даже не выделенные в отдельный столбец временные значения можно использовать напрямую. Чтобы определить, сколько водителей было в дороге каждый день и сколько миль проехал каждый из них, сгруппируйте записи `driver_log` по дате:

```
mysql> SELECT trav_date,
    -> COUNT(*) AS 'number of drivers', SUM(miles) As 'miles logged'
    -> FROM driver_log GROUP BY trav_date;
+------------+------------------+--------------+
| trav_date  | number of drivers | miles logged |
+------------+------------------+--------------+
| 2001-11-26 |                1 |          115 |
| 2001-11-27 |                1 |           96 |
| 2001-11-29 |                3 |          822 |
| 2001-11-30 |                2 |          355 |
| 2001-12-01 |                1 |          197 |
| 2001-12-02 |                2 |          581 |
+------------+------------------+--------------+
```

Однако при увеличении количества записей в таблице результат такого запроса будет становиться все длиннее и длиннее. В некоторый момент различных дат станет так много, что в подобном суммировании уже не будет смысла, и, вероятно, вы решите выводить итоги не по дням, а по неделям или месяцам.

Если временной столбец содержит слишком много различных значений для того, чтобы их можно было группировать естественным образом, итоги обычно группируют при помощи выражений, которые отображают соответствующие составляющие даты или времени на небольшое количество категорий. Например, чтобы вывести итоги по времени дня для таблицы `mail`, сделайте следующее:[1]

```
mysql> SELECT HOUR(t) AS hour,
    -> COUNT(*) AS 'number of messages',
    -> SUM(size) AS 'number of bytes sent'
    -> FROM mail
    -> GROUP BY hour;
+------+--------------------+----------------------+
| hour | number of messages | number of bytes sent |
+------+--------------------+----------------------+
|    7 |                  1 |                 3824 |
|    8 |                  1 |                  978 |
|    9 |                  2 |                 2904 |
|   10 |                  2 |              1056806 |
|   11 |                  1 |                 5781 |
|   12 |                  2 |               195798 |
```

[1] Обратите внимание на то, что результат содержит записи только для тех часов, которые реально представлены данными таблицы. Чтобы сформировать итоги, в которых будет запись для каждого часа, используйте соединение (join) для заполнения «недостающих» значений (см. рецепт 12.9).

```
|   13 |                 1 |                 271 |
|   14 |                 1 |               98151 |
|   15 |                 1 |                1048 |
|   17 |                 2 |             2398338 |
|   22 |                 1 |               23992 |
|   23 |                 1 |               10294 |
+------+-------------------+---------------------+
```

Чтобы вывести итоги по дню недели, используйте функцию DAYOFWEEK():

```
mysql> SELECT DAYOFWEEK(t) AS weekday,
    -> COUNT(*) AS 'number of messages',
    -> SUM(size) AS 'number of bytes sent'
    -> FROM mail
    -> GROUP BY weekday;
+---------+--------------------+---------------------+
| weekday | number of messages | number of bytes sent |
+---------+--------------------+---------------------+
|       1 |                  1 |                 271 |
|       2 |                  4 |             2500705 |
|       3 |                  4 |             1007190 |
|       4 |                  2 |               10907 |
|       5 |                  1 |                 873 |
|       6 |                  1 |               58274 |
|       7 |                  3 |              219965 |
+---------+--------------------+---------------------+
```

Чтобы сделать результат более понятным, можно использовать функцию DAYNAME() для вывода не номеров дней недели, а их названий. Однако названия дней недели сортируются в лексическом порядке (например, «Tuesday» будет стоять после «Friday»), так что применяйте функцию DAYNAME() только для отображения результатов. Продолжайте группировать по числовым значениям дней, чтобы строки вывода сортировались в таком порядке:

```
mysql> SELECT DAYNAME(t) AS weekday,
    -> COUNT(*) AS 'number of messages',
    -> SUM(size) AS 'number of bytes sent'
    -> FROM mail
    -> GROUP BY DAYOFWEEK(t);
+-----------+--------------------+---------------------+
| weekday   | number of messages | number of bytes sent |
+-----------+--------------------+---------------------+
| Sunday    |                  1 |                 271 |
| Monday    |                  4 |             2500705 |
| Tuesday   |                  4 |             1007190 |
| Wednesday |                  2 |               10907 |
| Thursday  |                  1 |                 873 |
| Friday    |                  1 |               58274 |
| Saturday  |                  3 |              219965 |
+-----------+--------------------+---------------------+
```

Аналогично можно суммировать значения по месяцам года, сортируя по числовому значению месяца и выводя его название.

Есть множество возможностей ввода категорий для временных значений:

- Столбцы DATETIME и TIMESTAMP обычно содержат много уникальных значений. Для получения итогов за день отбросьте составляющую времени дня, чтобы привести все значения одного дня к единому значению. Любая из предложенных инструкций GROUP BY выполнит такую операцию, но последняя будет наиболее медленной:

```
GROUP BY FROM_DAYS(TO_DAYS(имя_столбца))
GROUP BY YEAR(имя_столбца), MONTH(имя_столбца), DAYOFMONTH(имя_столбца)
GROUP BY DATE_FORMAT(имя_столбца,'%Y-%m-%e')
```

- Чтобы вывести месячный или квартальный отчет о продажах, группируйте по MONTH(*имя_столбца*) или QUARTER(*имя_столбца*) для помещения дат в соответствующую часть года.
- Чтобы вывести итоговые данные об активности веб-сервера, поместите журналы сервера в MySQL и запускайте запросы, которые будут распределять записи по категориям. В главе 18 рассказано о том, как сделать это для Apache.

7.16. Одновременная работа с итогами по группам и общим итогом

Задача

Вы хотите вывести отчет, который требует нескольких уровней итоговой информации. Или хотите сравнить итоговые значения по группам с общим итоговым значением.

Решение

Выполните два запроса, извлекающие различные уровни итоговой информации. Или используйте для выполнения некоторых действий язык программирования, чтобы можно было обойтись одним запросом.

Обсуждение

Иногда в отчете необходимо несколько уровней итоговой информации. Например, выведем общее количество миль для каждого водителя из таблицы driver_log, а также отношение этого количества для каждого водителя к общему количеству миль по всем водителям:

```
+-------+--------------+------------------------+
| name  | miles/driver | percent of total miles |
+-------+--------------+------------------------+
| Ben   |          362 |        16.712834718375 |
| Henry |          911 |        42.059095106187 |
| Suzi  |          893 |        41.228070175439 |
+-------+--------------+------------------------+
```

Для вычисления процентного вклада каждого водителя в общее количество проделанных миль необходимы итоги для каждого водителя, а также общий итог. Для формирования отчета в SQL необходимы два запроса, так как невозможно получить оба вида итогов в одном запросе.[1] Сначала выполним запрос, вычисляющий общее количество миль по всей таблице:

```
mysql> SELECT @total := SUM(miles) AS 'total miles' FROM driver_log;
+-------------+
| total miles |
+-------------+
|        2166 |
+-------------+
```

Затем вычислим значения итогов по группам и используем общий итог для получения отношения:

```
mysql> SELECT name,
    -> SUM(miles) AS 'miles/driver',
    -> (SUM(miles)*100)/@total AS 'percent of total miles'
    -> FROM driver_log GROUP BY name;
+-------+--------------+------------------------+
| name  | miles/driver | percent of total miles |
+-------+--------------+------------------------+
| Ben   |          362 |        16.712834718375 |
| Henry |          911 |        42.059095106187 |
| Suzi  |          893 |        41.228070175439 |
+-------+--------------+------------------------+
```

Можно предложить и другое решение, не использующее переменную, – извлечем общий итог в отдельную таблицу, затем соединим ее с исходной:

```
mysql> CREATE TEMPORARY TABLE t
    -> SELECT SUM(miles) AS total FROM driver_log;
mysql> SELECT driver_log.name,
    -> SUM(driver_log.miles) AS 'miles/driver',
    -> (SUM(driver_log.miles)*100)/t.total AS 'percent of total miles'
    -> FROM driver_log, t GROUP BY driver_log.name;
+-------+--------------+------------------------+
| name  | miles/driver | percent of total miles |
+-------+--------------+------------------------+
| Ben   |          362 |                  16.71 |
| Henry |          911 |                  42.06 |
| Suzi  |          893 |                  41.23 |
+-------+--------------+------------------------+
```

Если вы формируете отчет в программе, то можете выполнить некоторые арифметические операции при помощи языка программирования, избавившись от второго запроса. Рассмотрим пример для Python:

---

[1] Ну, вообще-то это не совсем правда... используя подзапрос, вы можете получить итоги в одном запросе. Но MySQL будет поддерживать подзапросы только начиная с версии 4.1.

```
# выдаем запрос для вычисления итогов по водителям
cursor = conn.cursor ()
cursor.execute ("SELECT name, SUM(miles) FROM driver_log GROUP BY name")
rows = cursor.fetchall ()
cursor.close ()

# осуществляем один проход для вычисления общего количества миль
total = 0
for (name, miles) in rows:
    total = total + miles

# повторяем снова для вывода отчета
print "name      miles/driver   percent of total miles"
for (name, miles) in rows:
    print "%-8s         %5d                %f" \
            % (name, miles, (100*miles)/total)
```

Еще одним типом задач, требующих нескольких уровней итоговой информации, являются задачи сравнения итоговых значений по группам с соответствующим общим итогом. Предположим, вы хотите узнать, кто из водителей в среднем за день проехал меньше миль, чем общее среднее значение для группы. Если использовать только SQL, задачу не удастся решить средствами одного запроса, но легко можно решить при помощи двух. Сначала вычислим общее среднее значение и сохраним его в переменной:

```
mysql> SELECT @overall_avg := AVG(miles) FROM driver_log;
+----------------------------+
| @overall_avg := AVG(miles) |
+----------------------------+
|                   216.6000 |
+----------------------------+
```

Затем будем сравнивать среднее значение для каждого водителя с сохраненным общим средним, используя инструкцию `HAVING`:

```
mysql> SELECT name, AVG(miles) AS driver_avg FROM driver_log
    -> GROUP BY name
    -> HAVING driver_avg < @overall_avg;
+-------+------------+
| name  | driver_avg |
+-------+------------+
| Ben   |   120.6667 |
| Henry |   182.2000 |
+-------+------------+
```

Как и при выводе отчетов, использующих несколько уровней итогов, можно решить задачу в одном запросе, если переписать программу, заставив поработать ваш язык программирования:

1. Выдайте запрос на извлечение итоговой информации по группе.
2. Вычислите в цикле общий итог.
3. Выполните еще один проход по результирующему множеству, сравнивая каждое групповое итоговое значение с общим и выводя только те записи, для которых сравнение верно.

7.17. Формирование отчета, содержащего итоговую информацию и список

Задача

Вы хотите написать запрос, который выводит итоговую информацию, а также список записей, сопоставленных каждому итоговому значению.

Решение

Осознайте, что речь идет об одном из вариантов работы с несколькими уровнями итоговой информации, и решайте задачу, используя описанные ранее приемы.

Обсуждение

Предположим, вы хотите получить отчет, который выглядел бы так:

```
Name: Ben; days on road: 3; miles driven: 362
  date: 2001-11-29, trip length: 131
  date: 2001-11-30, trip length: 152
  date: 2001-12-02, trip length: 79
Name: Henry; days on road: 5; miles driven: 911
  date: 2001-11-26, trip length: 115
  date: 2001-11-27, trip length: 96
  date: 2001-11-29, trip length: 300
  date: 2001-11-30, trip length: 203
  date: 2001-12-01, trip length: 197
Name: Suzi; days on road: 2; miles driven: 893
  date: 2001-11-29, trip length: 391
  date: 2001-12-02, trip length: 502
```

Для каждого водителя из таблицы `driver_log` отчет отображает следующую информацию:

- Итоговую строку, в которой приведены имя водителя, количество дней, проведенных в дороге, и количество проделанных миль.
- Список дат и расстояний в милях для отдельных поездок, из которых и получены итоговые значения.

Мы имеем дело с вариацией задачи «различные уровни итоговой информации», которая обсуждалась в предыдущем рецепте. Сначала это может показаться неочевидным, так как одним из типов выводимой информации является список, а не итог. Но фактически это итог «нулевого уровня». Подобные задачи могут встречаться и в других формах:

- У вас есть база данных, в которой перечислены взносы в пользу кандидатов от вашей политической партии. Руководитель партии просит вывести для каждого кандидата количество сделанных в его пользу взносов и их общую сумму, а также список спонсоров с адресами.

- Вы хотите подготовить для презентации компании пресс-релиз, в котором приводились бы объемы продаж по регионам со списком объемов продаж в каждом штате региона.

В каждом из случаев вы можете использовать приемы, описанные в предыдущем рецепте:

- Выполните отдельные запросы для получения информации на каждом уровне. (Так же как один запрос не может одновременно сформировать итоговые значения для групп и общий итог, один запрос не может вывести итоговые значения для групп и список отдельных записей каждой группы.)
- Выбирайте строки для составления списков и выполняйте итоговые вычисления самостоятельно, чтобы избавиться от запроса, формирующего итоги.

Давайте опробуем каждый из способов для вывода отчета о водителях, приведенного в начале раздела. Сформируем (в Python) отчет, используя один запрос для суммирования дней и миль водителя, а второй – для извлечения записей об отдельных поездках каждого водителя:

```
# Выбираем общее количество миль для водителя и создаем словарь (dictionary),
# сопоставляющий каждому водителю количество дней в дороге и количество миль.
name_map = { }
cursor = conn.cursor ()
cursor.execute ("""
                    SELECT name, COUNT(name), SUM(miles)
                    FROM driver_log GROUP BY name
                """)
for (name, days, miles) in cursor.fetchall ():
    name_map[name] = (days, miles)

# Выбираем поездки каждого водителя и печатаем отчет,
# отображающий итоговую запись для каждого водителя перед списком его поездок.
cursor.execute ("""
                    SELECT name, trav_date, miles
                    FROM driver_log ORDER BY name, trav_date
                """)
cur_name = ""
for (name, trav_date, miles) in cursor.fetchall ():
    if cur_name != name:          # новый водитель; выводим итоги для этого водителя
        print "Name: %s; days on road: %d; miles driven: %d" \
                        % (name, name_map[name][0], name_map[name][1])
        cur_name = name
    print "  date: %s, trip length: %d" % (trav_date, miles)
cursor.close ()
```

При вычислении итогов в программе можно сократить количество необходимых запросов. Если вы проходите по списку поездок, получая самостоятельно мили и дни, то достаточно одного запроса:

```
# Получить список поездок с именами водителей
cursor = conn.cursor ()
```

```
cursor.execute ("""
                    SELECT name, trav_date, miles FROM driver_log
                    ORDER BY name, trav_date
                """)
rows = cursor.fetchall ()
cursor.close ()

# Проходим строки один раз, чтобы сформировать словарь, сопоставляющий каждому
# водителю количество дней в дороге и количество миль (записи словаря – это списки,
# а не кортежи, так как нам необходимы записи, которые могли бы изменяться в цикле).
name_map = { }
for (name, trav_date, miles) in rows:
    if not name_map.has_key (name):    # инициализировать запись, если не существует
        name_map[name] = [0, 0]
    name_map[name][0] = name_map[name][0] + 1       # посчитать дни
    name_map[name][1] = name_map[name][1] + miles   # сложить мили

# Проходим строки еще раз, чтобы напечатать отчет,
# отображающий итоговую запись каждого водителя перед списком поездок.
cur_name = ""
for (name, trav_date, miles) in rows:
    if cur_name != name:          # новый водитель; выводим итоги для этого водителя
        print "Name: %s; days on road: %d; miles driven: %d" \
                        % (name, name_map[name][0], name_map[name][1])
        cur_name = name
    print "  date: %s, trip length: %d" % (trav_date, miles)
```

Если вам требуется больше уровней итоговой информации, задача усложняется. Например, вы можете захотеть вывести перед данными предыдущего отчета строку с информацией об общем количестве миль, проделанных всеми водителями вместе:

```
Total miles driven by all drivers combined: 2166

Name: Ben; days on road: 3; miles driven: 362
  date: 2001-11-29, trip length: 131
  date: 2001-11-30, trip length: 152
  date: 2001-12-02, trip length: 79
Name: Henry; days on road: 5; miles driven: 911
  date: 2001-11-26, trip length: 115
  date: 2001-11-27, trip length: 96
  date: 2001-11-29, trip length: 300
  date: 2001-11-30, trip length: 203
  date: 2001-12-01, trip length: 197
Name: Suzi; days on road: 2; miles driven: 893
  date: 2001-11-29, trip length: 391
  date: 2001-12-02, trip length: 502
```

В этом случае необходимо использовать еще один запрос для получения общего итога или еще одно соответствующее вычисление в программе.

8

Изменение таблицы с помощью предложения ALTER TABLE

8.0. Введение

Однажды вы наверняка столкнетесь с необходимостью перепроектирования некоторых своих таблиц. Изменение специфики приложения может потребовать хранения информации, которая не была предусмотрена в исходном определении таблицы. Или может оказаться, что столбец `AUTO_INCREMENT` уже занял весь доступный ему диапазон для формирования новых номеров последовательности, и необходимо изменить столбец, придав ему более вместительный целый тип. MySQL обеспечивает множество способов изменения структуры таблицы. В главе будут описаны следующие типы операций:

Удаление, добавление и изменение положения столбца. Столбцы, ставшие ненужными и оказавшиеся избыточными, можно удалить, чтобы упростить таблицу и освободить пространство. Можно переместить столбцы из одной таблицы в другую в процессе нормализации. Если вам необходимо сохранить какую-то дополнительную информацию, можно добавить в таблицу новые столбцы.

Изменение определения или имени столбца. Если в своем исходном виде столбец не отвечает стоящим перед вами задачам, вы можете исправить положение, подкорректировав его. Например, можно преобразовать строковый столбец из чувствительного к регистру в нечувствительный или наоборот. Или же ваш столбец `AUTO_INCREMENT` может относиться к типу `TINYINT` и позволять генерировать только 127 порядковых номеров. Изменив тип столбца на целое без знака или указав более вместительный целый тип, вы сделаете возможным формирование большего количества значений последовательности. Если после перехода к новой версии MySQL окажется, что имя столбца совпадает с зарезервированным словом, следует изменить это имя. Или же вам просто захочется заменить такое имя столбца, как `num`, на что-то более содержательное, например `test_score`, чтобы сделать более очевидным его назначение.

Изменение типа таблицы. В MySQL разные типы таблиц имеют разные характеристики. Если вы решили, что некоторый тип таблицы будет удобнее для работы приложения, чем существующий, вы можете изменить его.

Переименование таблицы. Как и в случае со столбцом, переименуйте таблицу, если придумаете для нее более удачное имя. Или это можно сделать с целью чередования названий множества таблиц, используемых для логов.

Изменение структуры индекса таблицы. Редко применяемое удаление индекса может повысить производительность операций вставки и обновления строк таблицы, так как не придется обновлять индекс. Добавление индекса столбца, на который вы часто ссылаетесь в запросах, может повысить эффективность запросов SELECT. Кроме того, индексирование можно применять для удаления из таблицы повторяющихся значений.

Большая часть перечисленных операций выполняется посредством предложения ALTER TABLE, одного из наиболее мощных и не оцененных по достоинству предложений MySQL. Применяя ALTER TABLE, можно сделать очень много полезных вещей, но пользователи MySQL часто используют его возможности не в полной мере. Возможно, причина кроется в том, что у предложения ALTER TABLE так много опций, что его синтаксис может обескураживать. Надеюсь, что предлагаемая вашему вниманию глава снимет с этого предложения покров таинственности и убедит в его полезности.

Прежде чем переходить к последующим разделам, создайте таблицу, которая будет использоваться в примерах:

```
CREATE TABLE mytbl
(
    i   INT,
    c   CHAR(1)
);
```

После выполнения каждого из преобразований, описанных в рецептах, вы сможете увидеть произошедшие в структуре таблицы изменения, создав предложение SHOW COLUMNS FROM mytbl или SHOW CREATE TABLE mytbl. Например, после создания mytbl предложение SHOW COLUMNS выводит такие данные:

```
mysql> SHOW COLUMNS FROM mytbl;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| i     | int(11) | YES  |     | NULL    |       |
| c     | char(1) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
```

Обратите внимание на то, что MySQL автоматически присваивает значения по умолчанию и определяет, может ли столбец содержать значения NULL, несмотря на то, что в предложении CREATE TABLE эти атрибуты явно не указаны (см. рецепт 8.3).

8.1. Удаление, добавление и перемещение столбца

Задача

Вы хотите избавиться от столбца таблицы, добавить новый столбец или переместить столбец внутри таблицы.

Решение

Используйте инструкции DROP и ADD предложения ALTER TABLE для удаления и добавления столбца. Для перемещения столбца следует удалить его и повторно вставить туда, куда нужно.

Обсуждение

Чтобы удалить столбец из таблицы, укажите инструкцию DROP, затем имя столбца. Удалим из mytbl столбец i, оставив в таблице только столбец c:

```
ALTER TABLE mytbl DROP i;
```

DROP не работает, если вы пытаетесь удалить единственный оставшийся столбец таблицы. (Чтобы проверить утверждение, попробуйте удалить столбец c после того, как столбец i уже удален.)

Для добавления столбца в таблицу используйте инструкцию ADD и укажите определение столбца. Следующее предложение восстанавливает столбец i в таблице mytbl:

```
ALTER TABLE mytbl ADD i INT;
```

Теперь mytbl снова содержит те же два столбца, что и при первом создании таблицы, но ее структура изменилась. Дело в том, что новые столбцы по умолчанию добавляются в конец таблицы. Так что столбец i, который был первым, теперь превратится в последний:

```
mysql> SHOW COLUMNS FROM mytbl;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| c     | char(1) | YES  |     | NULL    |       |
| i     | int(11) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
```

Если вы хотите поместить столбец в определенную позицию в таблице, используйте спецификатор FIRST, чтобы сделать его первым столбцом, или AFTER *имя_столбца*, чтобы разместить его после столбца с именем *имя_столбца*. Опробуйте приведенные ниже предложения ALTER TABLE, используя после каждого из них предложение SHOW COLUMNS, чтобы посмотреть на то, что произошло:

```
ALTER TABLE mytbl DROP i;
ALTER TABLE mytbl ADD i INT FIRST;
```

Перемещение столбцов и столбцы TIMESTAMP

Соблюдайте осторожность при перемещении столбцов таблицы, содержащей несколько столбцов TIMESTAMP. Первый столбец TIMESTAMP имеет особые свойства, не присущие другим столбцам (различия описаны в рецепте 5.31). Изменив порядок столбцов TIMESTAMP, вы измените и манеру поведения таблицы.

```
ALTER TABLE mytbl DROP i;
ALTER TABLE mytbl ADD i INT AFTER c;
```

Спецификаторы FIRST и AFTER работают только с инструкцией ADD. То есть если вы хотите изменить положение существующего столбца, вам придется сначала удалить его (посредством DROP), а затем добавить в новое место.

8.2. Изменение определения или имени столбца

Задача

Вы хотите изменить определение столбца.

Решение

Используйте MODIFY или CHANGE: MODIFY проще, но не умеет изменять название столбца, а CHANGE несколько непонятнее, зато может изменить и имя, и определение.

Обсуждение

Чтобы изменить определение столбца, используйте предложение MODIFY или ключевое слово CHANGE.[1] MODIFY имеет более простой формат: укажите имя столбца, затем его новое определение. Например, изменим тип столбца с CHAR(1) на CHAR(10):

```
ALTER TABLE mytbl MODIFY c CHAR(10);
```

Синтаксис CHANGE несколько другой. После ключевого слова CHANGE указывается имя столбца, который следует изменить, затем его новое определение, *включающее* новое имя. Второе имя столбца необходимо, так как CHANGE позволяет изменить и имя столбца, а не только его определение. Например, чтобы изменить тип столбца i с INT на BIGINT и одновременно переименовать его в j, запишем предложение так:

```
ALTER TABLE mytbl CHANGE i j BIGINT;
```

[1] Для MODIFY требуется версия MySQL 3.22.16 или более поздние.

Если теперь попробовать выполнить CHANGE для преобразования j из BIGINT обратно в INT без изменения имени, предложение может показаться немного странным:

```
ALTER TABLE mytbl CHANGE j j INT;
```

На первый взгляд, предложение некорректно – имя столбца употребляется слишком много раз. На самом же деле предложение записано абсолютно корректно. Вам просто нужно привыкнуть к тому, что синтаксис CHANGE требует указания двух имен столбцов (даже если они совпадают друг с другом). Особенно важно помнить об этом тем, чья версия MySQL еще не допускает использования MODIFY. Любое предложение ALTER TABLE с конструкцией MODIFY *имя_столбца* может быть заменено на аналог, использующий CHANGE *имя_столбца имя_столбца*. То есть два следующих предложения эквивалентны:

```
ALTER TABLE имя_таблицы MODIFY имя_столбца ... ;
ALTER TABLE имя_таблицы CHANGE имя_столбца имя_столбца ... ;
```

Было бы очень удобно использовать такую форму предложения ALTER TABLE, которая бы только переименовывала столбец, не требуя повторного ввода его определения. Особенно такая возможность пригодилась бы при работе со столбцами ENUM и SET, содержащими много значений. Но, к сожалению, такого предложения не существует, что усложняет работу с упомянутыми столбцами при использовании предложения ALTER TABLE. Предположим, что вы добавляете в mytbl столбец e типа ENUM, имеющий несколько членов:

```
ALTER TABLE mytbl ADD e
    ENUM('hardware','software','books','office supplies',
                'telecommunications','furniture','utilities',
                'shipping','tax');
```

Если вы захотите изменить название столбца с e на e2, то для задания нового имени используете CHANGE. Но придется повторить и определение столбца:

```
ALTER TABLE mytbl CHANGE e e2
    ENUM('hardware','software','books','office supplies',
                'telecommunications','furniture','utilities',
                'shipping','tax');
```

Да... Не хотелось бы все это набирать. Ручной ввод корректного предложения ALTER TABLE для таблиц такого вида утомителен, не говоря уже о том, что подвержен ошибкам. Чтобы избежать повторного ввода определения, запишите текущее определение в файл и редактируйте его для получения соответствующего предложения ALTER TABLE:

- Запустите *mysqldump* для получения предложения CREATE TABLE, содержащего определение таблицы:

  ```
  % mysqldump --no-data cookbook mytbl > test.txt
  ```

  В файле *test.txt* должно присутствовать такое определение:

  ```
  CREATE TABLE mytbl (
    c char(10) default NULL,
  ```

```
  j bigint(20) NOT NULL default '100',
  e enum('hardware','software','books','office supplies','telecommunications',
'furniture','utilities','shipping','tax') default NULL
) TYPE=MyISAM;
```

Опция *--no-data* указывает *mysqldump* на то, что не нужно делать дамп данных таблицы, необходимо только получить определение таблицы.

- Отредактируйте файл *test.txt*, удалив из него все, кроме определения столбца `e`:

```
e enum('hardware','software','books','office supplies','telecommunications',
'furniture','utilities','shipping','tax') default NULL
```

- Измените определение так, чтобы получилось предложение `ALTER TABLE` с точкой с запятой в конце:

```
ALTER TABLE mytbl CHANGE e e2
  enum('hardware','software','books','office supplies','telecommunications',
'furniture','utilities','shipping','tax') default NULL;
```

- Сохраните файл *test.txt*, выйдите из редактора и передайте *test.txt* как командный файл в *mysql*:

```
% mysql cookbook < test.txt
```

Конечно, для простых столбцов легче ввести предложение `ALTER TABLE` вручную, а не выполнять всю эту процедуру. Но для столбцов `ENUM` и `SET` с длинными и неизящными определениями часто бывает разумнее использовать редактор для создания командного файла *mysql* из вывода *mysqldump*. Этот же прием можно применять для упрощения изменения порядка элементов столбцов `ENUM` и `SET` или для добавления и удаления членов из определения столбца. Другой подход к перемещению столбцов изложен в рецепте 9.8, где создается сценарий, максимально упрощающий добавление новых членов. Сценарий исследует структуру таблицы и использует эту информацию для получения нужного предложения `ALTER TABLE`, изменяющего столбец `ENUM` или `SET`.

8.3. Предложение ALTER TABLE, значения NULL и значения по умолчанию

Задача

Вы изменили определение столбца, но MySQL изменяет и такие его атрибуты, как значения по умолчанию и допустимость значений `NULL`, о чем вы совсем не просили.

Решение

Эти атрибуты являются частью определения. Если вы явно не указываете их, MySQL сама выбирает значения. Так что указывайте определение столбцов как можно более конкретно.

Обсуждение

Если вы изменяете столбец посредством MODIFY или CHANGE, то можете указать, допускает ли столбец значения NULL и каковы его значения по умолчанию. Если не сделать этого, то MySQL автоматически присвоит этим атрибутам некоторые значения, в результате чего столбцы могут быть определены не совсем так, как вам хотелось бы. Давайте попробуем выполнить такую последовательность команд. Сначала изменим столбец j так, чтобы он не мог содержать значения NULL и имел значение по умолчанию, равное 100, тогда результат SHOW COLUMNS будет таким:[1]

```
mysql> ALTER TABLE mytbl MODIFY j INT NOT NULL DEFAULT 100;
mysql> SHOW COLUMNS FROM mytbl LIKE 'j';
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
+-------+---------+------+-----+---------+-------+
| j     | int(11) |      |     | 100     |       |
+-------+---------+------+-----+---------+-------+
```

Пока все хорошо. Теперь, если вы захотите изменить тип столбца j с INT на BIGINT, попробуйте выполнить такое предложение:

```
mysql> ALTER TABLE mytbl MODIFY j BIGINT;
```

Однако его выполнение приведет к отмене установок NULL и DEFAULT предыдущего предложения ALTER TABLE:

```
mysql> SHOW COLUMNS FROM mytbl LIKE 'j';
+-------+------------+------+-----+---------+-------+
| Field | Type       | Null | Key | Default | Extra |
+-------+------------+------+-----+---------+-------+
| j     | bigint(20) | YES  |     | NULL    |       |
+-------+------------+------+-----+---------+-------+
```

Чтобы избежать подобного результата, необходимо явно указать атрибуты значения по умолчанию и допустимости использования значения NULL в столбце в предложении MODIFY:

```
mysql> ALTER TABLE mytbl MODIFY j BIGINT NOT NULL DEFAULT 100;
mysql> SHOW COLUMNS FROM mytbl LIKE 'j';
+-------+------------+------+-----+---------+-------+
| Field | Type       | Null | Key | Default | Extra |
+-------+------------+------+-----+---------+-------+
| j     | bigint(20) |      |     | 100     |       |
+-------+------------+------+-----+---------+-------+
```

Мораль такова: если столбец определен так, что его значение по умолчанию и допустимость NULL отличны от атрибутов, которые бы автоматически при-

---

[1] Инструкция LIKE *'строка'* предложения SHOW COLUMNS указывает, что информация должна выводиться только для столбцов, имена которых соответствуют указанной строке. Строка может содержать символы шаблона SQL для задания нескольких столбцов (см. рецепт 9.5).

своила столбцу MySQL, и вы хотите, чтобы эти атрибуты не изменялись при изменении других аспектов определения столбца, указывайте их явно в предложениях ALTER TABLE.

Это соображение учитывается в некоторых рецептах главы 9. Например, одна из программ преобразует таблицу так, чтобы в ней использовались столбцы типа VARCHAR, а не CHAR, а вторая добавляет элементы в столбцы ENUM или SET. В обоих случаях программы заботятся о том, чтобы избежать нежелательных изменений столбцов, включая спецификаторы NULL и DEFAULT в формируемые ею предложения ALTER TABLE.

8.4. Изменение значения столбца по умолчанию

Задача

Единственное, что вы хотели бы изменить в определении столбца, – это его значение по умолчанию.

Решение

Используйте SET DEFAULT для явного задания значения по умолчанию или DROP DEFAULT – чтобы удалить текущее значение по умолчанию и позволить MySQL присвоить столбцу «умолчание по умолчанию».

Обсуждение

Значение столбца по умолчанию является частью его определения, но может быть изменено отдельно от других составляющих определения. Чтобы выполнить замену значения по умолчанию, используйте ALTER *имя_столбца* SET DEFAULT:

```
ALTER TABLE mytbl ALTER j SET DEFAULT 1000;
```

Значения по умолчанию должны быть константами. Например, как это ни заманчиво, нельзя присвоить NOW() столбцу дат в качестве значения по умолчанию.

Чтобы удалить значение по умолчанию, выполните ALTER *имя_столбца* DROP DEFAULT:

```
ALTER TABLE mytbl ALTER j DROP DEFAULT;
```

В этом случае столбец по умолчанию примет стандартное значение умолчания для данного типа столбца. Для столбцов, допускающих значения NULL, это будет значение NULL. В противном случае значением по умолчанию обычно бывает ноль – для числового столбца, пустая строка – для строкового столбца и «нулевые» дата и время – для столбца даты и времени. (Исключением являются столбцы типа AUTO_INCREMENT, ENUM и TIMESTAMP, для которых значениями по умолчанию являются следующий номер последовательности, первый член перечислимого типа и текущие дата и время.)

8.5. Изменение типа таблицы

Задача

Таблица относится к одному типу, но вы знаете, что другой тип обладает более удобными свойствами для решения вашей задачи.

Решение

Для изменения типа таблицы используйте предложение ALTER TABLE с инструкцией TYPE.

Обсуждение

MySQL поддерживает несколько типов таблиц, каждый из которых обладает особыми характеристиками. Иногда бывает необходимо или желательно преобразовать таблицу из одного типа в другой. Приведем примеры некоторых ситуаций, в которых такое преобразование может оказаться полезным:

- Изменение типа таблицы может выполняться для обеспечения доступа к тем возможностям, которые присущи одному, но не другому типу. Например, таблица ISAM не позволяет использовать значения NULL в индексированных столбцах. Кроме того, столбец AUTO_INCREMENT в таблицах ISAM ведет себя не совсем обычно, так что в некоторых условиях значения последовательности могут быть немонотонными (глава 11). Вы можете преобразовать таблицу ISAM в тип MyISAM, с которым не возникает проблем подобного рода. Или вы можете обнаружить, что вам необходимо выполнять транзакции для таблицы, созданной с таким типом, который этого не позволяет. Чтобы решить проблему, можно изменить тип таблицы, например, на InnoDB или BDB, которые поддерживают транзакции.
- Старейшим типом таблиц, поддерживаемым MySQL, является ISAM, но такие таблицы сейчас подвергаются резкой критике и вскоре перестанут поддерживаться. Так что в некоторый момент вы можете захотеть преобразовать ваши таблицы ISAM (если такие есть) в какой-то другой тип. В противном случае после отмены поддержки ISAM вы не сможете произвести обновление версии MySQL.

Для изменения типа таблицы используйте ALTER TABLE со спецификатором TYPE. Например, чтобы преобразовать таблицу в тип MyISAM, выполните предложение:

```
ALTER TABLE имя_таблицы TYPE = MYISAM;
```

Чтобы узнать текущий тип таблицы, используйте предложение SHOW TABLE STATUS (появилось в версии MySQL 3.23.0) или SHOW CREATE TABLE (введено в MySQL 3.23.20):

```
mysql> SHOW TABLE STATUS LIKE 'mytbl'\G
*************************** 1. row ***************************
           Name: mytbl
           Type: MyISAM
     Row_format: Fixed
```

```
           Rows: 0
 Avg_row_length: 0
    Data_length: 0
Max_data_length: 85899345919
   Index_length: 1024
      Data_free: 0
 Auto_increment: NULL
    Create_time: 2002-07-15 21:28:34
    Update_time: 2002-07-15 21:28:34
     Check_time: NULL
 Create_options:
        Comment:
mysql> SHOW CREATE TABLE mytbl\G
*************************** 1. row ***************************
       Table: mytbl
Create Table: CREATE TABLE `mytbl` (
  `c` char(10) default NULL,
  `j` bigint(20) default NULL,
  `e2` enum('hardware','software','books','office supplies',
'telecommunications','furniture','utilities','shipping','tax') default NULL
) TYPE=MyISAM
```

В качестве альтернативы можно использовать утилиту командной строки *mysqldump*:

```
% mysqldump --no-data cookbook mytbl
CREATE TABLE mytbl (
  c char(10) default NULL,
  j bigint(20) default NULL,
  e2 enum('hardware','software','books','office supplies',
'telecommunications','furniture','utilities','shipping','tax') default NULL
) TYPE=MyISAM;
```

8.6. Переименование таблицы

Задача

Необходимо изменить имя таблицы.

Решение

Используйте для этого ALTER TABLE или RENAME TABLE.

Обсуждение

Для того чтобы переименовать таблицу, используйте опцию RENAME предложения ALTER TABLE:

```
ALTER TABLE старое_имя RENAME TO новое_имя;
```

В MySQL версии 3.23.23 было введено явное предложение RENAME TABLE:

```
RENAME TABLE старое_имя TO новое_имя;
```

RENAME TABLE позволяет изменить названия нескольких таблиц, благодаря чему вы, например, можете осуществить перестановку имен двух таблиц в одном предложении:

```
RENAME TABLE имя1 TO врем_имя, имя2 TO имя1, врем_имя to имя2;
```

Такого же эффекта можно достичь и при помощи ALTER TABLE, только понадобится три отдельных предложения. При этом на короткие промежутки времени между предложениями таблицы становятся доступны пользователям, что может быть нежелательно. Единственное предложение RENAME TABLE позволяет избежать таких проблем.

Предложение RENAME TABLE также удобно использовать для циклического сдвига таблиц. Если вы хотите сделать так, чтобы таблица лога никогда не переставала быть доступной для клиентских приложений, создайте ее пустую версию с временным именем, затем сдвигайте файлы, используя предложение RENAME TABLE. Например, если вы хотите хранить ежемесячные таблицы протоколов, имена которых содержат год и месяц, можно сделать что-то типа:

```
CREATE TABLE log_temp (...);
RENAME TABLE log TO log_2001_05, log_temp TO log;
```

Если же вам нужно осуществить циклический сдвиг ежедневных таблиц за последнюю неделю, можно каждый день выполнять такие предложения:

```
CREATE TABLE log_temp (...);
DROP TABLE IF exists log_7;
RENAME TABLE log_6 TO log_7,
    log_5 TO log_6,
    log_4 TO log_5,
    log_3 TO log_4,
    log_2 TO log_3,
    log_1 TO log_2,
    log TO log_1,
    log_tmp TO log;
```

8.7. Добавление и удаление индексов

Задача

Просмотр таблицы выполняется слишком медленно. Или вставка и обновление записей требуют слишком много времени.

Решение

Предложение ALTER TABLE умеет удалять и добавлять не только столбцы, но и индексы для этих столбцов. Подобные операции часто улучшают производительность базы данных. Обычно индексирование часто используемого столбца ускоряет выполнение предложений SELECT за счет отсутствия необходимости полного просмотра таблиц. В некоторых случаях пользу может принести и удаление индекса. При любом обновлении строки MySQL приходится обновлять все индексы, содержащие измененные столбцы. Если вы редко

используете какой-то индекс, это может свидетельствовать о том, что таблица перегружена индексами, и удаление какого-то из них может повысить эффективность обработки таблицы.

Обсуждение

Для удобства работы начнем с создания нового экземпляра тестовой таблицы mytbl. Используем предложения DROP TABLE и CREATE TABLE для удаления существующей версии и воссоздания таблицы в ее первоначальной форме:

```
DROP TABLE mytbl;
CREATE TABLE mytbl
(
    i   INT,
    c   CHAR(1)
);
```

В начале главы мы применяли SHOW COLUMNS для наблюдения за результатами изменения таблицы. Теперь будем исследовать изменения индекса и выводить результаты при помощи SHOW INDEX, а не SHOW COLUMNS. В настоящий момент в таблице нет индексов, так как они не были указаны в предложении CREATE TABLE:

```
mysql> SHOW INDEX FROM mytbl;
Empty set (0.00 sec)
```

Добавление индексов

Существует четыре типа предложений, добавляющих индексы в таблицу:

```
ALTER TABLE имя_таблицы ADD PRIMARY KEY (список_столбцов);
ALTER TABLE имя_таблицы ADD UNIQUE имя_индекса (список_столбцов);
ALTER TABLE имя_таблицы ADD INDEX имя_индекса (список_столбцов);
ALTER TABLE имя_таблицы ADD FULLTEXT имя_индекса (список_столбцов);
```

Первое предложение добавляет первичный ключ (PRIMARY KEY), то есть индексированные значения должны быть уникальными и не содержать NULL. Второе предложение создает индекс, для которого значения должны быть уникальными (за исключением значений NULL, которые могут встречаться многократно). Третье предложение добавляет обычный индекс, в котором любое значение может появляться несколько раз. Последнее же создает специальный индекс FULLTEXT, который используется для просмотра текста. FULLTEXT-поиск подробно обсуждается в рецепте 4.11.

Если в конструкциях предложений есть *имя_индекса*, то оно не является обязательным. Если не указать его, MySQL автоматически присвоит индексу имя. Столбцы для индексирования указываются в параметре *список_столбцов* – списке из одного или нескольких имен столбцов, разделенных запятыми. Рассмотрим два простых примера: первый создает одностолбцовый индекс для c, а второй – многостолбцовый индекс, включающий c и i:

```
ALTER TABLE mytbl ADD INDEX (c);
ALTER TABLE mytbl ADD INDEX (c,i);
```

Во многих случаях индексируемые столбцы должны быть объявлены как не-NULL. Например, если вы создадите mytbl как таблицу типа ISAM, то приведенные выше предложения ADD INDEX не выполнятся, так как таблицы ISAM не допускают NULL ни в каких типах индексов. Кроме того, индексы типа PRIMARY KEY не могут содержать значения NULL вне зависимости от типа таблицы. Если вы пытаетесь добавить индекс, а MySQL жалуется на проблемы, связанные с NULL, используйте предложение ALTER TABLE для изменения соответствующего столбца (столбцов) на не-NULL и повторите попытку создания индекса. Например, если попробовать сделать первичным ключом столбец i, возникнет ошибка:

```
mysql> ALTER TABLE mytbl ADD PRIMARY KEY (i);
ERROR 1171 at line 5: All parts of a PRIMARY KEY must be NOT NULL;
If you need NULL in a key, use UNIQUE instead
```

Необходимо предварительно переопределить столбец i так, чтобы он не допускал использования NULL:

```
mysql> ALTER TABLE mytbl MODIFY i INT NOT NULL;
mysql> ALTER TABLE mytbl ADD PRIMARY KEY (i);
```

Все получилось. А в первом случае, как видно из сообщения об ошибке, вместо первичного ключа можно было бы создать индекс UNIQUE в случае необходимости присутствия в индексе значений NULL.

Удаление индексов

Чтобы удалить индекс, используйте одно из предложений:

```
ALTER TABLE имя_таблицы DROP PRIMARY KEY;
ALTER TABLE имя_таблицы DROP INDEX имя_индекса;
```

Проще всего удалить индекс PRIMARY KEY, так как не нужно знать имя индекса:

```
ALTER TABLE mytbl DROP PRIMARY KEY;
```

Чтобы удалить индекс, не являющийся первичным ключом, необходимо указать его имя. Если вы не знаете, как называется индекс, используйте SHOW INDEX. Во избежание вывода чересчур длинных строк используем вертикальный вывод (\G):

```
mysql> SHOW INDEX FROM mytbl\G
*************************** 1. row ***************************
       Table: mytbl
  Non_unique: 1
    Key_name: c
Seq_in_index: 1
 Column_name: c
   Collation: A
 Cardinality: NULL
    Sub_part: NULL
      Packed: NULL
     Comment:
*************************** 2. row ***************************
       Table: mytbl
```

```
    Non_unique: 1
      Key_name: c_2
   Seq_in_index: 1
   Column_name: c
     Collation: A
   Cardinality: NULL
      Sub_part: NULL
        Packed: NULL
       Comment:
*************************** 3. row ***************************
       Table: mytbl
    Non_unique: 1
      Key_name: c_2
   Seq_in_index: 2
   Column_name: i
     Collation: A
   Cardinality: NULL
      Sub_part: NULL
        Packed: NULL
       Comment:
```

Значения `Key_name` и `Seq_in_index` соответствуют именам индексов и позициям столбцов в индексе. Теперь вы знаете, что в таблице `mytbl` есть одностолбцовый индекс с именем `c` и многостолбцовый индекс с именем `c_2` (эти имена выбраны MySQL для двух созданных нами ранее индексов). Предложение, удаляющее индексы, будет таким:

```
ALTER TABLE mytbl DROP INDEX c, DROP INDEX c_2;
```

Как видите, в одном предложении `ALTER TABLE` можно выполнить несколько операций, которые необходимо разделять запятыми.

См. также

Альтернативой `ALTER TABLE` при изменении индексов является использование предложений `CREATE INDEX` и `DROP INDEX`. Внутри себя MySQL отображает их на предложения `ALTER TABLE`, так что с их помощью нельзя сделать ничего такого, что не умело бы предложение `ALTER TABLE` (дополнительную информацию вы можете найти в справочном руководстве по MySQL).

8.8. Удаление дубликатов путем добавления индекса

Задача

В таблице есть повторяющиеся записи, а вы хотели бы от них избавиться.

Решение

В качестве одного из способов решения задачи можно предложить создание уникального индекса для столбца или столбцов, содержащих дубликаты.

Обсуждение

Если при создании индекса PRIMARY KEY или UNIQUE MySQL обнаруживает повторяющиеся значения ключа, операция ALTER TABLE прерывается. Чтобы игнорировать дубликаты и продолжать работу, используйте ALTER IGNORE TABLE вместо ALTER TABLE. Ключевое слово IGNORE указывает MySQL, что следует сохранить первое из повторяющихся значений ключа и отбросить все остальные. На самом деле это удобный способ удаления дубликатов из столбца или множества столбцов. Просто создайте индекс с уникальными значениями и позвольте MySQL «выбросить» все дубликаты. (Однако такой прием не подходит, если вам необходимо знать, какие именно значения ключа были дублирующимися. О распознавании дубликатов рассказано в рецепте 14.3.)

Чтобы посмотреть, как работает IGNORE, удаляя повторения, используем таблицу mytbl, в которой больше нет индексов (если вы выполнили все рассмотренные ранее преобразования). Начнем со вставки в таблицу нескольких одинаковых значений:

```
mysql> INSERT INTO mytbl (i,c) VALUES(1,'a'),(1,'a'),(1,NULL),(1,NULL),
    -> (2,'a'),(2,'a'),(2,'b'),(2,'b');
mysql> SELECT * FROM mytbl;
+---+------+
| i | c    |
+---+------+
| 1 | a    |
| 1 | a    |
| 1 | NULL |
| 1 | NULL |
| 2 | a    |
| 2 | a    |
| 2 | b    |
| 2 | b    |
+---+------+
```

Теперь предположим, что вы хотите создать уникальный индекс, содержащий столбцы i и c. Индекс PRIMARY KEY использовать нельзя, так как c содержит значения NULL. Вы можете создать индекс UNIQUE, но если попытаетесь сделать это без IGNORE, то получите ошибку:

```
mysql> ALTER TABLE mytbl ADD UNIQUE (i,c);
ERROR 1062 at line 1: Duplicate entry '1-a' for key 1
```

Добавьте в предложение ключевое слово IGNORE, затем используйте SELECT, чтобы посмотреть на содержимое таблицы после удаления дубликатов:

```
mysql> ALTER IGNORE TABLE mytbl ADD UNIQUE (i,c);
mysql> SELECT * FROM mytbl;
+---+------+
| i | c    |
+---+------+
| 1 | NULL |
| 1 | NULL |
| 1 | a    |
```

```
| 2 | a    |
| 2 | b    |
+---+------+
```

Повторяющиеся записи были удалены, за исключением тех, что содержат значения NULL в ключевых столбцах. Дело в том, что индексы UNIQUE допускают множественные значения NULL. Приемы, позволяющие удалить и дубликаты, содержащие NULL, приведены в рецепте 14.6.

8.9. Использование предложения ALTER TABLE для нормализации таблицы

Задача

У вас есть таблица, не приведенная к нормальной форме.

Решение

ALTER TABLE поможет нормализовать ее.

Обсуждение

В предыдущих разделах в общих чертах рассказывалось о том, как использовать ALTER TABLE. В данном разделе рассматривается конкретное применение предложения ALTER TABLE на примере изменения таблицы, содержащей избыточные данные и, соответственно, не приведенной к нормальной форме.

Пусть у вас есть таблица client_billing, содержащая информацию об оплачиваемых услугах:

```
CREATE TABLE client_billing
(
    id          INT UNSIGNED NOT NULL,  # идентификационный номер клиента
    name        CHAR(20) NOT NULL,      # имя клиента
    address     CHAR(20) NOT NULL,      # адрес клиента
    date        DATE NOT NULL,          # дата предоставления услуги
    minutes     INT NOT NULL,           # количество оплачиваемых минут
    description CHAR(60) NOT NULL       # предоставленная услуга
);
```

Пока для каждого клиента есть всего одна строка, таблица выглядит замечательно:

```
+----+------+---------------+------------+---------+------------------+
| id | name | address       | date       | minutes | description      |
+----+------+---------------+------------+---------+------------------+
| 21 | John | 46 North Ave. | 2001-07-15 |      48 | consult by phone |
| 43 | Toby | 123 Elm St.   | 2001-07-13 |      12 | office visit     |
+----+------+---------------+------------+---------+------------------+
```

Но когда вы введете новые данные и одному клиенту будет соответствовать несколько строк, станет очевидной избыточность некоторой информации.

В частности, адреса и фамилии клиентов зачем-то хранятся в каждой записи, хотя такая информация для каждого конкретного клиента необходима лишь единожды:

```
+----+------+---------------+------------+---------+------------------+
| id | name | address       | date       | minutes | description      |
+----+------+---------------+------------+---------+------------------+
| 21 | John | 46 North Ave. | 2001-07-15 |      48 | consult by phone |
| 21 | John | 46 North Ave. | 2001-07-19 |     120 | court appearance |
| 43 | Toby | 123 Elm St.   | 2001-07-13 |      12 | office visit     |
| 43 | Toby | 123 Elm St.   | 2001-07-14 |      60 | draft proposal   |
| 43 | Toby | 123 Elm St.   | 2001-07-16 |     180 | present proposal |
+----+------+---------------+------------+---------+------------------+
```

Для исправления ситуации следует разбить информацию на две таблицы и сопоставлять их записи, используя значения `id`:

- Одна таблица (`client_info`) будет хранить информацию, уникальную для каждого клиента, она содержит по одной строке для каждого клиента: идентификатор, фамилия и адрес.
- Вторая таблица (`bill_item`) будет хранить информацию об услугах, за которые выставляется счет: дата, количество минут и описание предоставленной услуги. Каждая строка содержит идентификационный номер клиента, так что ей можно сопоставить соответствующую запись таблицы `client_info`.

Другими словами, таблицу `client_billing` можно разделить на таблицы `client_info` и `bill_item` так:

Таблица `client_info`:

```
+----+------+---------------+
| id | name | address       |
+----+------+---------------+
| 21 | John | 46 North Ave. |
| 43 | Toby | 123 Elm St.   |
+----+------+---------------+
```

Таблица `bill_item`:

```
+----+------------+---------+------------------+
| id | date       | minutes | description      |
+----+------------+---------+------------------+
| 21 | 2001-07-15 |      48 | consult by phone |
| 21 | 2001-07-19 |     120 | court appearance |
| 43 | 2001-07-13 |      12 | office visit     |
| 43 | 2001-07-14 |      60 | draft proposal   |
| 43 | 2001-07-16 |     180 | present proposal |
+----+------------+---------+------------------+
```

Чтобы выполнить такое преобразование, сначала создайте таблицы `client_info` и `bill_item`, определяя все столбцы так же, как и в исходной таблице `client_billing`:

```
CREATE TABLE client_info
(
    id          INT UNSIGNED NOT NULL,  # идентификатор клиента
    name        CHAR(20) NOT NULL,      # имя клиента
    address     CHAR(20) NOT NULL       # адрес клиента
);
CREATE TABLE bill_item
(
    id          INT UNSIGNED NOT NULL,  # идентификатор клиента
    date        DATE NOT NULL,          # дата предоставления услуги
    minutes     INT NOT NULL,           # количество оплачиваемых минут
    description CHAR(60) NOT NULL       # описание предоставленной услуги
);
```

Затем используйте INSERT INTO ... SELECT для копирования соответствующих столбцов из таблицы client_billing в две новые таблицы. В таблицу client_info скопируем информацию о клиентах так:

```
INSERT INTO client_info (id,name,address)
SELECT id,name,address FROM client_billing;
```

Аналогично для таблицы bill_item:

```
INSERT INTO bill_item (id,date,minutes,description)
SELECT id,date,minutes,description FROM client_billing;
```

Записи двух новых таблиц связаны посредством значений id, так что было бы удобно индексировать этот столбец в каждой из таблиц, чтобы сделать связь более эффективной. Однако не станем ограничиваться просто созданием предложения ALTER TABLE *имя_таблицы* ADD INDEX (id) для каждой таблицы. Во-первых, таблица client_info содержит несколько записей для каждого клиента, которые необходимо опять свернуть в единую запись. Значит, будем создавать для столбца id индекс PRIMARY KEY или UNIQUE, используя ключевое слово IGNORE для удаления повторяющихся записей. Кроме того, логично предположить, что многие запросы к таблице bill_item будут использовать дату, так что можно включить в индекс столбец date. Создающие такие индексы предложения ALTER TABLE выглядят так:

```
ALTER IGNORE TABLE client_info ADD PRIMARY KEY (id);
ALTER TABLE bill_item ADD INDEX (id, date);
```

После выполнения только что описанной процедуры таблица client_billing больше не понадобится, и ее можно удалить:

```
DROP TABLE client_billing;
```

Когда записи об оплачиваемых услугах хранятся в нескольких таблицах, это немного усложняет запросы, извлекающие такую информацию. Но, в конце концов, сила реляционных СУБД именно в отношениях между таблицами. Например, чтобы вывести имя и адрес клиента из таблицы client_info вместе с общим количеством оплачиваемых минут, перечисленных для каждого клиента в таблице bill_item, выполним такой запрос:

```
mysql> SELECT client_info.id, client_info.name, client_info.address,
    -> SUM(bill_item.minutes) AS 'total minutes'
    -> FROM client_info, bill_item
    -> WHERE client_info.id = bill_item.id
    -> GROUP BY client_info.id;
+----+------+---------------+---------------+
| id | name | address       | total minutes |
+----+------+---------------+---------------+
| 21 | John | 46 North Ave. |           168 |
| 43 | Toby | 123 Elm St.   |           252 |
+----+------+---------------+---------------+
```

О запросах, обращающихся к нескольким таблицам, подробно рассказано в главе 12.

Выше был рассмотрен пример нормализации таблицы, в которой значения данных повторялись *в разных* строках. Возможна и другая не нормальная форма таблицы, когда несколько столбцов внутри строки содержат одну и ту же информацию. Например, если вы занимаетесь исследованием, в ходе которого проводите для испытуемых два теста и записываете дату и результат каждого теста, то можете использовать таблицу с такой структурой:

```
CREATE TABLE test_subject
(
    id      INT UNSIGNED NOT NULL AUTO_INCREMENT,
    name    CHAR(20) NOT NULL,  # имя испытуемого
    date1   DATE,               # дата и результат первого теста
    result1 INT,
    date2   DATE,               # дата и результат второго теста
    result2 INT,
    PRIMARY KEY (id)
);
```

Информация в таблице будет выглядеть так:

```
+----+--------+------------+---------+------------+---------+
| id | name   | date1      | result1 | date2      | result2 |
+----+--------+------------+---------+------------+---------+
|  1 | Fred   | 2001-07-13 |      78 | 2001-07-14 |      85 |
|  2 | Barry  | 2001-07-12 |      79 | 2001-07-14 |      82 |
|  3 | Portia | 2001-07-16 |      82 | 2001-07-18 |      95 |
+----+--------+------------+---------+------------+---------+
```

Если теперь вы захотите вычислить, например, средний результат каждого протестированного, то выполните запрос, подобный предложенному:

```
SELECT id, name, (result1 + result2) / 2 FROM test_subject;
```

Если вы захотите провести третий и четвертый тесты, то без проблем сможете добавить в таблицу столбцы:

```
ALTER TABLE test_subject
ADD date3 DATE, ADD result3 INT,
ADD date4 DATE, ADD result4 INT;
```

Но запрос, вычисляющий средний результат, усложнится:

```
SELECT id, name, (result1 + result2 + result3 + result4) / 4
FROM test_subject;
```

Если результат допускает значение NULL, показывающее, что тест еще не пройден, то запрос будет гораздо более запутанным, так как складывать можно только значения не-NULL (чтобы получить сумму не-NULL). Возрастающая сложность является сигналом того, что структуру таблицы можно усовершенствовать. Проблема в том, что она зависит от количества проводимых тестов. Если вы изменяете количество тестов, приходится изменять и таблицу. Создадим две таблицы, устроенные так, что их не придется изменять при вводе дополнительных испытаний:

- Первая таблица будет хранить уникальную информацию о каждом испытуемом.
- Вторая таблица будет хранить результаты тестов, при этом каждому результату будет соответствовать строка. Каждая строка может быть сопоставлена с испытуемым посредством идентификационного номера.

Прием очень похож на использованный в предыдущем примере, когда мы создали две таблицы для хранения информации по оплате для клиентов. Однако вместо того чтобы удалять исходную таблицу после создания двух новых (как в первом случае), сохраним исходную таблицу, хотя и в несколько измененном виде:

- Создадим таблицу test_result для хранения дат и результатов тестов, а также идентификаторов испытуемых. Чтобы обеспечить сортировку результатов, можно добавить столбец с номером теста. (Для упорядочивания можно было бы использовать дату, но кто-то ведь может пройти несколько тестов за один день. С явным номером теста не возникнет таких проблем.)
- Скопируем идентификаторы испытуемых, даты и результаты из таблицы test_subject в test_result.
- Удалим из таблицы test_subject столбцы даты и результата, оставив только идентификаторы и имена испытуемых.

Начнем с создания таблицы для хранения результатов тестов:

```
CREATE TABLE test_result
(
    id          INT UNSIGNED NOT NULL,
    test_num    INT NOT NULL,
    date        DATE,
    result      INT
);
```

Скопируем информацию о тестах из test_subject в test_result. Необходимо выполнить операцию отдельно для каждого теста, так как результаты берутся из разных наборов столбцов:

```
INSERT INTO test_result (id,test_num,date,result)
SELECT id,1,date1,result1 FROM test_subject WHERE result1 IS NOT NULL;
```

```
INSERT INTO test_result (id,test_num,date,result)
SELECT id,2,date2,result2 FROM test_subject WHERE result2 IS NOT NULL;
```

Каждое предложение INSERT INTO ... SELECT указывает значение test_num «вручную», так как соответствующее значение не содержится в таблице test_subject и не может быть непосредственно получено из ее содержимого. Инструкция WHERE обеспечивает копирование только строк с не-NULL-результатами теста. Тем самым обрабатывается возможность неполноты записей test_subject (значение NULL указывает на то, что испытуемый еще не прошел данный тест). Если в таблице test_subject есть результаты всех тестов, в инструкции WHERE нет необходимости, и ее можно опустить.

Таблица test_result заполнена данными, можно индексировать ее. Заметьте, что, хотя столбец id в таблице test_subject является столбцом с уникальными значениями, в таблице test_result это уже не так, ведь для каждого испытуемого в ней содержится несколько записей. Однако можно создать уникальный индекс, используя сочетание id и test_num, считая, что каждый тест предлагается каждому участнику лишь единожды:

```
ALTER TABLE test_result ADD PRIMARY KEY (id, test_num);
```

Столбцы с результатами тестов в таблице test_subject больше не нужны, и их можно удалить:

```
ALTER TABLE test_subject DROP date1, DROP result1, DROP date2, DROP result2;
```

Преимущество использования двух таблиц заключается в том, что запросы, выполняющие операции, подобные вычислению среднего, перестают зависеть от количества проведенных тестов:

```
SELECT id, AVG(result) FROM test_result GROUP BY id;
```

Чтобы вывести и имена испытуемых, выполним соединение таблицы test_result с таблицей test_subject:

```
SELECT test_result.id, test_subject.name, AVG(test_result.result)
FROM test_subject, test_result
WHERE test_subject.id = test_result.id
GROUP BY test_result.id;
```

Можно определить испытуемых, которые прошли не все тесты. Например, если проводилось всего четыре теста, можно найти количество испытуемых, которым в таблице test_result соответствует менее четырех результатов:

```
SELECT test_subject.id, test_subject.name, COUNT(test_result.result) AS count
FROM test_subject LEFT JOIN test_result ON test_subject.id = test_result.id
GROUP BY test_subject.id
HAVING count < 4;
```

Запрос использует LEFT JOIN, чтобы обеспечить учет всех испытуемых, в том числе тех, тесты которых еще не обработаны. (Обычному соединению не удалось бы идентифицировать испытуемого, для которого существует запись в таблице test_subject, но еще нет записей в test_result, так как соответствие между таблицами не было бы установлено.) О LEFT JOIN будет подробно рассказано в главе 12.

9

Получение и использование метаданных

9.0. Введение

Большинство написанных ранее запросов работали с данными, хранящимися в базе данных. Собственно, именно для этого и создаются базы данных. Но иногда нам требуются не просто значения данных, а нечто большее: характеристики или описания этих значений, то есть метаданные (metadata). Метаданные часто используются при обработке результирующих множеств, но могут применяться и в других аспектах вашего взаимодействия с MySQL. В этой главе рассказано о том, как получить и применять следующие типы метаданных:

Информация о результатах запроса. При удалении или обновлении множества строк можно определить количество строк, которые были изменены. Для запроса `SELECT` можно также вычислить количество столбцов результирующего множества и получить для каждого столбца результата такую информацию, как имя столбца и длина выводимых значений (ширина вывода). Подобные сведения часто оказываются необходимы для обработки результатов. Например, если вы форматируете табличный вывод, то можете определить, насколько широким должен быть каждый столбец и следует ли выравнивать значения по правому или левому краю.

Информация о таблицах и базах данных. Данные, относящиеся к структуре таблиц и баз данных, полезны в приложениях, которые должны выводить перечни таблиц базы данных или баз данных, работающих на сервере (например, чтобы отобразить для пользователя список возможных вариантов для выбора). Вы можете использовать информацию и для того, чтобы определить, существует ли таблица или база данных. Кроме того, с помощью табличных метаданных можно определять разрешенные значения для столбцов `ENUM` и `SET`.

Информация о сервере MySQL. Некоторые API предоставляют информацию о сервере базы данных или о текущем соединении с сервером. Зная версию сервера, вы сможете определить, какие возможности он поддерживает, – такая информация весьма полезна для создания адаптивных

приложений. К данным о соединении относятся имена текущего пользователя и текущей базы данных.

Некоторые API пытаются обеспечить независящий от базы данных интерфейс для тех типов метаданных, которые доступны во многих СУБД (например, имена столбцов результирующего множества). Но обычно метаданные тесно связаны со структурой СУБД и, соответственно, зависят от базы данных. То есть если вы захотите перенести приложение, использующее рецепты этой главы, на другую СУБД, может потребоваться некоторая его переработка. Например, чтобы получить списки таблиц и баз данных в MySQL, следует использовать предложения `SHOW`. Однако `SHOW` является специальным расширением SQL, существующим только в MySQL, поэтому даже если вы будете работать с таким API, как DBI, DB-API или JDBC, которые обеспечивают независящий от базы данных способ выдачи запросов, в данном случае сам SQL будет зависеть от базы данных и потребует изменений для переноса на другую платформу.

Сценарии, содержащие код приведенных примеров, хранятся в каталоге *metadata* дистрибутива `recipes`. (Некоторые из них вызывают функции из каталога *lib*.) Чтобы создать таблицы, используемые в примерах, обратитесь к каталогу *tables*.

Иногда рецепты будут формировать запросы, используя имя базы данных, таблицы или столбца, хранящееся в переменной. Для простоты такие имена обычно вставляются в строку запроса:

```
$query = "SHOW COLUMNS FROM $tbl_name";
```

В большинстве случаев все работает хорошо, но возможны некоторые сложности, что стоит учитывать при использовании рецептов в собственных программах. Начиная с версии MySQL 3.23.6, в именах допустимы практически все символы, например, пробелы. Если вы предполагаете возможность работы с такими именами, заключайте их в обратные кавычки:

```
$query = "SHOW COLUMNS FROM `$tbl_name`";
```

Если сервер работает в режиме ANSI, имена следует заключать в двойные кавычки:

```
$query = "SHOW COLUMNS FROM \"$tbl_name\"";
```

Вы можете запросить версию сервера (см. рецепт 9.13), чтобы узнать, как она соотносится с MySQL 3.23.6 или более поздней. Чтобы узнать, работает ли сервер в режиме ANSI, выполните команду `SHOW VARIABLES`. Все рассматриваемые примеры не проводят этих проверок, чтобы не отвлекаться от своей главной цели – решения поставленных задач.

9.1. Определение количества строк, обработанных запросом

Задача

Вы хотите узнать, сколько строк было обработано запросом.

Решение

Некоторые API предоставляют счетчик в виде значения, возвращаемого функцией, выдающей запрос. В других имеется отдельная функция, вызываемая после выполнения запроса.

Обсуждение

Для запросов, обрабатывающих строки (`UPDATE`, `DELETE`, `INSERT`, `REPLACE`), каждый API предоставляет возможность определить количество строк, подвергшихся соответствующему изменению. В MySQL «обработанный» обычно означает «измененный», поэтому те строки, которые запрос не изменяет, не учитываются, даже если они соответствуют условиям, указанным в запросе. Например, для приведенного ниже предложения `UPDATE` количество «обработанных» строк будет равно 0, потому что оно не изменяет текущие значения столбцов (вне зависимости от того, сколько строк удовлетворяют условию инструкции `WHERE`):

```
UPDATE limbs SET arms = 0 WHERE arms = 0;
```

Perl

В сценариях, основанных на DBI, количество строк, измененных запросом, возвращает функция `do()` или `execute()`, в зависимости от того, как выполняется запрос:

```
# выполнить $query, используя do()
my $count = $dbh->do ($query);
# вывести 0 строк в случае ошибки
printf "%d rows were affected\n", (defined ($count) ? $count : 0);

# выполнить запрос, используя prepare() и execute()
my $sth = $dbh->prepare ($query);
my $count = $sth->execute ();
printf "%d rows were affected\n", (defined ($count) ? $count : 0);
```

При использовании DBI вы можете указать MySQL на необходимость вывода количества строк, соответствующих условиям запроса, а не измененных им. Для этого укажите `mysql_client_found_rows=1` в части опций имени источника данных, передаваемого в качестве аргумента при вызове `connect()` для соединения с сервером MySQL, например:

```
my $dsn =
    "DBI:mysql:cookbook:localhost;mysql_client_found_rows=1";
my $dbh = DBI->connect ($dsn, "cbuser", "cbpass", { PrintError => 0, RaiseError => 1 });
```

`mysql_client_found_rows` изменяет способ подсчета строк на время соединения.

PHP

В PHP для получения количества строк, измененных запросом, вызовите функцию `mysql_affected_rows()`:

```
$result_id = mysql_query ($query, $conn_id);
```

```
# вернуть 0 строк, если запрос вызвал ошибку
$count = ($result_id ? mysql_affected_rows ($conn_id) : 0);
print ("$count rows were affected\n");
```

Аргументом `mysql_affected_rows()` является идентификатор соединения. Если не указать аргумент, используется текущее соединение.

Python

В DB-API для Python счетчик строк доступен в виде значения атрибута `rowcount` курсора запроса:

```
cursor = conn.cursor ()
cursor.execute (query)
print "%d rows were affected" % cursor.rowcount
```

Java

Интерфейс Java JDBC предоставляет два способа подсчета количества строк в зависимости от метода, вызванного для выполнения запроса. Если используется `executeUpdate()`, то он возвращает непосредственно количество строк:

```
Statement s = conn.createStatement ();
int count = s.executeUpdate (query);
s.close ();
System.out.println (count + " rows were affected");
```

Если же вы применяете метод `execute()`, то он возвращает «истину» или «ложь». Значения показывают, генерирует ли предложение результирующее множество. Для предложений, не возвращающих результат, таких как `UPDATE` и `DELETE`, можно получить количество строк, обработанных ими, при помощи метода `getUpdateCount()`:

```
Statement s = conn.createStatement ();
if (!s.execute (query))
{
    // нет результирующего множества, вывести счетчик строк
    System.out.println (s.getUpdateCount () + " rows were affected");
}
s.close ();
```

Для предложений, изменяющих строки, драйвер MySQL Connector/J JDBC обеспечивает подсчет количества строк, удовлетворяющих условиям запроса, а не измененных запросом.

9.2. Получение метаданных результирующего множества

Задача

Вы знаете, как извлекать строки результирующего множества, но хотите получить дополнительную информацию о самом множестве: имена и типы данных столбцов, количество строк и столбцов.

Решение

Используйте возможности вашего API.

Обсуждение

Для запросов, формирующих результирующее множество, можно получить много различных метаданных. В разделе рассказано о сведениях, предоставляемых в каждом из API, на примере программ, выводящих метаданные результирующего множества, полученного после выполнения простого запроса (`SELECT name, foods FROM profile`). Также описаны возможности применения этой информации. Во многих примерах используется простейший способ: если вы извлекаете строки значений из результирующего множества и хотите обрабатывать их в цикле, то количество столбцов результирующего множества используется как верхняя граница переменной цикла.

Perl

В интерфейсе DBI вы можете получить результирующее множество двумя способами. Они отличаются объемом метаданных результирующего множества, доступных вашему сценарию:

Обработка запроса при помощи дескриптора предложения. В данном случае вы вызываете `prepare()` для получения дескриптора предложения, затем вызываете его метод `execute()` для формирования результирующего множества и в цикле извлекаете строки. При использовании такого подхода доступ к метаданным обеспечивается на время активности результирующего множества, то есть с момента вызова `execute()` и до извлечения последней строки результирующего множества. Когда извлекающий строки метод обнаруживает, что строк больше нет, он неявно вызывает `finish()`, и метаданные становятся недоступными (то же самое произойдет, если метод `finish()` будет вызван явно). То есть лучше всего обращаться к метаданным сразу же после вызова `execute()`, копируя все значения, которые могут понадобиться вне цикла выборки.

Обработка запроса при помощи метода дескриптора базы данных, возвращающего результирующее множество в одной операции. Если вы применяете этот подход, любые метаданные, сформированные при обработке запроса, будут ликвидированы к моменту завершения работы метода, хотя вы все же сможете определить количество столбцов и строк по размеру результирующего множества.

Когда для обработки запроса применяется дескриптор предложения, DBI делает метаданные результирующего множества доступными после того, как вы вызываете метод дескриптора `execute()`. Изначально информация представляет собой ссылки на массивы. Для каждого типа метаданных существует отдельный массив, содержащий по элементу для каждого столбца результирующего множества. К ссылкам на массивы можно обращаться как к атрибутам дескриптора предложения. Например, `$sth->{NAME}` указывает на массив названий столбцов. Отдельные имена столбцов – это элементы такого массива:

```
$name = $sth->{NAME}->[$i];
```

А можно обратиться ко всему массиву целиком:

```
@names = @{$sth->{NAME}};
```

Перечень и описание атрибутов, используя которые вы можете обращаться к метаданным, хранящимся в массиве, приведен в табл. 9.1. Имена, начинающиеся с прописной буквы, соответствуют стандартным атрибутам DBI и должны поддерживаться большинством СУБД. Имена атрибутов, начинающиеся с `mysql`, присущи только MySQL и являются непереносимыми; предоставляемая ими информация может быть доступна и в других базах данных, но с помощью атрибутов с другими именами.

Таблица 9.1. Атрибуты доступа к массиву метаданных

Имя атрибута	Описание элемента массива
NAME	Имя столбца
NAME_lc	Имя столбца в нижнем регистре
NAME_uc	Имя столбца в верхнем регистре
NULLABLE	1, если столбец допускает значения NULL, в противном случае пустая строка
PRECISION	Ширина столбца
SCALE	Количество десятичных разрядов (для числовых столбцов)
TYPE	Числовой тип столбца (значение DBI)
mysql_is_blob	Истина, если столбец имеет тип BLOB (или TEXT)
mysql_is_key	Истина, если столбец включен в неуникальный ключ
mysql_is_num	Истина, если столбец имеет числовой тип
mysql_is_pri_key	Истина, если столбец включен в первичный ключ
mysql_max_length	Фактическая максимальная длина значений столбцов результирующего множества
mysql_table	Имя таблицы, в которую входит столбец
mysql_type	Числовой тип столбца (внутреннее значение MySQL)
mysql_type_name	Имя типа столбца

Кроме метаданных, представленных в форме массива, вы можете получить доступ к скалярному значению, представляющему количество столбцов результирующего множества:

```
$num_cols = $sth->{NUM_OF_FIELDS};
```

Приведем пример кода, иллюстрирующего выполнение запроса и отображение метаданных результирующего множества:

```
my $query = "SELECT name, foods FROM profile";
printf "Query: %s\n", $query;
```

```
my $sth = $dbh->prepare ($query);
$sth->execute();
# с этого момента доступны метаданные ...
printf "NUM_OF_FIELDS: %d\n", $sth->{NUM_OF_FIELDS};
print "Note: query has no result set\n" if $sth->{NUM_OF_FIELDS} == 0;
for my $i (0 .. $sth->{NUM_OF_FIELDS}-1)
{
    printf "--- Column %d (%s) ---\n", $i, $sth->{NAME}->[$i];
    printf "NAME_lc:          %s\n", $sth->{NAME_lc}->[$i];
    printf "NAME_uc:          %s\n", $sth->{NAME_uc}->[$i];
    printf "NULLABLE:         %s\n", $sth->{NULLABLE}->[$i];
    printf "PRECISION:        %s\n", $sth->{PRECISION}->[$i];
    printf "SCALE:            %s\n", $sth->{SCALE}->[$i];
    printf "TYPE:             %s\n", $sth->{TYPE}->[$i];
    printf "mysql_is_blob:    %s\n", $sth->{mysql_is_blob}->[$i];
    printf "mysql_is_key:     %s\n", $sth->{mysql_is_key}->[$i];
    printf "mysql_is_num:     %s\n", $sth->{mysql_is_num}->[$i];
    printf "mysql_is_pri_key: %s\n", $sth->{mysql_is_pri_key}->[$i];
    printf "mysql_max_length: %s\n", $sth->{mysql_max_length}->[$i];
    printf "mysql_table:      %s\n", $sth->{mysql_table}->[$i];
    printf "mysql_type:       %s\n", $sth->{mysql_type}->[$i];
    printf "mysql_type_name:  %s\n", $sth->{mysql_type_name}->[$i];
}
$sth->finish (); # освободить результирующее множество, т. к. его строки не выбираются
```

Если использовать приведенный выше код для выполнения запроса `SELECT name, foods FROM profile`**, вывод будет таким:**

```
Query: SELECT name, foods FROM profile
NUM_OF_FIELDS: 2
--- Column 0 (name) ---
NAME_lc:          name
NAME_uc:          NAME
NULLABLE:
PRECISION:        20
SCALE:            0
TYPE:             1
mysql_is_blob:
mysql_is_key:
mysql_is_num:     0
mysql_is_pri_key:
mysql_max_length: 7
mysql_table:      profile
mysql_type:       254
mysql_type_name:  char
--- Column 1 (foods) ---
NAME_lc:          foods
NAME_uc:          FOODS
NULLABLE:         1
PRECISION:        42
SCALE:            0
TYPE:             1
```

```
mysql_is_blob:
mysql_is_key:
mysql_is_num:     0
mysql_is_pri_key:
mysql_max_length: 21
mysql_table:      profile
mysql_type:       254
mysql_type_name:  char
```

Для получения количества строк результирующего множества, сформированного вызовом `execute()`, необходимо извлечь строки и пересчитать их самостоятельно. (Применение `$sth->rows()` для подсчета строк, возвращаемых предложением `SELECT`, резко осуждается в документации DBI.)

Результирующее множество может быть сгенерировано и в результате вызова одного из методов DBI, использующих дескриптор базы данных, а не предложения, такого как `selectall_arrayref()` или `selectall_hashref()`. Для таких методов доступ к метаданным столбцов не предоставляется. Эта информация уже будет удалена к тому моменту, когда метод завершится, поэтому не будет доступна сценариям. Однако вы можете получить количество строк и столбцов, исследуя само результирующее множество. Способ такого исследования зависит от структуры данных, формируемой методом. Структуры результатов и способы получения количества их строк и столбцов обсуждаются в рецепте 2.4.

PHP

В PHP метаданные становятся доступны после успешного выполнения вызова `_query()` и остаются доступными до момента вызова `mysql_free_result()`. Для обращения к метаданным передайте идентификатор результирующего множества, возвращенный `mysql_query()`, той функции, которая выдает необходимую вам информацию. Для получения количества строк или столбцов результирующего множества вызовите `mysql_num_rows()` или `mysql_num_fields()`. Метаданные определенного столбца результирующего множества упаковываются в единый объект. Объект получается путем передачи в `mysql_fetch_field()` идентификатора результирующего множества и индекса столбца, а доступ к различным атрибутам метаданных организован как к членам этого объекта (табл. 9.2):

Таблица 9.2. Члены объекта метаданных

Имя члена	Описание
`blob`	`1`, если столбец относится к типу `BLOB` (или `TEXT`), `0` – в противном случае
`max_length`	Фактическая максимальная длина значений столбца результирующего множества
`multiple_key`	`1`, если столбец входит в неуникальный ключ, `0` – в противном случае
`name`	Имя столбца
`not_null`	`1`, если столбец не допускает значения `NULL`, `0` – в противном случае

Таблица 9.2 (продолжение)

Имя члена	Описание
numeric	1, если столбец имеет числовой тип, 0 – в противном случае
primary_key	1, если столбец входит в первичный ключ, 0 – в противном случае
table	Имя таблицы, в которую входит столбец
type	Имя типа столбца
unique_key	1, если столбец входит в уникальный ключ, 0 – в противном случае
unsigned	1, если столбец имеет атрибут UNSIGNED, 0 – в противном случае
zerofill	1, если столбец имеет атрибут ZEROFILL, 0 – в противном случае

Следующий фрагмент кода показывает, как получить доступ к метаданным результирующего множества и напечатать их:

```
$query = "SELECT name, foods FROM profile";
print ("Query: $query\n");
$result_id = mysql_query ($query, $conn_id);
if (!$result_id)
    die ("Query failed\n");
# с этого момента доступны метаданные ...
# ниже используется @, так как mysql_num_rows() и mysql_num_fields() выводят
# сообщение, если нет результирующего множества (по крайней мере, в PHP 4)
$nrows = @mysql_num_rows ($result_id);
print ("Number of rows: $nrows\n");
$ncols = @mysql_num_fields ($result_id);
print ("Number of columns: $ncols\n");
if ($ncols == 0)
    print ("Note: query has no result set\n");
for ($i = 0; $i < $ncols; $i++)
{
    $col_info = mysql_fetch_field ($result_id, $i);
    printf ("--- Column %d (%s) ---\n", $i, $col_info->name);
    printf ("blob:         %s\n", $col_info->blob);
    printf ("max_length:   %s\n", $col_info->max_length);
    printf ("multiple_key: %s\n", $col_info->multiple_key);
    printf ("not_null:     %s\n", $col_info->not_null);
    printf ("numeric:      %s\n", $col_info->numeric);
    printf ("primary_key:  %s\n", $col_info->primary_key);
    printf ("table:        %s\n", $col_info->table);
    printf ("type:         %s\n", $col_info->type);
    printf ("unique_key:   %s\n", $col_info->unique_key);
    printf ("unsigned:     %s\n", $col_info->unsigned);
    printf ("zerofill:     %s\n", $col_info->zerofill);
}
if ($ncols > 0)     # освободить результирующее множество, если оно было
    mysql_free_result ($result_id);
```

Вывод программы будет таким:

```
Query: SELECT name, foods FROM profile
Number of rows: 10
Number of columns: 2
--- Column 0 (name) ---
blob:         0
max_length:   7
multiple_key: 0
not_null:     1
numeric:      0
primary_key:  0
table:        profile
type:         string
unique_key:   0
unsigned:     0
zerofill:     0
--- Column 1 (foods) ---
blob:         0
max_length:   21
multiple_key: 0
not_null:     0
numeric:      0
primary_key:  0
table:        profile
type:         string
unique_key:   0
unsigned:     0
zerofill:     0
```

Python

DB-API для Python имеет более ограниченные (по сравнению с другими API) возможности по получению метаданных результирующего множества. Количества столбцов и строк доступны, но вот информации об отдельных столбцах не слишком много.

Чтобы получить количество строк результирующего множества, обратитесь к атрибуту курсора `rowcount`. Количество столбцов напрямую недоступно, но вы можете вызвать `fetchone()` или `fetchall()`, а затем вычислить длину любого кортежа строки результирующего множества. Можно определить количество столбцов и вообще без извлечения строк, используя `cursor.description`. Это кортеж, содержащий по одному элементу из каждого столбца результирующего множества, так что его длина равна количеству столбцов результата. (Но помните, что если запрос не формирует результирующее множество, как, например, предложение `UPDATE`, значением `description` будет `None`.) Каждый элемент кортежа `description` — это другой кортеж, представляющий метаданные соответствующего столбца результирующего множества. Для каждого столбца есть семь значений метаданных. Покажем, как к ним обратиться и что они означают:

```
query = "SELECT name, foods FROM profile"
print "Query: ", query
```

```
cursor = conn.cursor ()
cursor.execute (query)
# с этого момента доступны метаданные ...
print "Number of rows:", cursor.rowcount
if cursor.description == None:  # no result set
    ncols = 0
else:
    ncols = len (cursor.description)
print "Number of columns:", ncols
if ncols == 0:
    print "Note: query has no result set"
for i in range (ncols):
    col_info = cursor.description[i]
    # вывести имя, затем другую информацию
    print "--- Column %d (%s) ---" % (i, col_info[0])
    print "Type:         ", col_info[1]
    print "Display size: ", col_info[2]
    print "Internal size:", col_info[3]
    print "Precision:    ", col_info[4]
    print "Scale:        ", col_info[5]
    print "Nullable:     ", col_info[6]
cursor.close
```

Вывод программы выглядит так:

```
Query:  SELECT name, foods FROM profile
Number of rows: 10L
Number of columns: 2
--- Column 0 (name) ---
Type:          254
Display size:  7
Internal size: 20
Precision:     20
Scale:         0
Nullable:      0
--- Column 1 (foods) ---
Type:          254
Display size:  21
Internal size: 42
Precision:     42
Scale:         0
Nullable:      1
```

Java

В JDBC доступ к метаданным результирующего множества реализуется через объект `ResultSetMetaData`**, который получается при вызове метода** `getMetaData()` **объекта** `ResultSet`**. Объект метаданных обеспечивает доступ к различным видам информации. Его метод** `getColumnCount()` **возвращает количество столбцов результирующего множества. Другие типы метаданных предоставляют информацию об отдельных столбцах и принимают в качестве аргумента**

индекс столбца. Обратите внимание, что в JDBC значения индекса начинаются с 1, а не с 0, в отличие от DBI, PHP и DB-API.

```
String query = "SELECT name, foods FROM profile";
System.out.println ("Query: " + query);
Statement s = conn.createStatement ();
s.executeQuery (query);
ResultSet rs = s.getResultSet ();
ResultSetMetaData md = rs.getMetaData ();
// с этого момента доступны метаданные...
int ncols = md.getColumnCount ();
System.out.println ("Number of columns: " + ncols);
if (ncols == 0)
    System.out.println ("Note: query has no result set");
for (int i = 1; i <= ncols; i++)    // значения индексов столбцов начинаются с 1
{
    System.out.println ("--- Column " + i + " (" + md.getColumnName (i) + ") ---");
    System.out.println ("getColumnDisplaySize: " + md.getColumnDisplaySize (i));
    System.out.println ("getColumnLabel:       " + md.getColumnLabel (i));
    System.out.println ("getColumnType:        " + md.getColumnType (i));
    System.out.println ("getColumnTypeName:    " + md.getColumnTypeName (i));
    System.out.println ("getPrecision:         " + md.getPrecision (i));
    System.out.println ("getScale:             " + md.getScale (i));
    System.out.println ("getTableName:         " + md.getTableName (i));
    System.out.println ("isAutoIncrement:      " + md.isAutoIncrement (i));
    System.out.println ("isNullable:           " + md.isNullable (i));
    System.out.println ("isCaseSensitive:      " + md.isCaseSensitive (i));
    System.out.println ("isSigned:             " + md.isSigned (i));
}
rs.close ();
s.close ();
```

Вывод программы будет таким:

```
Query: SELECT name, foods FROM profile
Number of columns: 2
--- Column 1 (name) ---
getColumnDisplaySize: 20
getColumnLabel:       name
getColumnType:        1
getColumnTypeName:    CHAR
getPrecision:         0
getScale:             0
getTableName:         profile
isAutoIncrement:      false
isNullable:           0
isCaseSensitive:      true
isSigned:             false
--- Column 2 (foods) ---
getColumnDisplaySize: 42
getColumnLabel:       foods
getColumnType:        1
```

```
getColumnTypeName:    CHAR
getPrecision:         0
getScale:             0
getTableName:         profile
isAutoIncrement:      false
isNullable:           1
isCaseSensitive:      true
isSigned:             false
```

Как и в DBI, непосредственный доступ к количеству строк не предоставляется, вам необходимо извлечь строки и пересчитать их самостоятельно.

На самом деле в JDBC есть и другие вызовы для получения метаданных результирующего множества, не рассмотренные в примере, но многие из них не выводят полезную для MySQL информацию. Если вы захотите их опробовать, посмотрите в руководстве по JDBC, что это за вызовы, и измените программу так, чтобы посмотреть, что они возвращают (если возвращают).

9.3. Определение наличия или отсутствия результирующего множества

Задача

Вы выполнили запрос, полученный от внешнего источника, и точно не знаете, возвращает ли он результирующее множество.

Решение

Проверьте количество столбцов в метаданных. Если оно равно 0, то результирующего множества нет.

Обсуждение

Если вы пишете приложение, принимающее строки запроса из внешних источников (файл, пользовательский ввод с клавиатуры), вы можете не знать, возвращает ли указанный запрос результирующее множество. А такая информация необходима, так как запросы, возвращающие результирующее множество, обрабатываются не так, как запросы, не возвращающие его. Можно после выполнения запроса проверить значение метаданных, соответствующее количеству столбцов. Если считать, что запрос выполнился корректно, то нулевой счетчик столбцов показывает, что выполнялось предложение `INSERT`, `UPDATE` или какое-то другое, не возвращающее результирующее множество. Ненулевое значение означает наличие результирующего множества, так что вы можете приступать к извлечению строк. Этот прием позволяет отличать запросы `SELECT` от прочих, поскольку он применим и к запросам `SELECT`, возвращающим пустое результирующее множество. (Пустое результирующее множество – это не то же самое, что его отсутствие. В первом случае строки не возвращаются, но количество столбцов отлично от 0, а во втором случае столбцов вообще нет.)

Некоторые API предоставляют и другие способы распознавания типа запроса. В JDBC можно создавать произвольные запросы при помощи метода `execute()`, который явно указывает на наличие или отсутствие результирующего множества, возвращая «истину» или «ложь» соответственно. В Python значение `cursor.description` равно `None` для предложений, не формирующих результирующее множество.

9.4. Форматирование результатов запроса для отображения

Задача

Вы хотите вывести результирующее множество красиво отформатированным.

Решение

Прибегните к помощи метаданных. Они могут предоставить вам важную информацию о структуре и содержимом результата.

Обсуждение

Метаданные весьма полезны для форматирования результатов запроса, так как сообщают нам множество полезных вещей, например, имена и ширину вывода столбцов, даже в тех случаях, когда неизвестно, что именно делал запрос. Например, можно написать универсальную функцию, выводящую результирующее множество, в табличной форме, не имея сведений о природе запроса. Рассмотрим фрагмент кода на Java, который получает объект результирующего множества и применяет его для получения метаданных результата. Затем оба объекта используются вместе для извлечения и форматирования значений результирующего множества. Вывод похож на вывод *mysql*: за строкой с заголовками столбцов следуют строки результата, столбцы которого заключены в красивые рамочки и выровнены по вертикали. Вот как функция выведет результирующее множество, сформированное запросом `SELECT id, name, birth FROM profile`:

```
+----------+--------------------+----------+
|id        |name                |birth     |
+----------+--------------------+----------+
|1         |Fred                |1970-04-13|
|2         |Mort                |1969-09-30|
|3         |Brit                |1957-12-01|
|4         |Carl                |1973-11-02|
|5         |Sean                |1963-07-04|
|6         |Alan                |1965-02-14|
|7         |Mara                |1968-09-17|
|8         |Shepard             |1975-09-02|
|9         |Dick                |1952-08-20|
|10        |Tony                |1960-05-01|
```

```
|11        |Juan                |NULL      |
+----------+--------------------+----------+
11 rows selected
```

Первый вопрос, на который должно ответить такое приложение: «Какова правильная ширина вывода каждого из столбцов?». Метод `getColumnDisplaySize()` возвращает ширину столбца, но необходимо учитывать еще ряд соображений:

- Следует принять во внимание длину имени столбца, которое может превышать длину значений столбца.
- Для значений `NULL` будет отображаться слово «NULL», поэтому для столбцов, допускающих такие значения, ширина вывода должна быть не меньше четырех символов.

Напишем Java-функцию `displayResultSet()`, которая форматирует результирующее множество, принимая в расчет перечисленные выше факторы. Кроме того, при извлечении строк она подсчитывает их количество, так как в JDBC получить это значение непосредственно из метаданных невозможно.

```
public static void displayResultSet (ResultSet rs) throws SQLException
{
    ResultSetMetaData md = rs.getMetaData ();
    int ncols = md.getColumnCount ();
    int nrows = 0;
    int[] width = new int[ncols + 1];       // массив для хранения ширины столбцов
    StringBuffer b = new StringBuffer ();  // буфер для хранения ограничивающей линии

    // вычисление ширины столбцов
    for (int i = 1; i <= ncols; i++)
    {
        // некоторые драйверы возвращают -1 для getColumnDisplaySize();
        // если это так, то замещаем его длиной имени столбца
        width[i] = md.getColumnDisplaySize (i);
        if (width[i] < md.getColumnName (i).length ())
            width[i] = md.getColumnName (i).length ();
        // isNullable() возвращает 1/0, а не истина/ложь
        if (width[i] < 4 && md.isNullable (i) != 0)
            width[i] = 4;
    }

    // формируем строку +---+---...
    b.append ("+");
    for (int i = 1; i <= ncols; i++)
    {
        for (int j = 0; j < width[i]; j++)
            b.append ("-");
        b.append ("+");
    }

    // выводим ограничительную линию, заголовки столбцов, еще линию
    System.out.println (b.toString ());
    System.out.print ("|");
```

```
        for (int i = 1; i <= ncols; i++)
        {
            System.out.print (md.getColumnName (i));
            for (int j = md.getColumnName (i).length (); j < width[i]; j++)
                System.out.print (" ");
            System.out.print ("|");
        }
        System.out.println ();
        System.out.println (b.toString ());

        // выводим содержимое результата
        while (rs.next ())
        {
            ++nrows;
            System.out.print ("|");
            for (int i = 1; i <= ncols; i++)
            {
                String s = rs.getString (i);
                if (rs.wasNull ())
                    s = "NULL";
                System.out.print (s);
                for (int j = s.length (); j < width[i]; j++)
                    System.out.print (" ");
                System.out.print ("|");
            }
            System.out.println ();
        }
        // выводим ограничительную линию и количество строк
        System.out.println (b.toString ());
        if (nrows == 1)
            System.out.println ("1 row selected");
        else
            System.out.println (nrows + " rows selected");
    }
```

Если вы хотите придать выводу более законченный вид, то можете провести проверку столбца на числовые значения и, если столбец числовой, выровнять его по правому краю. В сценариях DBI и PHP это легко сделать, так как можно обратиться к атрибутам метаданных `mysql_is_num` или `numeric`, поддерживаемым этими API. В DB-API и JDBC все несколько сложнее. Значения метаданных, показывающего, является ли столбец числовым, не существует, так что приходится смотреть на индикатор типа столбца, чтобы понять, относится ли он к одному из числовых типов.

Недостатком функции `displayResultSet()` также является то, что она выводит столбцы, используя ширину, указанную в определении таблицы, а не фактическую ширину значений, реально присутствующих в результирующем множестве (которая часто бывает меньше первой). Посмотрим на пример вывода, приведенный перед функцией `displayResultSet()`. Столбцы `id` и `name` имеют ширину 10 и 20 символов, несмотря на то, что их самые большие значения имеют длину, равную двум и семи символам соответственно. В DBI, PHP

и DB-API вы можете получить максимальную ширину значений результирующего множества. Для того чтобы определить эти значения в JDBC, следует обойти результирующее множество и самостоятельно проверить длины значений столбцов. Необходим драйвер JDBC 2.0, поддерживающий прокрутку результирующих множеств. Если предположить, что вы работаете именно с этим драйвером, то можно изменить код `displayResultSet()` так:

```
// вычислить ширину столбцов
for (int i = 1; i <= ncols; i++)
{
    width[i] = md.getColumnName (i).length ();
    // isNullable() возвращает 1/0, а не истина/ложь
    if (width[i] < 4 && md.isNullable (i) != 0)
        width[i] = 4;
}
// прокрутить результирующее множество и скорректировать ширину вывода
while (rs.next ())
{
    for (int i = 1; i <= ncols; i++)
    {
        byte[] bytes = rs.getBytes (i);
        if (!rs.wasNull ())
        {
            int len = bytes.length;
            if (width[i] < len)
                width[i] = len;
        }
    }
}
rs.beforeFirst ();  // "перемотать" результат обратно перед выводом
```

После проделанных изменений результирующее множество выглядит более компактно:

```
+--+-------+----------+
|id|name   |birth     |
+--+-------+----------+
|1 |Fred   |1970-04-13|
|2 |Mort   |1969-09-30|
|3 |Brit   |1957-12-01|
|4 |Carl   |1973-11-02|
|5 |Sean   |1963-07-04|
|6 |Alan   |1965-02-14|
|7 |Mara   |1968-09-17|
|8 |Shepard|1975-09-02|
|9 |Dick   |1952-08-20|
|10|Tony   |1960-05-01|
|11|Juan   |NULL      |
+--+-------+----------+
11 rows selected
```

9.5. Получение информации о структуре таблицы

Задача

Вы хотите узнать, как определена таблица.

Решение

Есть несколько способов сделать это, начиная с предложений, непосредственно возвращающих такую информацию, и заканчивая метаданными запроса к таблице.

Обсуждение

Наличие информации о структуре таблицы позволяет ответить на вопросы: «Какие столбцы и каких типов содержит таблица?» и «Каковы разрешенные значения для столбца ENUM или SET?». В MySQL есть ряд способов узнать, какова структура таблицы:

- Использовать предложение SHOW COLUMNS.
- Использовать запрос SELECT, выбирающий столбцы из таблицы, затем исследовать метаданные запроса на предмет информации о каждом столбце.
- Использовать программу командной строки *mysqldump* или предложение SHOW CREATE TABLE для получения предложения CREATE TABLE, выводящего структуру таблицы.

В следующих разделах показано, как запрашивать информацию о таблице MySQL каждым из упомянутых способов. Для работы с примерами необходимо создать таблицу item, в которой перечислены идентификаторы изделий, их названия и цвета, имеющиеся в наличии:

```
CREATE TABLE item
(
    id      INT UNSIGNED NOT NULL AUTO_INCREMENT,
    name    CHAR(20),
    colors  SET('chartreuse','mauve','lime green','puce') DEFAULT 'puce',
    PRIMARY KEY (id)
);
```

Получение структуры таблицы с помощью предложения SHOW COLUMNS

Предложение SHOW COLUMNS формирует одну строку вывода для каждого столбца таблицы, при этом каждая строка содержит несколько элементов информации о соответствующем столбце.[1] Я советовал бы вам опробовать предложение SHOW COLUMNS на нескольких собственных таблицах, чтобы получить

[1] Предложение SHOW COLUMNS FROM *имя_таблицы* эквивалентно SHOW FIELDS FROM *имя_таблицы* или DESCRIBE *имя_таблицы*.

представление о том, какой вывод оно генерирует для различных типов столбцов. Рассмотрим пример вывода SHOW COLUMNS для таблицы item. (Обратите внимание на использование \G в качестве признака конца предложения для формирования «вертикального» вывода; дело в том, что строки вывода SHOW COLUMNS часто бывают настолько длинными, что переносятся на следующую строку, и их сложно читать.)

```
mysql> SHOW COLUMNS FROM item\G
*************************** 1. row ***************************
  Field: id
   Type: int(10) unsigned
   Null:
    Key: PRI
Default: NULL
  Extra: auto_increment
*************************** 2. row ***************************
  Field: name
   Type: char(20)
   Null: YES
    Key:
Default: NULL
  Extra:
*************************** 3. row ***************************
  Field: colors
   Type: set('chartreuse','mauve','lime green','puce')
   Null: YES
    Key:
Default: puce
  Extra:
```

Предложение выводит информацию следующим образом:

`Field`

Имя столбца.

`Type`

Тип столбца.

`Null`

`YES`, если столбец допускает значения `NULL`, иначе – пробел.

`Key`

Информация о том, индексирован ли столбец.

`Default`

Значение по умолчанию.

`Extra`

Дополнительная информация.

Формат `SHOW COLUMNS` может несколько меняться, но перечисленные значения доступны всегда. В вашей версии MySQL может отображаться еще какая-то информация.

Вывод SHOW COLUMNS удобно использовать в программах. Важно знать, что предложение SHOW подобно SELECT в том, что оно возвращает результирующее множество, поэтому для обработки запроса SHOW вы создаете предложение и организуете цикл для извлечения строк. Приведем для иллюстрации функцию PHP, которая принимает имя таблицы в качестве аргумента и использует SHOW COLUMNS для получения списка имен столбцов таблиц:

```
function get_column_names_with_show ($conn_id, $tbl_name)
{
    $query = "SHOW COLUMNS FROM $tbl_name";
    if (!($result_id = mysql_query ($query, $conn_id)))
        return (FALSE);
    $names = array();               # создать пустой массив
    # первое значение каждой строки вывода - это имя столбца
    while (list ($name) = mysql_fetch_row ($result_id))
        $names[] = $name;           # добавить имя в конец массива
    mysql_free_result ($result_id);
    return ($names);
}
```

Возвращенный функцией массив содержит имена столбцов в том же порядке, в котором столбцы присутствуют в таблице. Заметьте, что get_column_names_with_show() не принимает аргумент для указания имени базы данных. В этом нет необходимости, так как MySQL понимает полные ссылки на таблицы в формате *db_name.tbl_name*. Для обращения к таблице базы данных, отличной от текущей, просто передайте аргумент $tbl_name, содержащий имя базы данных:

```
$names = get_column_names_with_show ($conn_id, "some_db.some_tbl");
```

Аналогичная функция Python выглядит так:

```
def get_column_names_with_show (conn, tbl_name):
    names = []
    cursor = conn.cursor ()
    cursor.execute ("SHOW COLUMNS FROM " + tbl_name)
    rows = cursor.fetchall ()
    cursor.close ()
    for row in rows:
        names.append (row[0])
    return (names)
```

В DBI операция тривиальна, так как selectcol_arrayref() возвращает непосредственно первый столбец результата запроса:

```
sub get_column_names_with_show
{
my ($dbh, $tbl_name) = @_;

    my $ref = $dbh->selectcol_arrayref ("SHOW COLUMNS FROM $tbl_name");
    return (defined ($ref) ? @{$ref} : ());
}
```

Интерпретация значений по умолчанию, выводимых SHOW COLUMNS

Если вы выполняете SHOW COLUMNS из программы *mysql*, то предложение выводит слово «NULL» для значений по умолчанию в столбцах, которые могут содержать значения NULL. Однако если вы запускаете предложение SHOW COLUMNS в собственной программе, то не ищите строку «NULL». Ищите специальное значение, используемое вашим API для представления таких значений (undef в DBI или None в DB-API).

Если нужны сведения только об одном столбце, используйте инструкцию LIKE с именем интересующего вас столбца:

```
mysql> SHOW COLUMNS FROM item LIKE 'colors'\G
*************************** 1. row ***************************
  Field: colors
   Type: set('chartreuse','mauve','lime green','puce')
   Null: YES
    Key:
Default: puce
  Extra:
```

Обратите внимание на кавычки, в которые заключено имя столбца, следующего за ключевым словом LIKE. Кавычки необходимы, так как имя – это не настоящее имя столбца, а строковый шаблон SQL. Строка интерпретируется так же, как для оператора LIKE инструкции WHERE предложения SELECT. (Про поиск по образцу рассказано в рецепте 4.6.) Предложение SHOW COLUMNS выводит информацию для всех столбцов, имена которых соответствуют образцу. Если указать буквальное имя столбца, то строка будет соответствовать только этому имени, и SHOW COLUMNS выведет данные только по нужному столбцу. Однако неосторожных здесь ожидает ловушка. Если имена столбцов содержат специальные символы SQL (% или _), и вы хотите установить им буквальное соответствие, необходимо экранировать такие символы в строке образца при помощи обратного слэша, иначе другие имена тоже будут восприниматься как совпадающие с шаблоном. Символ % нечасто используется в именах столбцов, а вот символ подчеркивания _ весьма распространен, так что вы вполне можете столкнуться с описанной проблемой. Предположим, что у вас есть таблица, содержащая результаты измерений уровня углекислого газа в столбце co_2 и вычисления тригонометрических функций косинуса и котангенса в столбцах cos1, cos2, cot1 и cot2. Если вы хотите получить информацию только о столбце co_2, то не следует использовать запрос:

```
SHOW COLUMNS FROM имя_таблицы LIKE 'co_2';
```

Символ _ в поиске по образцу означает «соответствие любому символу», поэтому запрос вернул бы строки co_2, cos2 и cot2. Чтобы получить только co_2, изменим команду SHOW:

```
SHOW COLUMNS FROM имя_таблицы LIKE 'co\_2';
```

В программе можно использовать возможности языка API для экранирования символов шаблона SQL перед передачей имен столбцов в запрос SHOW. Например, в Perl, PHP и Python можно использовать такие выражения:

Perl:

```
$name =~ s/([%_])/\\$1/g;
```

PHP:

```
$name = ereg_replace ("([%_])", "\\\\1", $name);
```

В Python импортируйте модуль re, затем выполните:

```
name = re.sub (r'([%_])', r'\\1', name)
```

Если вам покажется, что выражения содержат слишком много символов обратного слэша, вспомните о том, что транслятор языка API сам интерпретирует такие символы и перед выполнением сравнения с образцом «снимает верхний слой». Чтобы получить в результирующем множестве буквальный символ обратного слэша, необходимо продублировать его в шаблоне. В PHP появляется еще один уровень, так как здесь свою лепту вносит еще и обработчик шаблона.

В Java необходимо выбрать библиотеку класса регулярного выражения. В следующем примере используется библиотека ORO, доступная по адресу *jakarta.apache.org*, которая содержит классы, эмулирующие регулярные выражения Perl5:

```
import org.apache.oro.text.perl.*;

Perl5Util util = new Perl5Util ();
name = util.substitute ("s/([_%])/\\\\$1/g", name);
```

Необходимость экранирования символов % и _ для буквального соответствия значению LIKE относится и к другим формам предложения SHOW, допускающим использование образцов в инструкции LIKE (SHOW TABLES, SHOW DATABASES и SHOW VARIABLES).

Получение структуры таблицы с помощью метаданных результирующего множества

Еще один способ получения информации о столбцах таблицы – использование метаданных, формируемых при выдаче предложения SELECT (см. рецепт 9.2). Метаданные доступны для любого запроса, поэтому их можно специально получить для столбцов указанной таблицы. Например, результирующее множество запроса SELECT * FROM *имя_таблицы* будет содержать метаданные каждого столбца таблицы *имя_таблицы*. Однако если вас интересуют *только* метаданные, но не сами данные таблицы, разумным желанием будет минимизировать размер результирующего множества для генерирования минимального сетевого трафика. Для того чтобы обеспечить пустоту результирующего множества, просто добавим в запрос заведомо ложную инструкцию WHERE:

```
SELECT * FROM имя_таблицы WHERE 1 = 0
```

Несмотря на то что такой запрос не выбирает строк, он совершенно корректен; MySQL обработает его и получит метаданные, из которых можно извлечь любую необходимую информацию. Например, ранее в этом разделе мы писали функцию PHP `get_column_names_with_show()`, получавшую перечень имен столбцов с помощью предложения `SHOW COLUMNS`. Функцию можно переделать так, чтобы она получала имена столбцов из метаданных столбца, вместо того чтобы выдавать запрос `SELECT` и вызывать `mysql_fetch_field()`:

```
function get_column_names_with_meta ($conn_id, $tbl_name)
{
    $query = "SELECT * FROM $tbl_name WHERE 1 = 0";
    if (!($result_id = mysql_query ($query, $conn_id)))
        return (FALSE);
    $names = array();               # создать пустой массив
    for ($i = 0; $i < mysql_num_fields ($result_id); $i++)
    {
        if ($field = mysql_fetch_field ($result_id, $i))
            $names[] = $field->name;    # добавить имя в конец массива
    }
    mysql_free_result ($result_id);
    return ($names);
}
```

Аналогичная функция Perl будет проще. DBI организует метаданные в массивы, так что нужно просто обратиться к ссылке `$sth->{NAME}` на массив имен столбцов. Единственная хитрость заключается в том, что необходимо создать копию массива перед вызовом `finish()`, так как `finish()` уничтожает метаданные и делает недоступным массив `NAME`:

```
sub get_column_names_with_meta
{
my ($dbh, $tbl_name) = @_;
my ($sth, @names);

    $sth = $dbh->prepare ("SELECT * FROM $tbl_name WHERE 1=0");
    $sth->execute ();
    @names = @{$sth->{NAME}};   # создать копию; finish() разрушает метаданные
    $sth->finish ();            # освободить результирующее множество
    return (@names);
}
```

Без труда можно преобразовать функции, получающие имена столбцов, к более общему виду, чтобы указать, какие именно метаданные извлекать. Давайте назовем эти функции `get_column_info()` и добавим параметр для указания типа информации. В PHP функция будет такой:

```
function get_column_info ($conn_id, $tbl_name, $info_type)
{
    $query = "SELECT * FROM $tbl_name WHERE 1 = 0";
    if (!($result_id = mysql_query ($query, $conn_id)))
        return (FALSE);
    $info = array();                        # создать пустой массив
```

```
    for ($i = 0; $i < mysql_num_fields ($result_id); $i++)
    {
        if ($field = mysql_fetch_field ($result_id, $i))
            $info[] = $field->$info_type;   # добавить информацию в массив
    }
    mysql_free_result ($result_id);
    return ($info);
}
```

Чтобы использовать функцию, вызовите ее следующим образом:

```
$names = get_column_info ($conn_id, "item", "name");
$types = get_column_info ($conn_id, "item", "type");
$numeric = get_column_info ($conn_id, "item", "numeric");
```

Версия на Perl выглядит так:

```
sub get_column_info
{
my ($dbh, $tbl_name, $info_type) = @_;
my ($sth, @info);

    $sth = $dbh->prepare ("SELECT * FROM $tbl_name WHERE 1=0");
    $sth->execute ();
    @info = @{$sth->{$info_type}};  # создать копию; finish() разрушает метаданные
    $sth->finish ();                # освободить результирующее множество
    return (@info);
}
```

И вызывается так:

```
my @names = get_column_info ($dbh, "item", "NAME");
my @types = get_column_info ($dbh, "item", "mysql_type_name");
my @numeric = get_column_info ($dbh, "item", "mysql_is_num");
```

Работая с `get_column_info()`, помните, что функцию нельзя использовать для получения ширины вывода столбцов таблицы. (То есть она бесполезна для значений `mysql_max_length` в Perl и значений `max_length` в PHP.) Если используются метаданные столбца, полученные из предложения `SELECT`, ширина вывода столбца будет соответствовать ширине значений, *фактически присутствующих в результирующем множестве*. Если речь идет о запросе `SELECT ... WHERE 1=0`, то его результирующее множество пусто, и для всех столбцов ширина вывода будет равна 0!

Переносимость SELECT ... WHERE 1=0

Запрос `SELECT ... WHERE 1=0` можно переносить в другие СУБД, но он не обязательно будет работать со всеми СУБД. Если вы хотите использовать этот запрос для получения информации о таблице не в MySQL, сначала проверьте, работает ли он в вашей СУБД.

Получение структуры таблицы с помощью предложения CREATE TABLE

Третий способ получения от MySQL информации о структуре таблицы – выполнение в командной строке *mysqldump --no-data* для формирования предложения CREATE TABLE, которое показывает структуру таблицы. Рассмотрим один пример. Опция *--no-data* указывает программе *mysqldump*, что не нужно выгружать данные из таблицы, *--all* задает вывод всех опций предложения CREATE TABLE, а *--quote-names* означает заключение в кавычки названий, содержащих специальные символы.[1] Можно опустить опцию *--all* или *--quote-names*, если они для вас не важны.

```
% mysqldump --no-data --all --quote-names cookbook item
# MySQL dump 8.16
#
# Host: localhost    Database: cookbook
#--------------------------------------------------------
# Server version        3.23.46-log

#
# Table structure for table 'item'
#

CREATE TABLE `item` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `name` char(20) default NULL,
  `colors` set('chartreuse','mauve','lime green','puce') default 'puce',
  PRIMARY KEY  (`id`)
) TYPE=MyISAM;
```

Если вы работаете с версией MySQL 3.23.20 или выше, то можете получить ту же информацию при помощи предложения SHOW CREATE TABLE:

```
mysql> SHOW CREATE TABLE item;
+-------+--------------+
| Table | Create Table |
+-------+--------------+
| item  | CREATE TABLE `item` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `name` char(20) default NULL,
  `colors` set('chartreuse','mauve','lime green','puce') default 'puce',
  PRIMARY KEY  (`id`)
) TYPE=MyISAM |
+-------+--------------+
```

Вывод информативен и удобен для восприятия, так как информация о столбцах выводится в формате, аналогичном используемому при первичном создании таблицы. Явно показана структура индекса, чего не обеспечивают другие методы. Однако вы, наверное, согласитесь с тем, что этот способ проверки структуры таблицы больше подходит для визуального контроля, чем

[1] Опция *--all* появилась в MySQL 3.22.23, а *--quote-names* – в версии 3.23.6.

Какой метод следует использовать для получения информации о таблице?

Можно ли обойтись использованием только SHOW COLUMNS или только SELECT ... WHERE 1=0 для получения всей необходимой информации о таблицах? Не всегда. Данные, получаемые этими двумя способами, совпадают только частично. Если вам нужно определить имена столбцов или узнать, допускают ли столбцы использование значений NULL, подойдет любой метод. Если же необходимо получить разрешенный диапазон значений столбца ENUM или SET, вы должны использовать SHOW COLUMNS. Познакомьтесь с возможностями разных методов и выбирайте тот, который лучше подходит для решения вашей задачи. Все зависит от того, какую именно информацию нужно получить.

для использования внутри программы. Информация выводится не в обычном формате строка-столбец, поэтому ее трудно анализировать. Кроме того, формат может несколько меняться в результате усовершенствования предложения CREATE TABLE, которые случаются периодически по мере расширения возможностей MySQL.

Конечно, это не значит, что вывод SHOW CREATE TABLE не подлежит использованию в программах. Например, он применяется для создания полной копии таблицы, включающей все ее индексы (см. рецепт 3.25, посвященный клонированию таблиц).

9.6. Получение информации о столбцах ENUM и SET

Задача

Вы хотите узнать, какие значения допустимы в качестве членов столбца ENUM или SET.

Решение

Используйте предложение SHOW COLUMNS для получения определения столбца и извлечения списка его членов.

Обсуждение

Иногда возникает необходимость получить список разрешенных значений столбца ENUM или SET. Предположим, что вы хотите создать веб-форму, где есть всплывающее меню, которое предлагает для выбора варианты, соответствующие разрешенным значениям столбца ENUM, например имеющиеся в наличии размеры одежды при заказе или возможные способы отправки при доставке. Можно жестко запрограммировать значения в сценарии, формирующем страницу, но если вы впоследствии измените столбец (например,

чтобы добавить новое значение), возникнет несоответствие между столбцом и использующим его сценарием. Если же находить разрешенные значения, применяя метаданные таблицы, сценарий всегда будет генерировать меню, содержащее корректный набор значений. Аналогичный подход можно использовать и для столбцов SET.

Чтобы определить, какие значения разрешены для столбца ENUM или SET, создайте для столбца предложение SHOW COLUMNS и посмотрите на значение Type результата. Например, для столбца colors таблицы item значение Type будет таким:

```
set('chartreuse','mauve','lime green','puce')
```

Со столбцами ENUM все аналогично, только выводится enum, а не set. Чтобы получить разрешенные значения любого из этих двух типов, следует отбросить первое слово и скобки, а то, что получится, разбить на части, используя в качестве разделителей запятые и заключая получившиеся отдельные значения в кавычки. Давайте напишем функцию get_enumorset_info(),[1] которая будет извлекать значения из определения столбца. Функция будет возвращать тип столбца, его значение по умолчанию и сообщать, допускает ли столбец значения NULL. Тогда функцию можно использовать в сценариях, которым требуется нечто большее, чем просто список значений. Приведем версию функции на Python. Она принимает аргументы, представляющие собой идентификатор соединения, имя таблицы и тип столбца, и возвращает словарь записей, соответствующих различным аспектам определения столбца:

```
def get_enumorset_info (conn, tbl_name, col_name):
    # создадим словарь для хранения информации о столбцах
    info = { }
    try:
        cursor = conn.cursor ()
        # экранируем символы шаблона SQL в названиях столбцов
        # для буквального соответствия
        col_name = re.sub (r'([%_])', r'\\\1', col_name)
        # это *не* использование заполнителей
        cursor.execute ("SHOW COLUMNS FROM %s LIKE '%s'" \ % (tbl_name, col_name))
        row = cursor.fetchone ()
        cursor.close
        if row == None:
            return None
    except:
        return None

    info["name"] = row[0]
    # получаем строку типа столбца; убедимся, что она начинается с ENUM или SET
    s = row[1]
    match = re.match ("(enum|set)\((.*)\)$", s)
    if not match:
        return None
    info["type"] = match.group (1)      # тип столбца
```

[1] Вы можете придумать для нее более приятное имя.

```
    # получаем значения, разделяя список на части по запятым,
    # затем применяем к каждой части функцию заключения в кавычки
    s = re.split (",", match.group (2))
    f = lambda x: re.sub ("^'(.*)'$", "\\1", x)
    info["values"] = map (f, s)

    # определяем, допускает ли столбец значения NULL
    info["nullable"] = (row[2] == "YES")

    # получаем значение по умолчанию (None соответствует NULL)
    info["default"] = row[4]
    return info
```

Следующий пример демонстрирует один из способов обращения к каждому элементу словаря, возвращенного функцией `get_enumorset_info()`, и его вывода:

```
info = get_enumorset_info (conn, tbl_name, col_name)
print "Information for " + tbl_name + "." + col_name + ":"
if info == None:
    print "No information available (not an ENUM or SET column?)"
else:
    print "Name: " + info["name"]
    print "Type: " + info["type"]
    print "Legal values: " + string.join (info["values"], ",")
    if info["nullable"]:
        print "Nullable"
    else:
        print "Not nullable"
    if info["default"] == None:
        print "Default value: NULL"
    else:
        print "Default value: " + info["default"]
```

Такой код выводит для столбца `colors` таблицы `item` следующий результат:

```
Information for item.colors:
Type: set
Legal values: chartreuse,mauve,lime green,puce
Nullable
Default value: puce
```

Для других API эквивалентные функции выглядят аналогично. Их удобно использовать для формирования списков элементов в веб-формах (см. рецепты 18.2 и 18.3).

9.7. Способы получения информации о таблицах, не зависящие от СУБД

Задача

Вы хотите получить информацию о таблице, не используя собственные запросы MySQL типа `SHOW COLUMNS`.

Решение

Это возможно не во всех API. Одним из исключений является JDBC, предоставляющий стандартный интерфейс для метаданных таблиц.

Обсуждение

Было показано, как использовать различные API для обработки информации о таблицах, полученной при помощи способов, применяющих специальные запросы `SHOW` и `SELECT`. Эти приемы работают только в MySQL. JDBC обеспечивает доступ к информации о таблицах посредством стандартного интерфейса, не ссылающегося на конкретные запросы, который может переноситься в другие СУБД. Вы используете объект соединения для получения объекта метаданных базы данных, затем вызываете метод `getColumns()` этого объекта для извлечения информации. Метод `getColumns()` возвращает результирующее множество, содержащее строку для каждого имени столбца, так что для получения информации о нескольких столбцах следует выполнить цикл (табл. 9.3):

Таблица 9.3. Элементы строк результирующего множества для MySQL

Индекс	Описание
3	Имя таблицы
4	Имя столбца
6	Имя типа столбца
7	Размерность столбца (для числовых столбцов – точность)
8	Количество десятичных разрядов для числовых столбцов
18	Могут ли значения столбца быть `NULL`

Рассмотрим пример использования функции `getColumns()` для вывода списка имен столбцов и типов:

```
DatabaseMetaData md = conn.getMetaData ();
ResultSet rs = md.getColumns (dbName, "", tblName, "%");
int i = 0;
while (rs.next ())
{
    i++;
    System.out.println ("--- Column " + i + " ---");
    System.out.println ("Name: " + rs.getString (4));
    System.out.println ("Type: " + rs.getString (6));
}
rs.close ();
```

Если значением переменной `tblName` было бы `"item"`, вывод выглядел бы так:

```
--- Column 1 ---
Name: id
Type: int
--- Column 2 ---
```

```
Name: name
Type: char
--- Column 3 ---
Name: colors
Type: enum
```

Четыре аргумента `getColumns()` – это имена каталога, схемы и таблицы, а также шаблон SQL, которому должны соответствовать выбираемые имена столбцов. В MySQL эти аргументы таковы:

- Имя каталога – это имя базы данных. Чтобы использовать текущую базу данных, введите пустую строку.
- В MySQL нет понятия «схема», так что аргумент имени схемы к делу не относится и может быть пустой строкой.
- Аргумент имени таблицы – это символьная строка имени таблицы.
- Шаблон имени столбца аналогичен использованию инструкции `LIKE` в предложении `SHOW COLUMNS`. В примере используется символ `%`, означающий совпадение со всеми именами столбцов. Для получения информации об отдельном столбце передайте функции его имя. (Не забывайте экранировать символы `%` и `_` обратным слэшем для буквального соответствия.)

9.8. Применение информации о структуре таблицы

Задача

Конечно, хорошо уметь получать информацию о структуре таблицы, но что с ней делать дальше?

Решение

Есть множество вариантов: выводить списки столбцов таблицы, создавать элементы веб-формы, формировать предложения `ALTER TABLE` для изменения столбцов `ENUM` или `SET`, и т. д.

Обсуждение

В этом разделе описано несколько способов применения информации о структуре таблицы, предоставляемой MySQL.

Вывод списка столбцов

Вероятно, простейшим применением информации о таблице является вывод списка столбцов таблицы, который часто используется в веб-приложениях или приложениях с графическим интерфейсом пользователя для интерактивного формирования запросов. Из списка выбирается столбец таблицы и вводится значение, с которым будут сравниваться значения столбца. Основой для вывода такого списка могут послужить различные версии приведенных ранее функций `get_column_names_with_show()` и `get_column_names_with_meta()`.

Интерактивное редактирование записей

Знание структуры таблицы чрезвычайно полезно для приложений, в которых осуществляется интерактивное редактирование записей. Предположим, у вас есть приложение, которое извлекает запись из базы данных, выводит форму с содержимым записи, чтобы пользователь мог его отредактировать, а после того как пользователь внесет изменения и передаст их, выполняет обновление записи в базе данных. Вы можете использовать информацию о структуре таблицы для проверки корректности значений столбцов. Например, если столбец относится к типу ENUM, можно определить разрешенные для него значения и проверить переданное пользователем значение на соответствие. Если это столбец целого типа, можно проверить переданное значение, чтобы убедиться, что оно целиком состоит из цифр, которым может предшествовать знак. Если столбец содержит даты, можно выполнить проверку на корректность формата переданного значения.

Но что делать, если пользователь оставил поле пустым? Если, например, поле соответствует столбцу таблицы типа CHAR, следует ли установить столбец в значение NULL или в пустую строку? И на такой вопрос можно ответить, проверив структуру таблицы. Определим, допускает ли столбец использование значений NULL. Если да, то установим столбец в NULL; в противном случае – в пустую строку.

Сопоставление типов столбцов элементам веб-страницы

Некоторые типы столбцов, такие как ENUM или SET, естественным образом отображаются на элементы веб-форм:

- Столбец ENUM имеет фиксированный набор значений, из которых можно выбрать одно значение. Возникает аналогия с группой переключателей (radio button), всплывающим меню или прокручиваемым списком с возможностью выбора только одного элемента.
- Столбец SET похож на ENUM, но допускает выбор нескольких значений, что соответствует группе флажков или прокручиваемому списку с возможностью выбора нескольких элементов.

Если вы получаете информацию о таких типах столбцов при помощи SHOW COLUMNS, то без труда можете определить разрешенные значения столбца и автоматически сопоставить их соответствующим элементам формы. Благодаря этому вы можете предложить пользователям список допустимых значений, чтобы они могли осуществить выбор варианта, не вводя ничего с клавиатуры. Ранее в главе рассказывалось о том, как получить метаданные столбцов ENUM и SET. Такие методы используются в главе 18, где подробно рассмотрено создание формы.

Добавление элементов в определения столбцов ENUM и SET

Добавление нового элемента в определение столбца ENUM или SET при использовании ALTER TABLE – занятие не из приятных, так как приходится указывать

не только новый элемент, но и все уже существующие. Один из способов выполнения подобной операции с помощью *mysqldump* и редактора описан в рецепте 8.2. Можно также написать собственную программу, которая сделает за вас бульшую часть работы, используя метаданные столбца. Давайте напишем сценарий на Python, *add_element.py*, который будет автоматически формировать соответствующее предложение ALTER TABLE для указанного имени таблицы, имени столбца ENUM или SET и значения нового элемента. Предположим, вы хотите добавить в столбец colors таблицы item элемент hot pink (ярко-розовый). Текущая структура таблицы такова:

```
mysql> SHOW COLUMNS FROM item LIKE 'colors'\G
*************************** 1. row ***************************
  Field: colors
   Type: set('chartreuse','mauve','lime green','puce')
   Null: YES
    Key:
Default: puce
  Extra:
```

Сценарий *add_element.py* использует эту информацию для построения корректного предложения ALTER TABLE и его вывода:

```
% ./add_element.py item colors "hot pink"
ALTER TABLE item
    MODIFY colors
    set('chartreuse','mauve','lime green','puce','hot pink')
    NULL DEFAULT 'puce';
```

Данное предложение является выводом *add_element.py*, который вы затем можете передать программе *mysql* для незамедлительного исполнения или сохранить в файле:

```
% ./add_element.py item colors "hot pink" | mysql cookbook
% ./add_element.py item colors "hot pink" > stmt.sql
```

Второй вариант предпочтительнее, если вы хотите поместить новый элемент не в конец списка значений, где его расположит сценарий *add_element.py*. Тогда вы можете сохранить вывод в файле, а затем отредактировать *stmt.sql*, поместив элемент туда, куда следует, после чего выполнить предложение:

```
% vi stmt.sql
% mysql cookbook < stmt.sql
```

Первая часть сценария *add_element.py* импортирует необходимые модули и проверяет аргументы командной строки. Все достаточно просто:

```
#! /usr/bin/python
# add_element.py - вывод предложения ALTER TABLE для столбца ENUM или SET
# (предполагается работа с базой данных cookbook)

import sys
sys.path.insert (0, "/usr/local/apache/lib/python")
import re
```

```
import MySQLdb
import Cookbook

if len (sys.argv) != 4:
    print "Usage: add_element.py tbl_name col_name new_element"
    sys.exit (1)
tbl_name = sys.argv[1]
col_name = sys.argv[2]
elt_val = sys.argv[3]
```

После соединения с сервером MySQL (код не приводится) необходимо выполнить запрос SHOW COLUMNS для извлечения информации об указанном столбце. Это делает следующий фрагмент кода, проверяя наличие такого столбца в таблице:

```
cursor = conn.cursor ()
# экранируем символы шаблона SQL в имени столбца для буквального соответствия
esc_col_name = re.sub (r'([%_])', r'\\\1', col_name)
# это *не* использование заполнителя
cursor.execute ("SHOW COLUMNS FROM %s LIKE '%s'" % (tbl_name, esc_col_name))
info = cursor.fetchone ()
cursor.close
if info == None:
    print "Could not retrieve information for table %s, column %s" \
                                          % (tbl_name, col_name)
    sys.exit (1)
```

К этому моменту, если предложение SHOW COLUMNS успешно выполнено, выведенная им информация доступна в виде кортежа, хранящегося в переменной info. Нам понадобится несколько элементов этого кортежа. Наиболее важным является значение типа столбца, в котором есть строка enum(...) или set(...), содержащая текущее определение столбца. Эту информацию можно использовать для того, чтобы определить, действительно ли столбец имеет тип ENUM или SET, затем добавить в строку новый элемент непосредственно перед закрывающей скобкой. Для столбца colors мы хотим заменить:

```
set('chartreuse','mauve','lime green','puce')
```

На:

```
set('chartreuse','mauve','lime green','puce','hot pink')
```

Кроме того, следует проверить, допускает ли столбец значения NULL и каково значение по умолчанию, чтобы программа могла добавить соответствующую информацию в предложение ALTER TABLE. Код будет таким:

```
# получаем строку типа столбца, убеждаемся в том, что она начинается с ENUM или SET
type = info[1]
if not re.match ('(enum|set)', type):
    print "table %s, column %s is not an ENUM or SET" % (tbl_name, col_name)
    sys.exit(1)
# добавляем кавычки, вставляем запятую и новый элемент прямо перед закрывающей скобкой
elt_val = conn.literal (elt_val)
```

```
type = re.sub (r'\)$', ',' + elt_val + ')', type)

# определяем, может ли столбец содержать значения NULL
if info[2] == "YES":
    nullable = "NULL"
else:
    nullable = "NOT NULL";

# создаем инструкцию DEFAULT (заключаем в кавычки, если только это не значение NULL)
default = "DEFAULT " + conn.literal (info[4])

print "ALTER TABLE %s\n\tMODIFY %s\n\t%s\n\t%s %s;" \
            % (tbl_name, col_name, type, nullable, default)
```

Вот и все. Получилась рабочая программа по изменению столбца `ENUM` или `SET`. Но это только базовая программа, которую можно усовершенствовать несколькими способами:

- Проверьте, не входит ли уже в столбец добавляемое в него значение.
- Разрешите сценарию *add_element.py* принимать несколько аргументов после имени столбца и одновременно добавлять их в определение таблицы.
- Добавьте опцию, которая указывала бы на то, что заданный элемент необходимо не добавить, а удалить.
- Добавьте опцию, которая заставляла бы сценарий сразу же выполнять предложение `ALTER TABLE`, вместо того, чтобы выводить его.
- Если вы работаете с более ранней версией MySQL, чем 3.22.16, то используемая сценарием *add_element.py* конструкция `MODIFY` *имя_столбца* не будет распознана. Необходимо отредактировать сценарий так, чтобы он использовал `CHANGE` *имя_столбца*. Два приведенных ниже предложения эквивалентны:

```
ALTER TABLE имя_таблицы MODIFY имя_столбца определение_столбца;
ALTER TABLE имя_таблицы CHANGE имя_столбца имя_столбца определение_столбца;
```

Сценарий *add_element.py* использует предложение `MODIFY`, потому что его синтаксис более понятен.

Извлечение дат в формате не-ISO

MySQL хранит даты в формате ISO 8601 (*CCYY-MM-DD*), но часто требуется перезаписать значения дат, например при передаче табличных данных в другую программу, которая ожидает поступления дат в другом формате. Можно написать сценарий, который бы извлекал и печатал строки таблицы, используя метаданные для распознавания столбцов `DATE`, `DATETIME` и `TIMESTAMP`, и переформатировал их при помощи `DATE_FORMAT()` в интересующий вас формат даты. (Например, в рецепте 10.33 описан небольшой сценарий *iso_to_us.pl*, применяющий такой подход для преобразования дат ISO в формат, принятый в США).

Преобразование символьных столбцов из типа фиксированной длины в тип переменной длины, и наоборот

Столбцы CHAR имеют фиксированную длину, а VARCHAR – переменную. Обычно таблицы, использующие столбцы CHAR, обрабатываются быстрее, но занимают больше места, чем таблицы со столбцами VARCHAR. Чтобы упростить преобразование таблиц для использования столбцов CHAR или VARCHAR, можно обратиться к информации SHOW COLUMNS для формирования предложения ALTER TABLE, выполняющего необходимое преобразование столбцов. Напишем на Python функцию alter_to_char(), создающую предложение для замены всех столбцов VARCHAR на CHAR:

```
def alter_to_char (conn, tbl_name):
    cursor = conn.cursor ()
    cursor.execute ("SHOW COLUMNS FROM " + tbl_name)
    rows = cursor.fetchall ()
    cursor.close ()
    str = ""
    for info in rows:
        col_name = info[0]
        type = info[1]
        if re.match ('varchar', type):      # это столбец VARCHAR
            type = re.sub ("var", "", type) # преобразуем его в CHAR
            # определяем, допускает ли столбец NULL
            if info[2] == "YES":
                nullable = "NULL"
            else:
                nullable = "NOT NULL";
            # формируем инструкцию DEFAULT
            # (добавляем кавычки, только если значение не NULL)
            default = "DEFAULT " + conn.literal (info[4])
            # добавляем инструкцию MODIFY
            if str != "":
                str = str + ",\n\t"
            str = str + \
                    "MODIFY %s %s %s %s" % (col_name, type, nullable, default)
    cursor.close ()
    if str == "":
        return None
    return "ALTER TABLE " + tbl_name + "\n\t" + str
```

Пусть у вас была такая таблица:

```
CREATE TABLE chartbl
(
    c1  VARCHAR(10),
    c2  VARCHAR(10) BINARY,
    c3  VARCHAR(10) NOT NULL DEFAULT 'abc\'def'
);
```

Если передать ее имя функции alter_to_varchar(), она вернет следующее предложение:

```
ALTER TABLE chartbl
    MODIFY c1 char(10) NULL DEFAULT NULL,
    MODIFY c2 char(10) binary NULL DEFAULT NULL,
    MODIFY c3 char(10) NOT NULL DEFAULT 'abc\'def'
```

Функция, выполняющая преобразование столбцов в обратном направлении (из CHAR в VARCHAR), будет аналогичной. Вот ее вариант на Perl:

```
sub alter_to_varchar
{
my ($dbh, $tbl_name) = @_;
my ($sth, $str);

    $sth = $dbh->prepare ("SHOW COLUMNS FROM $tbl_name");
    $sth->execute ();
    while (my @row = $sth->fetchrow_array ())
    {
        if ($row[1] =~ /^char/)     # столбец CHAR
        {
            $row[1] = "var" . $row[1];
            $str .= ",\n\t" if $str;
            $str .= "MODIFY $row[0] $row[1]";
            $str .= ($row[2] eq "YES" ? "" : " NOT") . " NULL";
            $str .= " DEFAULT " . $dbh->quote ($row[4]);
        }
    }
    $str = "ALTER TABLE $tbl_name\n\t$str" if $str;
    return ($str);
}
```

Для полноты картины функция генерирует предложение ALTER TABLE, которое явно преобразует все столбцы типа CHAR в VARCHAR. В действительности достаточно преобразовать всего один такой столбец. MySQL отметит замену типа столбца фиксированной длины на тип переменной длины и автоматически преобразует все остальные столбцы фиксированной длины, имеющие эквивалент с переменной длиной.

Выбор всех столбцов, кроме некоторых

Предположим, что вы хотите извлечь «почти все» столбцы таблицы. Пусть у вас есть таблица image, содержащая столбец типа BLOB с именем data, используемый для хранения изображений, которые могут быть очень большими, и другие столбцы, характеризующие столбец BLOB, такие как идентификатор изображения, его описание и т. д. Очень просто написать запрос SELECT *, извлекающий все столбцы, но если вас интересует только описательная информация об изображениях, но не они сами, то было бы неэффективно загружать соединение с базой данных передачей значений BLOB. Вместо этого хотелось бы выбрать из записи все, *кроме* столбца data.

К сожалению, нет способа напрямую указать в SQL необходимость выбора «всех столбцов, кроме одного». Необходимо явно назвать все столбцы, кроме data. Но такой запрос достаточно просто построить при помощи информации

о структуре таблицы. Извлекаем список столбцов, удаляем тот, который следует исключить из запроса, и формируем запрос SELECT для всех оставшихся столбцов. Покажем, как выполнить такую операцию в PHP, используя написанную ранее в этой главе функцию get_column_names_with_show() для получения имен столбцов таблицы:

```
$names = get_column_names_with_show ($conn_id, $tbl_name);
$query = "";
# формируем список столбцов для выбора: все, кроме "data"
reset ($names);
while (list ($index, $name) = each ($names))
{
    if ($name == "data")
        continue;
    if ($query != "")   # ставим запятые между именами столбцов
        $query .= ",";
    $query .= $name;
}
$query = "SELECT $query FROM $tbl_name";
```

Тот же самый код построения запроса на Perl будет несколько короче (и понятнее):

```
my @names = get_column_names_with_show ($dbh, $tbl_name);
my $query = "SELECT "
            . join (",", grep (!/^data$/, @names))
            . " FROM $tbl_name";
```

Какой бы язык вы ни выбрали, результатом будет запрос, извлекающий все столбцы, кроме data. Он будет эффективнее, чем SELECT *, так как не требует передачи по сети значений BLOB. Конечно, в данном случае необходимо дополнительное обращение к серверу для выполнения предложения, извлекающего имена столбцов, так что следует учитывать контекст планируемого использования запроса SELECT. Если вам нужно просто извлечь одну запись, выбор целой строки может оказаться эффективнее, чем совершение дополнительного путешествия на сервер. Но если вы извлекаете много строк, то уменьшение объема трафика, достигаемое за счет пропуска столбцов BLOB, будет важнее, чем дополнительные расходы на запрос о структуре таблицы.

9.9. Вывод списков таблиц и баз данных

Задача

Вы хотите получить список таблиц базы данных или баз данных, работающих на сервере MySQL.

Решение

Используйте запрос SHOW TABLES или SHOW DATABASES.

Обсуждение

Чтобы получить список таблиц текущей базы данных, используйте запрос:

```
SHOW TABLES;
```

Однако если никакая база данных не была выбрана, запрос не выполнится. Чтобы избежать подобной проблемы, убедитесь в том, что имеется текущая база данных, или укажите имя базы данных явно:

```
SHOW TABLES FROM имя_БД;
```

Другая форма SHOW возвращает список баз данных, работающих на сервере:

```
SHOW DATABASES;
```

Будьте внимательны при использовании предложений SHOW

Будьте внимательны при анализе результатов SHOW TABLES и SHOW DATABASES. Результат SHOW TABLES будет пустым, если у вас нет прав на доступ к таблице. Результат SHOW DATABASES тоже может быть пустым. Если сервер был запущен с опцией *--safe-show-database* или *--skip-show-database*, то вам мало что удастся узнать при помощи SHOW DATABASES.

Если вам нужен не зависящий от базы данных способ получения списка таблиц или баз данных, и вы работаете с Perl или Java, попробуйте следующие приемы.

Что касается Perl, то DBI предоставляет функцию tables(), которая возвращает список таблиц. Она работает только для текущей базы данных:

```
my @tables = $dbh->tables ();
```

В Java можно использовать методы JDBC, предназначенные для получения списков таблиц или баз данных. Для каждого метода вызовите метод getMetaData() вашего объекта соединения и используйте полученный объект DatabaseMetaData для извлечения необходимой информации. Вот как выводится список таблиц указанной базы данных:

```
// вывод списка таблиц базы данных, имя которой указано в dbName;
// если dbName - пустая строка, используется текущая база данных
DatabaseMetaData md = conn.getMetaData ();
ResultSet rs = md.getTables (dbName, "", "%", null);
while (rs.next ())
    System.out.println (rs.getString (3));  // столбец 3 = имя таблицы
rs.close ();
```

Аналогичная процедура выводит список баз данных:

```
// вывод списка баз данных
DatabaseMetaData md = conn.getMetaData ();
ResultSet rs = md.getCatalogs ();
```

```
while (rs.next ())
    System.out.println (rs.getString (1));  // столбец 1 = имя базы данных
rs.close ();
```

9.10. Проверка существования таблицы

Задача

Вы хотите знать, существует ли таблица.

Решение

Используйте предложение SHOW TABLES, чтобы посмотреть, присутствует ли таблица в списке.

Обсуждение

Вы можете использовать предложение SHOW TABLES для проверки существования определенной таблицы, добавив инструкцию LIKE с именем таблицы:

```
SHOW TABLES LIKE 'имя_таблицы';
SHOW TABLES FROM имя_БД LIKE 'имя_таблицы';
```

Если строка возвращается, таблица существует. Если нет, то не существует. Рассмотрим функцию на Perl, осуществляющую проверку наличия таблицы:

```
sub table_exists
{
my ($dbh, $tbl_name) = @_;
my $db_clause = "";

    ($db_clause, $tbl_name) = (" FROM $1", $2) if $tbl_name =~ /(.*)\.(.*)/;
    $tbl_name =~ s/([%_])/\\$1/g;   # экранировать любые специальные символы
    return ($dbh->selectrow_array ("SHOW TABLES $db_clause LIKE '$tbl_name'"));
}
```

Функция проверяет аргумент имени таблицы на соответствие формату *имя_БД.имя_таблицы*. Если формат такой, то функция извлекает имя таблицы и добавляет в предложение инструкцию FROM. В противном случае производится проверка для текущей базы данных. Обратите внимание на то, что функция возвращает «ложь», если таблица существует, но у вас нет прав на доступ к ней.

Есть и другие способы проверки существования таблицы. Любое из приведенных ниже предложений SELECT выполнится успешно, если таблица существует, и не выполнится, если она не существует:

```
SELECT * FROM имя_таблицы WHERE 1=0;
SELECT COUNT(*) FROM имя_таблицы;
```

Чтобы использовать такие предложения в программе, сначала настройте режим диагностирования ошибок вашего API так, чтобы он не прерывал

выполнение программы в случае обнаружения ошибки. Затем попробуйте выполнить предложение и посмотреть, выполнится ли оно.

Предложение `SELECT *` предпочтительнее, чем `SELECT COUNT(*)`, для таблиц некоторых типов, таких как BDB и InnoDB, которые требуют полного просмотра таблицы для вычисления `COUNT(*)`. В таблицах же типов ISAM и MyISAM предложение `COUNT(*)` оптимизировано и использует счетчик записей, хранящийся в таблице.

9.11. Проверка существования базы данных

Задача

Вы хотите узнать, существует ли база данных.

Решение

Используйте предложение `SHOW DATABASES`, чтобы посмотреть, присутствует ли база данных в списке.

Обсуждение

`SHOW DATABASES` может использоваться для определения того, существует ли база данных, за счет добавления инструкции `LIKE` с именем базы данных:

```
SHOW DATABASES LIKE 'имя_БД';
```

Рассмотрим функцию на Perl, выполняющую такую операцию:

```
sub database_exists
{
my ($dbh, $db_name) = @_;

    $db_name =~ s/([%_])/\\$1/g;    # экранировать любые специальные символы
    return ($dbh->selectrow_array ("SHOW DATABASES LIKE '$db_name'"));
}
```

Функция возвращает «ложь», если сервер был запущен с опцией *--skip-show-database*, и вы не имеете привилегий MySQL-пользователя `root`.

9.12. Получение метаданных сервера

Задача

Вы хотите, чтобы сервер MySQL рассказал вам о себе.

Решение

Информацию о сервере возвращают несколько предложений `SELECT` и `SHOW`.

Обсуждение

В MySQL есть ряд предложений, обеспечивающих вас информацией о самом сервере и о клиентском соединении. Некоторые наиболее полезные из них перечислены в табл. 9.4. Чтобы получить информацию, выводимую любой из функций, создайте запрос, затем обработайте результирующее множество для извлечения вывода запроса. Оба предложения SHOW разрешают использование инструкции LIKE '*образец*' для ограничения результата строками, совпадающими с образцом.

Таблица 9.4. Информация о сервере

Предложение	Выводимая информация
SELECT VERSION()	Символьная строка версии сервера
SELECT DATABASE()	Имя текущей базы данных (пустая строка, если текущая база данных не выбрана)
SELECT USER()	Текущее имя пользователя
SHOW STATUS	Индикаторы статуса сервера
SHOW VARIABLES	Переменные конфигурации сервера

Все эти запросы присущи только MySQL. Если вы работаете в Java, JDBC предоставляет ряд не зависящих от базы данных методов для получения метаданных сервера; некоторые из них выводят ту же информацию, что и рассмотренные выше предложения. Используйте объект соединения для получения метаданных базы данных, затем вызовите соответствующие методы для получения необходимой информации. Полный список приведен в руководстве по JDBC, пока же взгляните на некоторых типичных представителей:

```
DatabaseMetaData md = conn.getMetaData ();
// можно получить и при помощи SELECT VERSION()
System.out.println ("Product version: " + md.getDatabaseProductVersion ());
// аналогично SELECT USER(), но без имени хоста
System.out.println ("Username: " + md.getUserName ());
```

9.13. Создание приложений, адаптирующихся к версии сервера MySQL

Задача

Вы хотите использовать некоторую возможность, которая доступна только начиная с какой-то версии MySQL.

Решение

Запросите версию сервера. Если она слишком старая, вероятно, удастся вернуться назад и использовать обходной маневр, существовавший ранее (если такой был).

Обсуждение

Если вы пишете приложение, которое может выполнять некоторые функции, только если сервер MySQL поддерживает необходимые базовые операции, то, зная номер версии сервера, вы сможете определить, доступны ли такие операции (или потребуется ли как-то обходить ситуацию, считая, что возможность для маневра существует).

Чтобы узнать версию сервера, создайте предложение `SELECT VERSION()`. Результатом будет символьная строка, похожая на `3.23.27-gamma`. Другими словами, возвращается строка, состоящая из старшего, младшего и самого младшего номеров версии, возможно, с каким-то суффиксом. Строку версии можно использовать для отображения, если вы хотите выводить для пользователя строку состояния. Однако для сравнения удобнее работать с числами, в частности, с пятизначным числом в формате *Mmmtt*, где *M*, *mm*, *tt* – это старший, младший и самый младший номера версии. Преобразование можно выполнить, разбив строку по точкам, отделив от третьей части суффикс, который начинается с первого нецифрового символа, и объединив затем фрагменты.[1]

Рассмотрим функцию DBI, которая принимает аргумент дескриптора базы данных и возвращает список из двух элементов, содержащий и строковую, и числовую форму версии сервера. Код написан в предположении, что две младшие составляющие версии не превышают 100 и занимают не более двух разрядов каждая. Это законное допущение, поскольку исходный код самой MySQL использует такой же формат.

```
sub get_server_version
{
my $dbh = shift;
my ($ver_str, $ver_num);
my ($major, $minor, $teeny);

    # извлекаем результат в скалярную строку
    $ver_str = $dbh->selectrow_array ("SELECT VERSION()");
    return undef unless defined ($ver_str);
    ($major, $minor, $teeny) = split (/\./, $ver_str);
    $teeny =~ s/\D*$//; # отбрасываем нечисловой суффикс, если такой есть
    $ver_num = $major*10000 + $minor*100 + $teeny;
    return ($ver_str, $ver_num);
}
```

Чтобы сразу получить информацию о версии в двух формах, вызовем функцию так:

```
my ($ver_str, $ver_num) = get_server_version ($dbh);
```

Чтобы вывести только одно из значений, вызов должен быть таким:

---

[1] Мой первый алгоритм преобразования разбивал строку версии по точкам, затем отделял от третьей части суффикс, начинающийся с символа `-`. Он перестал работать с того момента, когда была выпущена версия MySQL 3.23.29a-gamma. (После отделения `-gamma` от `29a-gamma` получается `29a`, то есть не число.)

```
my $ver_str = (get_server_version ($dbh))[0];   # строковый формат
my $ver_num = (get_server_version ($dbh))[1];   # числовой формат
```

Приведем примеры использования числового значения версии для проверки того, поддерживает ли сервер какие-то функции:

```
my $ver_num = (get_server_version ($dbh))[1];
printf "GET_LOCK()/RELEASE_LOCK(): %s\n",  ($ver_num >= 32127 ? "yes" : "no");
printf "Functional GRANT statement: %s\n", ($ver_num >= 32211 ? "yes" : "no");
printf "Temporary tables: %s\n",           ($ver_num >= 32302 ? "yes" : "no");
printf "Quoted identifiers: %s\n",         ($ver_num >= 32306 ? "yes" : "no");
printf "UNION statement: %s\n",            ($ver_num >= 40000 ? "yes" : "no");
printf "Subselects: %s\n",                 ($ver_num >= 40100 ? "yes" : "no");
```

9.14. Определение текущей базы данных

Задача

Была ли какая-то база данных выбрана текущей? Если да, то как она называется?

Решение

Используйте функцию `DATABASE()`.

Обсуждение

Предложение `SELECT DATABASE()` возвращает имя текущей базы данных или пустую строку, если база данных не была выбрана. Следующий код на Python использует это предложение для отображения информации о текущем соединении:

```
cursor = conn.cursor ()
cursor.execute ("SELECT DATABASE()")
row = cursor.fetchone ()
cursor.close
if row == None or len (row) == 0 or row[0] == "":
    db = "(no database selected)"
else:
    db = row[0]
print "Current database:", db
```

9.15. Определение текущего пользователя MySQL

Задача

Каково имя пользователя клиентского приложения, и с какого хоста установлено соединение?

Решение

Используйте функцию `USER()`.

Обсуждение

Предложение SELECT USER() возвращает строку вида *user@host*, указывающую имя текущего пользователя (*user*) и хост (*host*), с которого произведено соединение.[1] Чтобы выбрать только составляющую имени или хоста, выполните такие запросы:

```
SELECT SUBSTRING_INDEX(USER(),'@',1);
SELECT SUBSTRING_INDEX(USER(),'@',-1);
```

Эту информацию можно применять по-разному. Например, чтобы приложение на Perl приветствовало пользователя, сделайте нечто подобное:

```
my ($user, $host) = $dbh->selectrow_array (q{
            SELECT SUBSTRING_INDEX(USER(),'@',1),
                   SUBSTRING_INDEX(USER(),'@',-1)
            });
print "Hello, $user!  Good to see you.\n";
print "I see you're connecting from $host.\n" unless $host eq "";
```

Можно получить составляющие, извлекая значение USER() целиком, а затем разбивая его на части при помощи операции сравнения с образцом:

```
my ($user, $host) = ($dbh->selectrow_array (
                        "SELECT USER()") =~ /([^@]+)@?(.*)/);
```

Или использовать split:

```
my ($user, $host) = split (/@/, $dbh->selectrow_array ("SELECT USER()"));
```

Значения, возвращаемые функцией USER(), можно использовать для ведения журнала пользователей приложения. Простая таблица журнала может выглядеть так (значения 16 и 60 соответствуют длинам столбцов user и host в таблицах привилегий (grant table) MySQL):

```
CREATE TABLE app_log
(
    t       TIMESTAMP,
    user    CHAR(16),
    host    CHAR(60)
);
```

Чтобы вставить новые записи в таблицу app_log, используйте приведенное ниже предложение. Столбец TIMESTAMP автоматически устанавливается в текущую дату и время, так что нет необходимости вводить значения для него:

```
INSERT INTO app_log
    SET user = SUBSTRING_INDEX(USER(),'@',1),
        host = SUBSTRING_INDEX(USER(),'@',-1);
```

Таблица хранит значения user и host отдельно, так как суммирующие запросы к этим значениям эффективнее, если не приходится разбивать их на части. Например, если вы периодически проверяете, со скольких различных

[1] До версии MySQL 3.22.1 значение USER() не включало составляющую *@host*.

хостов устанавливаются соединения с сервером, удобнее один раз разделить значение USER() при создании записи, чем проделывать эту операцию при запуске каждого предложения SELECT для формирования итогов. Кроме того, если хранить значения хостов отдельно, можно индексировать столбец host, что невозможно при хранении составных значений *user@host*.

9.16. Мониторинг сервера MySQL

Задача

Вы хотите посмотреть, как был сконфигурирован сервер, или отслеживать его состояние.

Решение

Используйте для этого предложения SHOW VARIABLES и SHOW STATUS.

Обсуждение

Предложения SHOW VARIABLES и SHOW STATUS выводят информацию о конфигурации сервера и его рабочие характеристики:

```
mysql> SHOW VARIABLES;
+---------------------------------+-------------------+
| Variable_name                   | Value             |
+---------------------------------+-------------------+
| back_log                        | 50                |
| basedir                         | /usr/local/mysql/ |
| bdb_cache_size                  | 8388600           |
| bdb_log_buffer_size             | 0                 |
| bdb_home                        |                   |
...
mysql> SHOW STATUS;
+--------------------------+----------+
| Variable_name            | Value    |
+--------------------------+----------+
| Aborted_clients          | 319      |
| Aborted_connects         | 22       |
| Bytes_received           | 32085033 |
| Bytes_sent               | 26379272 |
| Connections              | 65684    |
...
```

Такая информация полезна для создания административных приложений. Например, вы можете написать фоновую программу, которая периодически исследует активность сервера. Простое приложение такого типа может запрашивать у сервера количество установленных соединений и время работы для определения усредненной активности клиентов. Запросы для получения такой информации выглядят так:

```
SHOW STATUS LIKE 'Connections';
SHOW STATUS LIKE 'Uptime';
```

Если вы не хотите заново устанавливать соединение при каждом запуске запросов, то можете запросить у сервера значение его тайм-аута для клиентских приложений и обращаться к серверу через промежутки времени, не превышающие полученное значение. Величину тайм-аута в секундах можно получить в таком запросе:

```
SHOW VARIABLES LIKE 'wait_timeout';
```

Значение по умолчанию равно 28 800 (8 часов), хотя в вашей системе оно может быть другим.

Принцип неопределенности MySQL

Принцип неопределенности Гейзенберга для измерения характеристик волны имеет свой аналог в MySQL. Если вы следите за статусом MySQL, чтобы посмотреть, как он меняется со временем, то можете заметить любопытный факт. Каждый раз, когда вы выполняете измерение, вы изменяете измеряемое значение! Например, вы можете определить количество запросов, полученных сервером, с помощью следующего предложения:

```
SHOW STATUS LIKE 'Questions'
```

Однако предложение само по себе является запросом, так что при каждом его выполнении изменяется значение `Questions`. Получается, что ваши средства оценки производительности сами портят собственные измерения, и нельзя это не учитывать.

9.17. Определение типов таблиц, поддерживаемых сервером

Задача

Вы хотите понять, можно ли создать таблицу указанного типа.

Решение

Спросите у сервера, какие типы таблиц он поддерживает.

Обсуждение

Предложение `SHOW VARIABLES` может сообщить вам о том, доступен ли определенный тип таблиц. Например, вы можете проверить переменные `have_bdb`, `have_innodb` и `have_gemini`, чтобы посмотреть, поддерживает ли сервер безопасные для транзакций таблицы. Код на PHP показывает, как проверить переменную `have_bdb` на значение `YES`, используя двухэтапный подход. Сначала убедимся в том, что переменная существует, проверив результирующее множество на непустоту. (Переменной вообще не будет, если ваша версия MySQL

еще не включает поддержку таблиц данного типа.) Затем извлечем значение переменной, чтобы проверить, равно ли оно YES:

```
$avail = FALSE;
# корректно экранируем имя переменной
$var_name = ereg_replace ("([%_])", "\\\\1", "have_bdb");
if ($result_id = mysql_query ("SHOW VARIABLES LIKE '$var_name'", $conn_id))
{
    if ($row = mysql_fetch_row ($result_id))
        $avail = ($row[1] == "YES" ? TRUE : FALSE);
    mysql_free_result ($result_id);
}
```

После выполнения кода $avail будет «истиной» или «ложью», показывая, поддерживает ли сервер таблицы BDB. Чтобы проверить поддержку другого типа таблиц, измените код так, чтобы тестировалась соответствующая серверная переменная.

Более общий подход заключается в создании функции, проверяющей все известные дескрипторы и возвращающей список поддерживаемых. Невозможно попросить такой список у сервера напрямую, но если вы знаете все возможные типы, то можете определить, какие из них поддерживаются, следуя таким правилам:

- До версии MySQL 3.23 были доступны только таблицы ISAM.
- Начиная с MySQL 3.23 всегда присутствует тип MyISAM. Могут поддерживаться и другие типы. Для типов ISAM, BDB и Gemini проверяйте серверные переменные have_isam, have_bdb и have_gemini соответственно (ищите значение YES). Доступность таблиц InnoDB определяется переменной have_innodb. Однако какое-то время эта переменная называлась have_innobase, так что проверьте лучше обе.

Эти правила реализуются функцией на Perl get_table_handlers(). Она возвращает список слов, извлеченных из списка bdb, gemini, innodb, isam и myisam, который соответствует поддерживаемым типам таблиц:

```
sub get_table_handlers
{
my $dbh = shift;
my @types;
my %typemap =
(
    "have_bdb"      => "bdb",
    "have_gemini"   => "gemini",
    "have_innodb"   => "innodb",
    "have_innobase" => "innodb",    # устаревшая форма have_innodb
    "have_isam"     => "isam"
);

    # получаем номер версии сервера
    my $ver_num = (get_server_version ($dbh))[1];   # числовой формат

    if ($ver_num < 32300)       # до версии 3.23.xx доступен только тип ISAM
```

```
    {
        @types = ("isam");
    }
    else
    {
        @types = ("myisam");    # тип MyISAM всегда  доступен после 3.23.xx

        # создаем запрос SHOW VARIABLES для получения серверных переменных 'have_',
        # указывающих на наличие обработчиков таблиц.  (Некоторые переменные 'have_'
        # не относятся к типам таблиц, но все же эффективнее выдать один запрос,
        # чем отдельный запрос для каждой переменной.)

        my $sth = $dbh->prepare ("SHOW VARIABLES LIKE 'have\\_%'");
        $sth->execute ();
        while (my ($var, $val) = $sth->fetchrow_array ())
        {
            push (@types, $typemap{$var})
                if exists ($typemap{$var}) && $val eq "YES";
        }
    }
    return (sort (@types));
}
```

10

Импорт и экспорт данных

10.0. Введение

Предположим, у вас есть файл в формате CSV с именем *somedata.csv*, содержащий 12 столбцов значений, разделенных запятыми. Вы хотите извлечь данные из столбцов 2, 11, 5, 9 и сохранить их в таблице MySQL, включающей столбцы `name`, `birth`, `height` и `weight` (имя, дата рождения, рост, вес). При этом необходимо убедиться в том, что рост и вес представлены целыми положительными числами и преобразовать дату из формата *MM/DD/YY* в формат *CCYY-MM-DD*. Как это можно сделать?

С одной стороны, задача достаточно специфичная. Но с другой – она вполне типична, так как вопросы переноса данных с различными условиями часто возникают при загрузке информации в MySQL. Было бы очень хорошо, если бы файлы с данными всегда были аккуратно отформатированы и подготовлены к загрузке в MySQL, но, как правило, это не так. В результате часто приходится предварительно обрабатывать информацию, чтобы привести ее к виду, приемлемому для MySQL. Обратное тоже верно: данные, выгруженные из MySQL, могут нуждаться в обработке перед использованием в других программах.

Иногда операции при переносе данных бывают настолько сложны, что требуют большой ручной работы по проверке и форматированию, но в большинстве случаев удается автоматизировать хотя бы часть работы. Практически все проблемы, возникающие при портировании, в том или ином виде включают в себя вопросы преобразования данных. В этой главе обсуждается, что же представляют из себя эти вопросы, как их решить, пользуясь имеющимся инструментарием, и как, при необходимости, создать собственные инструменты. Наша цель не в том, чтобы охватить все возможные случаи импорта и экспорта (это просто невозможно), а в том, чтобы показать некоторые типичные приемы и средства. Вы сможете использовать их «как есть» или адаптировать к своим потребностям. (Существуют также коммерческие средства преобразования данных, которые могут вам пригодиться, но моя цель – помочь вам справиться самостоятельно.)

Эта глава начинается с рецептов, рассказывающих о собственных средствах импорта данных в MySQL (предложение `LOAD DATA` и утилита командной строки *mysqlimport*) и экспорта (предложение `SELECT ... INTO OUTFILE` и программа *mysqldump*). Этих возможностей часто бывает достаточно в случаях, не требующих проверки данных или их переформатирования.

Затем рассматриваются способы применения сторонних программ и создания собственных – для тех случаев, когда родных средств импорта и экспорта MySQL оказывается недостаточно. До некоторого момента вы можете избегать написания собственных утилит, пользуясь готовыми программами. Например, *cut* может извлекать столбцы из файла, а *sed* и *tr* можно использовать как постпроцессоры для преобразования вывода в требуемый формат. Но вполне вероятно, что наступит момент, когда вы решите написать собственные программы. В таком случае вам придется рассмотреть два больших набора вопросов:

- Способы манипулирования структурами данных. Если файл имеет формат, не подходящий для импорта, вам придется преобразовать его в другой формат. Здесь может потребоваться замена разделителей, символов конца строки, удаление или перестановка столбцов файла.
- Способы манипулирования содержимым файлов данных. Если вы не уверены в корректности содержащихся в файле данных, то, вероятно, захотите выполнить их предварительную проверку или форматирование. Числовые значения могут потребовать проверки на принадлежность заданному диапазону, даты, возможно, придется преобразовать в формат ISO и т. п.

Исходные тексты программных фрагментов и сценариев, обсуждаемых в этой главе, находятся в каталоге *transfer* дистрибутива `recipes`, за исключением некоторых вспомогательных функций, расположенных в библиотеках в каталоге *lib*. Код некоторых коротких утилит приведен полностью. Для более крупных программ обычно рассматриваются только принципы их работы и использования, но вы всегда можете обратиться к исходным текстам, если хотите более детально разобраться в том, как они написаны.

Задачи, рассматриваемые в данной главе, требуют большого объема текстовых вычислений и сравнений с образцом. Perl силен именно в этой области, поэтому приводимые утилиты и фрагменты программ написаны в основном на этом языке. PHP и Python также обладают возможностями поиска по образцу, поэтому большинство рассмотренных примеров можно выполнить на них. Если вы хотите перенести предложенные решения на Java, потребуется библиотека, поддерживающая поиск по образцу на основе регулярных выражений. В приложении A приведены некоторые рекомендации.

Основные вопросы импорта и экспорта

При обмене данными между различными программами наибольшие трудности вызывают несовместимость форматов данных и различия в правилах интерпретации значений. Тем не менее, есть и другие общие проблемы. Зная о них, вы быстрее поймете, что именно необходимо для решения вопросов с импортом и экспортом, возникающих в каждом конкретном случае.

Основой входного потока (stream) является последовательность байтов, не имеющих какого-либо специального значения. Процесс загрузки данных в MySQL предполагает умение отличать те из них, которые отвечают за структурирование информации, от байтов, содержащих значения данных, организованных в эти структуры. Основные вопросы, ответ на которые позволит выполнить такое распознавание и выделить необходимые фрагменты данных, таковы:

- Чем разделяются записи? Ответ на этот вопрос позволит выделить во входном потоке отдельные записи.
- Чем разделяются поля? Ответ на этот вопрос позволит разделить каждую запись на поля, содержащие значения. Получение фактических значений может также потребовать удаления окружающих значение кавычек или распознания символов экранирования.

Способность разделять вводимые данные на записи и поля, несомненно, важна, но и сами значения могут иметь формат, непригодный для их непосредственного использования, поэтому вам придется ответить еще и на такие вопросы:

- Соответствуют ли порядок и количество столбцов структуре таблицы в базе данных? В случае расхождений может потребоваться упорядочивание или удаление столбцов.
- Надо ли переформатировать данные и проверять их правильность? Если формат данных соответствует тому, который ожидает получить MySQL, обработка необязательна. В противном случае требуется проверить их и, возможно, преобразовать.
- Как будут обрабатываться пустые и неопределенные (`NULL`) значения? Допустимы ли они? Есть ли возможность распознать `NULL`? (Некоторые системы при экспорте представляют `NULL` пустой строкой, так что их невозможно отличить.)

При экспорте из MySQL придется решать обратные задачи. Можно предположить, что в базе данных хранятся корректные данные, но они могут потребовать форматирования, кроме того, чтобы сформировать выходной поток, пригодный для распознавания в другой программе, необходимо позаботиться о разделителях полей и записей.

Эта глава рассматривает перечисленные выше вопросы на примерах переноса данных целыми файлами, но большинство обсуждаемых методик применимо и в других ситуациях. Рассмотрим веб-приложение, которое предлагает пользователю форму для заполнения, обрабатывает ее содержимое и создает запись в базе данных. Это соответствует импорту данных. Веб-ориентированные API обычно передают содержимое формы в виде набора отдельных значений, поэтому приложению, возможно, не придется заботиться о разделителях полей и записей. Зато проверка достоверности данных имеет первостепенное значение. Вы не можете знать заранее, какие данные пользователь отправит в ваш сценарий, поэтому проверка необходима.

Форматы файлов

Из множества встречающихся форматов файлов данных в этой главе чаще всего используются следующие:

- Файл, элементы данных в котором разделяются символом табуляции. Это одна из простейших структур: строки содержат значения, разделенные знаками табуляции. Короткий файл с такими разделителями может выглядеть приблизительно так (промежутки между столбцами представляют собой символы табуляции):

```
a       b       c
a,b,c   d e     f
```

- Формат CSV (Comma-separated values – значения, разделенные запятыми). Этот формат не стандартизован, поэтому использующие его файлы могут в чем-то различаться. Главное его правило в том, что строки содержат значения, отделенные друг от друга запятыми, а те значения, которые содержат запятые внутри себя, заключаются в кавычки во избежание интерпретации этих запятых как разделителей. Значения, содержащие пробелы, тоже обычно помещаются в кавычки. Вот пример, в котором каждая строка содержит три значения:

```
a,b,c
"a,b,c","d e",f
```

  Обрабатывать CSV-файл сложнее, чем файл с табуляциями, в силу того, что символы кавычек и запятых имеют двойное значение: они могут как определять структуру файла, так и быть частью данных.

Еще одна важная характеристика файла данных – это последовательность символов окончания строки. Чаще всего используются «возврат каретки», «перевод строки» и их комбинация, обычно обозначаемые аббревиатурами CR (carriage return), LF (linefeed) и CRLF.

Файлы данных часто начинаются со строки заголовков столбцов. Кстати, CSV-файл, начинающийся со строки имен – это то, что в FileMaker Pro называется файлом в формате слияния (merge). Иногда строка заголовков мешает при импорте, так как ее приходится отбрасывать, чтобы не загрузить имена столбцов в таблицу в качестве данных. Но в остальных случаях заголовки бывают весьма полезны:

- При импорте в существующую таблицу заголовки позволяют установить соответствие между столбцами файла и таблицы, если их порядок отличается.
- При автоматическом и полуавтоматическом создании таблиц на основе файлов заголовки могут использоваться в качестве имен столбцов. Например, в рецепте 10.36 описывается утилита, которая читает файл данных и предлагает предложение `CREATE TABLE` для создания соответствующей таблицы. Если строка заголовков присутствует, утилита использует их для именования столбцов. В противном случае ей приходится использовать менее содержательные имена вида `c1`, `c2` и т. д.

Символы табуляции и перевода строки в качестве разделителей

Несмотря на то, что встречаются файлы данных самых различных форматов, вряд ли вы захотите в каждую из разрабатываемых утилит включать по несколько различных алгоритмов. Мне тоже этого не хотелось, поэтому в большинстве описанных в этой главе программ для простоты предполагается, что входные файлы используют в качестве разделителей символы табуляции, а конец строки обозначатся символом перевода строки. (Этот формат MySQL использует по умолчанию в предложении `LOAD DATA`.) Такое допущение заметно облегчает создание программ чтения файлов.

С другой стороны, надо как-то читать файлы и в других форматах. Для решения этой проблемы напишем сценарий *cvt_file.pl*, способный читать и писать в нескольких различных форматах (см. рецепт 10.18). Этот сценарий основан на модуле Perl *Text::CSV_XS*, который, вопреки названию, может обрабатывать не только CSV-формат. Утилита *cvt_file.pl* способна конвертировать файлы разных типов, благодаря чему с ними могут работать и программы, рассчитывающие лишь на символ табуляции в качестве разделителя.

Замечания о запуске команд оболочки

В этой главе рассматривается ряд программ, вызываемых из командной строки оболочки – такой, как *bash* или *tcsh* в UNIX или *CMD.EXE* («приглашение DOS») в Windows. Во многих примерах использования этих программ значения опций заключены в кавычки, причем символ кавычки иногда тоже может быть значением опции. В разных оболочках приняты разные соглашения об использовании кавычек, но в большинстве из них (включая *CMD.EXE* в Windows) выполняются следующие правила:

- Аргумент, содержащий пробелы, заключается в двойные кавычки, чтобы оболочка не интерпретировала его как несколько отдельных аргументов. Оболочка отбросит кавычки и передаст такой аргумент команде целиком.
- Символ двойной кавычки, расположенный внутри значения аргумента, предваряется обратным слэшем (\).

Некоторые команды оболочки имеют большую длину и располагаются на нескольких строках, в которых обратный слэш используется как знак продолжения:

```
% имя_программы \
    аргумент1 \
    аргумент2 ...
```

Это работает в UNIX и не работает в Windows, где знак продолжения следует удалить и вводить всю команду в одной строке:

```
C:\> имя_программы аргумент1 аргумент2 ...
```

10.1. Импорт с помощью LOAD DATA и утилиты mysqlimport

Задача

Вы хотите загрузить данные из файла в таблицу, используя встроенные средства импорта MySQL.

Решение

Используйте предложение `LOAD DATA` или утилиту командной строки *mysqlimport*.

Обсуждение

Предложение `LOAD DATA` используется в MySQL для загрузки массивов данных. Вот пример этого предложения, который читает файл *mytbl.txt* из текущего каталога и сохраняет данные в таблице `mytbl` текущей базы данных:

```
mysql> LOAD DATA LOCAL INFILE 'mytbl.txt' INTO TABLE mytbl;
```

Кроме того, в MySQL имеется утилита *mysqlimport*, выступающая в роли оболочки для `LOAD DATA`, что позволяет загружать файлы прямо из командной строки. Если предположить, что таблица `mytbl` находится в базе данных `cookbook`, то команда *mysqlimport*, эквивалентная приведенному выше предложению `LOAD DATA`, выглядит так:[1]

```
% mysqlimport --local cookbook mytbl.txt
```

Большинство из перечисленных ниже характеристик предложения `LOAD DATA` относятся и к *mysqlimport*. Есть несколько различий, которых мы коснемся по мере изложения, но в большинстве случаев там, где написано «`LOAD DATA`», можно читать «`LOAD DATA` или *mysqlimport*». Предложение `LOAD DATA` имеет опции, соответствующие таким рассмотренным выше параметрам импорта, как символы конца строки для разбиения данных на записи, разделитель столбцов для выделения отдельных значений, символы для заключения значений столбцов в кавычки, экранирования кавычек внутри значений и представления неопределенных значений:

- По умолчанию `LOAD DATA` предполагает, что файл данных содержит то же количество столбцов, что и таблица, в которую загружаются данные, и эти столбцы находятся в том же порядке. Если в файле отсутствуют значения некоторых столбцов или их порядок отличается от принятого в таблице, то вы можете указать, какие столбцы есть в наличии и в каком порядке. Если в файле меньше столбцов, чем в таблице, то MySQL записывает значения по умолчанию в те столбцы, для которых нет данных.

---

[1] Для *mysqlimport*, как и для других программ MySQL, вы можете указать параметры соединения в виде *--user* или *--host*. Эти параметры должны предшествовать параметру, задающему имя базы данных.

- LOAD DATA исходит из предположения, что значения разделены символами табуляции и что строки заканчиваются символом LF (перевод строки). Если файл не удовлетворяет этим условиям, вы можете явно указать формат данных.
- Вы можете указать на необходимость удаления кавычек, окружающих значения, и определить символ, используемый в качестве кавычки.
- В процессе ввода распознаются и преобразовываются некоторые специальные экранирующие последовательности. По умолчанию в качестве экранирующего символа используется обратный слэш (\), но при желании это можно изменить. Комбинация \N используется для представления значения NULL. Последовательности \b, \n, \r, \t, \\ и \0 интерпретируются как символы забоя, перевода строки, возврата каретки, табуляции, обратного слэша и ASCII-ноля (NUL) соответственно. (Значение NUL – это байт, содержащий нулевое значение (это не то же самое, что значение NULL в SQL).
- Предложение LOAD DATA выводит диагностическую информацию, но в обобщенном виде, не дающем представления о том, в каких строках может находиться источник проблем. В MySQL 4 предполагается расширить возможности диагностики. Пока же можете пользоваться рецептом 10.37, в котором описана диагностическая утилита для LOAD DATA.

Следующие несколько разделов рассказывают, как импортировать в MySQL файлы данных при помощи LOAD DATA или *mysqlimport*. Предполагается, что значения данных корректны и могут быть использованы в MySQL. Для чего сделано это допущение? Дело в том, что хотя у LOAD DATA и есть некоторые средства контроля входных данных, они ориентированы в основном на проверку структуры файла. LOAD DATA не проверяет правильность значений и не выполняет преобразований. Эти операции придется выполнять либо до загрузки – над файлом данных, либо после – средствами SQL. Если вам надо выполнить предварительную проверку или преобразование входного файла, чтобы убедиться в его правильности, то в этом вам помогут некоторые рецепты этой главы.

10.2. Определение местоположения файла данных

Задача

Вы не уверены в том, как указать в предложении LOAD DATA местоположение файла данных, особенно если он находится в другом каталоге.

Решение

Надо запомнить правила, которыми MySQL руководствуется при поиске файлов.

Обсуждение

Сервер MySQL, приступая к выполнению предложения LOAD DATA, обычно предполагает, что файл с данными находится на том же хосте. Однако при таком подходе могут возникнуть определенные трудности:

- Если вы обращаетесь к серверу MySQL с клиента на удаленном хосте и не имеете возможности передать файл данных на серверный хост (например, у вас нет учетной записи для входа туда).
- Даже если у вас есть учетная запись, позволяющая зайти на серверный хост, учетная запись MySQL должна иметь привилегию FILE, а сам загружаемый файл должен либо иметь доступ на чтение для всех, либо располагаться в каталоге данных текущей базы данных. Большинство пользователей MySQL не имеют привилегии FILE (так как она позволяет выполнять опасные действия), кроме того вы вряд ли захотите открывать свой файл на чтение для всех (по соображениям безопасности), или же каталог данных может оказаться для вас недоступным.

К счастью, если вы работаете с MySQL версии 3.22.15 и выше, то можете загружать локальные файлы, расположенные на клиентском хосте, используя предложение LOAD DATA LOCAL вместо LOAD DATA. Единственная привилегия, необходимая для импорта локального файла – это доступ к нему на чтение.[1]

Если ключевое слово LOCAL отсутствует, то MySQL ищет файл данных на серверном хосте по следующим правилам:

- Абсолютное путевое имя полностью определяет положение файла в файловой системе. MySQL читает файл из указанного места.
- Относительное путевое имя интерпретируется двумя способами, в зависимости от того, состоит оно из одного компонента или из нескольких. Имя из одного компонента, например *mytbl.txt*, MySQL ищет в каталоге текущей базы данных. В случае многокомпонентных имен, таких как *xyz/mytbl.txt*, MySQL ищет файл в соответствующем подкаталоге каталога данных *data*. (Предполагается, что файл *mytbl.txt* находится в каталоге *xyz*.)

Каталоги баз данных располагаются непосредственно в каталоге данных (*data*), поэтому следующие предложения эквиваленты – в предположении, что текущей базой данных является cookbook:

```
mysql> LOAD DATA INFILE 'mytbl.txt' INTO TABLE mytbl;
mysql> LOAD DATA INFILE 'cookbook/mytbl.txt' INTO TABLE mytbl;
```

Если присутствует ключевое слово LOCAL, то MySQL ищет файл на клиентском хосте, интерпретируя имя по тем же правилам, что и командный интерпретатор:

---

[1] В MySQL 3.23.49 по умолчанию запрещено использование ключевого слова LOCAL. Возможно, его удастся разрешить при помощи опции *--local-infile* команды *mysql*. Если это не сработало, значит в конфигурации вашего сервера запрещено использование LOAD DATA LOCAL.

- Абсолютное путевое имя полностью определяет положение файла относительно корня файловой системы.
- Относительное путевое имя интерпретируется относительно текущего каталога.

Если файл расположен на клиентском хосте, но вы забыли указать, что он локальный, то вы получите сообщение об ошибке.

```
mysql> LOAD DATA 'mytbl.txt' INTO TABLE mytbl;
ERROR 1045: Access denied for user: 'cbuser@localhost' (Using password: YES)
```

Такое сообщение `Access denied` может сбивать с толку: ведь то, что вы смогли установить соединение с сервером и дать команду `LOAD DATA`, означает, что у вас есть доступ к MySQL. На самом деле такое сообщение об ошибке говорит о том, что сервер MySQL пытался открыть на своем хосте файл *mytbl.txt* и не смог этого сделать.

Утилита *mysqlimport* при поиске файлов придерживается тех же правил, что и `LOAD DATA`. По умолчанию она предполагает, что файл данных расположен на серверном хосте. Чтобы открыть локальный файл, используйте опцию *--local* (или *-L*) в командной строке.

Если база данных не указана явно, то `LOAD DATA` предполагает, что таблица находится в текущей базе данных. Утилита *mysqlimport* всегда требует указания базы данных:

```
% mysqlimport --local cookbook mytbl.txt
```

Если вы загружаете данные при помощи `LOAD DATA` в базу данных, отличную от текущей, то можете добавить имя базы к имени таблицы. В следующем предложении указано, что таблица `mytbl` находится в базе данных `other_db`:

```
mysql> LOAD DATA LOCAL 'mytbl.txt' INTO TABLE other_db.mytbl;
```

Команда `LOAD DATA` не предполагает связи между именами файла данных и таблицы, в которую эти данные загружаются. Утилита *mysqlimport* устанавливает однозначное соответствие между именем файла и именем таблицы. А именно, она использует последний компонент имени файла в качестве имени таблицы. Например, файлы *mytbl.txt*, *mytbl.dat*, */tmp/mytbl.txt*, */u/paul/*

Имена файлов данных в Windows

ОС Windows использует \ как разделитель в путевых именах. Это создает небольшую проблему, так как MySQL интерпретирует обратный слэш в строковых значениях как символ экранирования. Для указания пути в Windows используйте либо двойной обратный слэш, либо прямой слэш. Следующие предложения иллюстрируют два способа указания одного и того же пути в Windows:

```
mysql> LOAD DATA LOCAL INFILE 'D:\\projects\\mydata.txt' INTO mytbl;
mysql> LOAD DATA LOCAL INFILE 'D:/projects/mydata.txt' INTO mytbl;
```

data/mytbl.csv и *D:\projects\mytbl.txt* будут рассматриваться этой утилитой как источники данных для таблицы `mytbl`.

10.3. Указание формата файла данных

Задача

Формат вашего файла данных отличается от используемого в предложении `LOAD DATA` по умолчанию.

Решение

Используйте инструкции `FIELDS` и `LINES`, сообщающие предложению `LOAD DATA`, как следует интерпретировать файл.

Обсуждение

По умолчанию `LOAD DATA` полагает, что строки файла данных разделяются символом перевода строки LF, а значения данных в строке разделены символами табуляции. В следующем предложении нет никаких указаний на формат файла данных, поэтому MySQL принимает формат по умолчанию:

```
mysql> LOAD DATA LOCAL INFILE 'mytbl.txt' INTO TABLE mytbl;
```

Для явного указания формата используйте инструкцию `FIELDS`, определяющую характеристики полей в строке, и инструкцию `LINES`, устанавливающую разделители строк. В приведенном ниже предложении `LOAD DATA` указано, что значения в файле данных разделяются двоеточием, а строки завершаются символом возврата каретки:

```
mysql> LOAD DATA LOCAL INFILE 'mytbl.txt' INTO TABLE mytbl
    -> FIELDS TERMINATED BY ':'
    -> LINES TERMINATED BY '\r';
```

Обе инструкции следуют за именем таблицы. Если присутствуют обе инструкции, то первой должна быть `FIELDS`. Разделители полей и строк могут состоять из нескольких символов. Например, `\r\n` означает, что строки разделяются последовательностью CRLF.

При использовании программы *mysqlimport* формат указывается в командной строке. Команды, соответствующие двум приведенным выше предложениям `LOAD DATA`, выглядят так:

```
% mysqlimport --local cookbook mytbl.txt
% mysqlimport --local --fields-terminated-by=":" --lines-terminated-by="\r" \
    cookbook mytbl.txt
```

Опции формата *mysqlimport* могут следовать в любом прядке, но обязательно должны предшествовать имени базы данных.

Определение двоичных значений опций формата

Начиная с версии MySQL 3.22.10 можно использовать шестнадцатеричную нотацию для задания произвольных значений формата в инструкциях FIELDS и LINES. Предположим, файл данных состоит из строк, заканчивающихся символом Ctrl-B, а разделителем полей служит символ Ctrl-A. ASCII-коды для Ctrl-A и Ctrl-B – это 1 и 2 соответственно, поэтому они могут быть записаны как 0x01 и 0x02:

```
FIELDS TERMINATED BY 0x01 LINES TERMINATED BY 0x02
```

Утилита *mysqlimport* начала понимать шестнадцатеричные значения в спецификации формата, начиная с версии MySQL 3.23.30. Эта возможность может пригодиться, если вы не хотите запоминать, как вводятся экранирующие последовательности в командной строке и когда они должны быть заключены в кавычки. Символ табуляции имеет код 0x09, перевода строки – 0x0a, возврата каретки – 0x0d. В приведенном примере поля разделены символом табуляции, а строки заканчиваются последовательностью CRLF:

```
% mysqlimport --local --lines-terminated-by=0x0d0a \
    --fields-terminated-by=0x09 cookbook mytbl.txt
```

10.4. Использование кавычек и специальных символов

Задача

Ваш файл данных содержит значения, заключенные в кавычки, или экранированные символы.

Решение

В предложении LOAD DATA укажите, что определенные символы не должны загружаться в базу данных необработанными.

Обсуждение

Кроме опции TERMINATED BY в инструкции FIELDS могут присутствовать и другие спецификаторы формата. По умолчанию LOAD DATA считает, что значения не заключены в кавычки, и интерпретирует обратный слэш (\) как экранирующий символ для специальных символов. Для явного определения символов, исполняющих роль кавычек, используйте опцию ENCLOSED BY; MySQL в процессе ввода отбросит указанные символы, если встретит их в начале или в конце значения данных. Для изменения значения экранирующего символа используйте опцию ESCAPED BY.

Эти три опции инструкции FIELDS (ENCLOSED BY, ESCAPED BY и TERMINATED BY) могут следовать в любом порядке, если вы используете более, чем одну из них. Например, следующие инструкции FIELDS эквивалентны:

```
FIELDS TERMINATED BY ',' ENCLOSED BY '"'
FIELDS ENCLOSED BY '"' TERMINATED BY ','
```

Значение, определяемое опцией TERMINATED BY, может состоять из нескольких символов. Если значения данных внутри строк разделены чем-то вроде *@*, вы можете написать так:

```
FIELDS TERMINATED BY '*@*'
```

Для полной отмены экранирования укажите пустое значение экранирующей последовательности:

```
FIELDS ESCAPED BY ''
```

Если чс помощью ENCLOSED BY вы указываете, что значения данных должны быть очищены от ограничивающих символов, то для включения их литеральных значений в данные их нужно продублировать или предварить экранирующим символом. Например, если в качестве кавычек и экранирующего символа используются ¨ и \, исходное значение ¨a¨"b\¨c¨ будет интерпретировано как a¨b¨c.

Для утилиты *mysqlimport* соответствующие параметры командной строки для определения кавычек и экранирования называются *--fields-enclosed-by* и *--fields-escaped-by*. (Используя опции *mysqlimport*, содержащие кавычки и обратный слэш, которые могут иметь специальное значение в командном интерпретаторе, помните о необходимости помещать в кавычки или экранировать кавычки и символы экранирования!)

10.5. Импорт файлов в формате CSV

Задача

Вам надо загрузить файл, имеющий формат CSV.

Решение

Просто укажите соответствующие спецификаторы формата в предложении LOAD DATA.

Обсуждение

Файлы в формате CSV содержат значения, разделенные запятыми вместо табуляций, которые могут быть заключены в двойные кавычки. Например, CSV-файл *mytbl.txt*, строки которого заканчиваются символами CRLF, может быть загружен в таблицу mytbl таким предложением LOAD DATA:

```
mysql> LOAD DATA LOCAL INFILE 'mytbl.txt' INTO TABLE mytbl
    -> FIELDS TERMINATED BY ',' ENCLOSED BY '"'
    -> LINES TERMINATED BY '\r\n';
```

Или так, при помощи *mysqlimport*:

```
% mysqlimport --local --lines-terminated-by="\r\n" \
    --fields-terminated-by="," --fields-enclosed-by="\"" \
    cookbook mytbl.txt
```

10.6. Чтение файлов, полученных из разных операционных систем

Задача

В разных операционных системах используются разные последовательности для обозначения конца строки.

Решение

Именно поэтому в предложении LOAD DATA имеется инструкция LINES TERMINATED BY.

Обсуждение

Как правило, последовательность, задающая конец строки в файле, определяется той системой, в которой файл был создан, а не той, в которой он импортируется. Имейте это в виду, когда загружаете файл, полученный из другой операционной системы.

В UNIX строки обычно заканчиваются символом конца строки, который в предложении LOAD DATA можно задать так:

```
LINES TERMINATED BY '\n'
```

Однако благодаря тому что \n используется в LOAD DATA по умолчанию, инструкция LINES TERMINATED BY не обязательна, если только вам не требуется явно указать значение разделителя строк.

Файлы, создаваемые в операционных системах Mac OS или Windows, обычно содержат строки, заканчивающиеся символом возврата каретки или парой CRLF. Для работы с такими разделителями используйте соответствующую инструкцию LINES TERMINATED BY:

```
LINES TERMINATED BY '\r'
LINES TERMINATED BY '\r\n'
```

Например, для загрузки файла из Windows, содержащего поля, разделенные табуляциями, и строки, заканчивающиеся парой CRLF, используйте такое предложение LOAD DATA:

```
mysql> LOAD DATA LOCAL INFILE 'mytbl.txt' INTO TABLE mytbl
    -> LINES TERMINATED BY '\r\n';
```

Соответствующий вызов *mysqlimport* имеет такой вид:

```
% mysqlimport --local --lines-terminated-by="\r\n" cookbook mytbl.txt
```

10.7. Обработка дубликатов индексированных записей

Задача

Записи в ваших входных данных дублируют значения уникальных ключей уже существующих записей таблицы.

Решение

В предложении LOAD DATA укажите, будут ли новые записи игнорироваться или замещать старые.

Обсуждение

По умолчанию при попытке загрузить запись, дублирующую значения столбцов, составляющих первичный ключ или уникальный индекс, генерируется ошибка. Для изменения этой реакции используйте ключевое слово IGNORE или REPLACE после имени файла, чтобы сообщить MySQL, как поступать с дублирующими записями: игнорировать их или заменять ими старые записи.

Предположим, что вы периодически получаете метеорологические данные о текущем состоянии погоды от различных станций наблюдения и храните результаты измерений различных параметров в такой таблице:

```
CREATE TABLE weatherdata
(
    station INT UNSIGNED NOT NULL,
    type    ENUM('precip','temp','cloudiness','humidity','barometer') NOT NULL,
    value   FLOAT,
    UNIQUE (station, type)
);
```

Для того чтобы гарантировать, что сохраняется только одна запись для каждого типа измерений от каждой станции, в таблице имеется уникальный ключ на сочетание идентификатора станции и типа измерения. Таблица предназначена для хранения только текущих значений, поэтому когда для данной станции загружается новое значение измерений, предыдущее значение должно исчезнуть. В таких случаях применяется ключевое слово REPLACE:

```
mysql> LOAD DATA LOCAL INFILE 'data.txt' REPLACE INTO TABLE weatherdata;
```

10.8. Расширенная диагностика в LOAD DATA

Задача

Предложение LOAD DATA выдает слишком мало информации о проблемах в файле данных.

Решение

Тут уж ничего не поделаешь. Хотя давайте попробуем.

Обсуждение

Предложение `LOAD DATA` по окончании выполнения выдает строку со сведениями о количестве ошибок и проблемных ситуаций. Предположим, вы загрузили файл в таблицу и получили такое сообщение:

```
Records: 134  Deleted: 0  Skipped: 2  Warnings: 13
```

В этих значениях содержится некоторая общая информация об операции импорта:

- `Records` сообщает количество записей, обнаруженных в файле.
- `Deleted` и `Skipped` относятся к действиям над записями, дублирующими уникальные значения уже существующих записей. `Deleted` сообщает, сколько записей было удалено из таблицы и заменено новыми, а `Skipped` указывает, сколько входных записей было отброшено из-за наличия уже существующих.
- `Warnings` – что-то вроде счетчика ошибочных ситуаций, возникших в процессе загрузки: правильно или нет записано значение в столбец. Если нет, значит в MySQL попало какое-то другое значение, и счетчик предупреждений увеличивается. (Например, попытка записи строки `abc` в числовой столбец приводит к сохранению в нем значения `0`.)

О чем говорят эти числа? Значение `Records`, как правило, должно совпадать с количеством строк входного файла. Несоответствие означает, что MySQL неправильно распознает формат файла. В этом случае значение счетчика `Warnings`, скорее всего, тоже будет весьма большим, так как несоответствие типов данных приводит к их конвертированию. (Решение этой проблемы часто заключается в использовании соответствующих инструкций `FIELDS` и `LINES`.) В противном случае эти числа мало что могут сказать. По ним нельзя определить, какие записи вызвали проблемы и в каких столбцах произошли ошибки. В MySQL версии 4 предприняты некоторые усилия по увеличению информативности сообщений. Пока же можно пользоваться сценарием из рецепта 10.37, проверяющим файл данных на наличие сомнительных значений.

10.9. Не преувеличивайте возможности LOAD DATA

Задача

Вам кажется, что `LOAD DATA` умнее, чем есть на самом деле.

Решение

Не думайте, что `LOAD DATA` знает все о формате вашего файла данных. И удостоверьтесь в том, что сами знаете, каков этот формат. Если файл передавал-

ся с одной машины на другую, его содержимое могло слегка измениться, а вы могли этого не заметить.

Обсуждение

Часто разочарование от использования `LOAD DATA` наступает из-за того, что люди слишком преувеличивают возможности MySQL. Предложение `LOAD DATA` делает ряд допущений о структуре входного файла, которые заключаются в установленных по умолчанию разделителях строк и полей, кавычках и экранирующих символах. Если вводимые данные не соответствуют этим допущениям, вы должны сообщить об этом MySQL.

Если у вас есть сомнения, проверьте содержимое файла с помощью шестнадцатеричного дампа или какой-либо утилиты, явно отображающей такие пробельные символы, как табуляция, возврат каретки и конец строки. В UNIX можно воспользоваться программой *od*, способной отображать содержимое файла в различных форматах. При отсутствии *od* и других подобных утилит, загляните в каталог *transfer* дистрибутива `recipes`, в котором вы найдете программы вывода шестнадцатеричного дампа на Perl и Python (*hexdump.pl* и *hexdump.py*), а также пару программ, визуализирующих все символы файла (*see.pl* и *see.py*). Они могут пригодиться для выяснения того, какие символы в действительности содержит ваш файл. Вас может удивить, насколько содержимое файла иногда отличается от ваших ожиданий. В действительности, это довольно часто случается при передаче файла с одной машины на другую:

- При передаче файлов по FTP в текстовом (не двоичном) режиме между разными операционными системами обычно выполняется преобразование разделителей строк в формат, принятый принимающей стороной. Предположим, у вас есть файл, в котором поля и записи разделены символами TAB и LF соответственно, и который прекрасно загружается под UNIX предложением `LOAD DATA` с установками по умолчанию. Если вы передадите этот файл по FTP в текстовом режиме на Windows-машину, то вполне возможно, что символы LF будут заменены парами CRLF. На этой машине файл не сможет загрузиться корректно тем же самым предложением `LOAD DATA`, так как его содержимое претерпело изменения. Может ли MySQL знать об этом? Нет. Так что это ваша обязанность – добавить инструкцию `LINES TERMINATED BY '\r\n'` в команду загрузки. К такому результату может привести передача файлов между любыми двумя системами с разными соглашениями о разделителе строк. К примеру, в файлах Macintosh, содержащих символ возврата каретки CR, после их передачи в UNIX могут появиться символы конца строки LF. Вы должны либо отражать такие изменения в инструкции `LINES TERMINATED BY`, либо передавать файлы в двоичном режиме во избежание нежелательных преобразований.
- Файлы данных, вставленные в тело электронного письма, часто подвергаются изменениям. Почтовые программы могут свернуть (разбить на части) длинные строки или заменить разделители строк. По электронной почте файлы данных лучше отправлять присоединенными к письму, а не в теле письма.

10.10. Пропуск строк в файле данных

Задача

Вы хотите, чтобы предложение LOAD DATA пропустило одну или несколько строк в начале файла, прежде чем начать загрузку записей.

Решение

Сообщите предложению LOAD DATA, сколько строк надо пропустить.

Обсуждение

Чтобы пропустить первые *n* строк файла, добавьте инструкцию IGNORE *n* LINES в предложение LOAD DATA. Например, если файл с символами табуляции в качестве разделителей начинается со строки заголовков столбцов, то ее можно пропустить:

```
mysql> LOAD DATA LOCAL INFILE 'mytbl.txt' INTO TABLE mytbl IGNORE 1 LINES;
```

Начиная с версии MySQL 4.0.2, утилита *mysqlimport* поддерживает опцию *--ignore-lines=n*, имеющую тот же эффект.

Инструкция IGNORE часто бывает полезна для файлов, полученных из внешних источников. Например, FileMaker Pro может экспортировать файлы в так называемом формате слияния (merge), представляющем собой не что иное как формат CSV со строкой заголовков. Следующее предложение пропустит заголовки в таком файле, созданном FileMaker Pro в Mac OS (с символом CR в качестве разделителя строк):

```
mysql> LOAD DATA LOCAL INFILE 'mytbl.txt' INTO TABLE mytbl
    -> FIELDS TERMINATED BY ',' ENCLOSED BY '"'
    -> LINES TERMINATED BY '\r'
    -> IGNORE 1 LINES;
```

Заметим, что импорт файла FileMaker Pro часто не так прост, как здесь показано. Если, например, он содержит даты, то они могут оказаться в неподходящем для MySQL формате. Вам придется подготовить файл к загрузке или выполнить обработку после нее (см. рецепт 10.40).

10.11. Определение порядка ввода столбцов

Задача

Столбцы файла расположены не в том же порядке, что столбцы таблицы, в которую вы его загружаете.

Решение

Укажите предложению LOAD DATA, как совместить столбцы файла и таблицы, определив их взаимное соответствие.

Обсуждение

По умолчанию LOAD DATA считает, что столбцы файла и таблицы упорядочены одинаково. Если это не так, вы можете задать список, определяющий, в какой столбец таблицы должен быть загружен каждый столбец файла. Предположим, в таблице есть столбцы a, b и c, но в файле они располагаются в порядке b, c и a. Тогда файл можно загрузить так:

```
mysql> LOAD DATA LOCAL INFILE 'mytbl.txt' INTO TABLE mytbl (b, c, a);
```

В соответствующей команде *mysqlimport* для задания списка столбцов используется опция *--columns*:

```
% mysqlimport --local --columns=b,c,a cookbook mytbl.txt
```

Опция *--columns* в *mysqlimport* появилась в MySQL версии 3.23.17. Если у вас более старая версия, используйте предложение LOAD DATA или предварительно упорядочивайте столбцы в файле в соответствии с их расположением в таблице (для этой цели подходит утилита из рецепта 10.19).

10.12. Пропуск столбцов файла данных

Задача

Файл содержит столбцы, которые следует проигнорировать и не загружать в таблицу.

Решение

Никаких проблем, если эти столбцы – последние. В противном случае перед загрузкой потребуется предварительная обработка файла данных.

Обсуждение

Лишние столбцы, если они расположены последними, не создают трудностей. Если в строке файла больше полей, чем в записи таблицы, то LOAD DATA просто игнорирует их (хотя и может предупредить о наличии несоответствия).

Пропуск столбцов, расположенных в середине, потребует чуть больших усилий. Предположим, вы хотите загрузить информацию из файла паролей UNIX */etc/passwd*, который содержит строки такого вида:

```
account:password:UID:GID:GECOS:directory:shell
```

Предположим также, что вы не хотите связываться с загрузкой столбца паролей. Таблица для хранения данных из остальных столбцов выглядит так:

```
CREATE TABLE passwd
(
    account     CHAR(8),      # регистрационное имя
    uid         INT,          # ID пользователя
    gid         INT,          # ID группы
    gecos       CHAR(60),     # имя, телефон, офис, и т.п.
```

```
    directory   CHAR(60),     # домашний каталог
    shell       CHAR(60)      # командный интерпретатор
);
```

Чтобы загрузить файл, мы должны указать, что разделителем полей служит двоеточие, для этого воспользуемся инструкцией FIELDS:

```
FIELDS TERMINATED BY ':'
```

Но надо еще объяснить предложению LOAD DATA, что нужно пропустить второе поле, содержащее пароль. Это проблематично, так как LOAD DATA всегда загружает столбцы файла подряд. Вы можете определить соответствие столбцов файла столбцам таблицы, но пропустить столбцы в файле не можете. Чтобы справиться с проблемой, преобразуем входной файл в промежуточный, не содержащий столбец с паролями, и загрузим его. В UNIX для этого можно воспользоваться утилитой *cut*, которая извлечет нужные нам столбцы:

```
% cut -d":" -f0,3- /etc/passwd > passwd.txt
```

Опция *-d* устанавливает разделитель полей, а *-f* определяет, что надо извлечь первый столбец и все столбцы, начиная с третьего. В результате остается все, кроме второго столбца. (Запустите *man cut* для получения подробной справки по команде *cut*.) Теперь воспользуйтесь предложением LOAD DATA для импорта полученного файла *passwd.txt* в таблицу passwd, например так:

```
mysql> LOAD DATA LOCAL INFILE 'passwd.txt' INTO TABLE passwd
    -> FIELDS TERMINATED BY ':';
```

Соответствующая команда *mysqlimport* выглядит так:

```
% mysqlimport --local --fields-terminated-by=":" cookbook passwd.txt
```

См. также

Команда *cut* всегда выводит столбцы в том же порядке, что и в исходном файле, независимо от того, в каком порядке они перечислены в опции *-f*. (Например, *cut -f1,2,3* и *cut -f3,2,1* дадут одинаковый результат.) В рецепте 10.19 рассматривается утилита, которая может извлекать и отображать столбцы в произвольном порядке.

10.13. Экспорт результатов запроса из MySQL

Задача

Вы хотите экспортировать результаты запроса из MySQL в файл или в другую программу.

Решение

Воспользуйтесь предложением SELECT ... INTO OUTFILE или перенаправьте вывод программы *mysql*.

Обсуждение

В MySQL можно использовать предложение SELECT ... INTO OUTFILE для вывода результатов запроса непосредственно в файл на хосте сервера. Если же вы хотите получить результат на клиентском хосте, то есть другая возможность: перенаправить вывод программы *mysql*. У каждого из этих методов есть свои сильные и слабые стороны, поэтому вам следует познакомиться с обоими и использовать тот, который больше подходит в конкретной ситуации.

Экспорт с помощью SELECT ... INTO OUTFILE

Здесь используется обычное предложение SELECT, к которому добавлена конструкция INTO OUTFILE *имя_файла*. Формат вывода по умолчанию тот же, что и для LOAD DATA, так что следующее предложение экспортирует таблицу passwd в файл */tmp/passwd.txt* с символами TAB и LF в качестве разделителей полей и строк:

```
mysql> SELECT * FROM passwd INTO OUTFILE '/tmp/passwd.txt';
```

Для изменения формата вывода вы можете использовать опции, аналогичные тем, которые использовались в LOAD DATA для определения кавычек и разделителей столбцов и строк. Чтобы экспортировать таблицу passwd в формате CSV с разделением строк символами CRLF, используйте такое предложение:

```
mysql> SELECT * FROM passwd INTO OUTFILE '/tmp/passwd.txt'
    -> FIELDS TERMINATED BY ',' ENCLOSED BY '"'
    -> LINES TERMINATED BY '\r\n';
```

Предложение SELECT ... INTO OUTFILE обладает следующими свойствами:

- Выходной файл создается непосредственно сервером MySQL, поэтому имя файла должно соответствовать его желаемому расположению на серверном хосте. Не существует локальной версии этого предложения, аналогичной LOAD DATA LOCAL.
- Для выполнения предложения SELECT ... INTO необходима привилегия FILE в MySQL.
- Нельзя использовать уже существующий файл. Это предохраняет от потери файлов, возможно, содержащих важную информацию.
- Вам потребуется учетная запись на сервере или другой способ для получения файла с этого хоста. Иначе предложение SELECT ... INTO OUTFILE окажется для вас бесполезным.
- В UNIX файл создается доступным для чтения всем, а его владельцем является сервер MySQL. Это означает, что, хотя вы и сможете прочитать его, у вас может не оказаться прав на его удаление.

Экспорт с помощью клиента mysql

Вследствие того, что предложение SELECT ... INTO OUTFILE записывает файл на хост сервера, вы не сможете использовать его, если у вашей учетной записи в MySQL нет привилегии FILE. Для экспорта данных в локальный файл

требуется другой подход. Если все, что вам нужно – это вывод данных, разделенных символами табуляции, можно воспользоваться «экспортом для бедных», выполняя предложение SELECT из программы *mysql* с перенаправленным в файл выводом. При таком способе вам не потребуется привилегия FILE для вывода результата запроса в файл. Вот пример, в котором экспортируются столбцы регистрационного имени и командного интерпретатора из ранее созданной таблицы passwd:

```
% mysql -e "SELECT account, shell FROM passwd" -N cookbook > shells.txt
```

Опция *-e* определяет выполняемый запрос, а *-N* сообщает MySQL, что не надо выводить строку заголовков столбцов, обычно предваряющую вывод. Вторая опция была добавлена в MySQL 3.22.20; если у вас более старая версия, вы можете получить тот же результат, указав вместо нее опцию *-ss*, заставляющую *mysql* «помолчать»:

```
% mysql -e "SELECT account, shell FROM passwd" -ss cookbook > shells.txt
```

Имейте в виду, что значения NULL выводятся в виде строки «NULL». Возможно, потребуется выполнить некоторые преобразования в зависимости от того, как вы планируете использовать результат.

Перенаправив вывод результатов запроса в фильтр, заменяющий символы табуляции на другие разделители, можно получить выходные данные в формате, отличном от принятого по умолчанию. Например, чтобы использовать в качестве разделителя решетку, преобразуем символы TAB в # (строка *TAB* указывает то место в команде, где следует нажать клавишу табуляции):

```
% mysql -N -e "здесь ваш запрос" имя_БД | sed -e "s/TAB/#/g" > файл_вывода
```

Для этих целей можно также использовать команду *tr*, хотя ее синтаксис может отличаться в разных реализациях. В системах Mac OS X или RedHat Linux команда выглядит так:

```
% mysql -N -e "здесь ваш запрос" имя_БД | tr "\t" "#" > файл_вывода
```

В приведенных командах *mysql* используются опции *-N* или *-ss* для удаления из выходных данных заголовков столбцов. В некоторых случаях эти заголовки могут пригодиться. (Например, они могут понадобиться при последующем импорте.) Тогда исключите из команды опцию их отмены. В этом смысле экспорт результатов запроса посредством *mysql* более гибок, чем с помощью предложения SELECT ... INTO OUTFILE, так как в последнем отсутствует возможность вывода заголовков столбцов.

См. также

Еще одним способом экспорта результатов запроса в файл на клиентской машине является использование утилиты *mysql_to_text.pl*, описанной в рецепте 10.17. У этой программы есть опции для явного задания формата вывода. Для экспорта результатов запроса в формате Excel или FileMaker Pro воспользуйтесь рецептами 10.39 и 10.40.

10.14. Экспорт таблиц в виде необработанных данных

Задача

Вы хотите экспортировать в файл таблицу целиком.

Решение

Воспользуйтесь программой *mysqldump* с опцией *--tab*.

Обсуждение

Программа *mysqldump* применяется для копирования, в том числе резервного, таблиц и баз данных. Она может выгрузить таблицу как «сырой» файл и как набор предложений `INSERT`, пересоздающих записи таблицы. Первый способ обсуждается здесь, а второй – в рецептах 10.15 и 10.16.

Для создания дампа таблицы вы должны указать опцию *-tab* и определить в ней каталог, в который сервер MySQL поместит файл. (Каталог должен существовать, MySQL не станет его создавать.) Например, для создания дампа таблицы `states` из базы данных `cookbook` в каталоге */tmp* можно выполнить такую команду:

```
% mysqldump --no-create-info --tab=/tmp cookbook states
```

Программа *mysqldump* создает файл с именем, совпадающим с именем таблицы, и добавляет к нему суффикс *.txt*, так что в приведенном примере будет создан файл с именем */tmp/states.txt*. Такая форма *mysqldump* в некотором смысле является эквивалентом предложения `SELECT ... INTO OUTFILE` для командной строки. В частности, команда создает файл на серверном хосте, и для доступа к нему необходима привилегия `FILE`. Основные свойства предложения `SELECT ... INTO OUTFILE` рассмотрены в рецепте 10.13.

Если не указывать опцию *--no-create-info*, то *mysqldump* также создаст файл */tmp/states.sql*, в котором будет присутствовать предложение `CREATE TABLE`, создающее данную таблицу. (Владельцем последнего файла, в отличие от файла данных, будете вы, а не сервер.)

После имени базы данных можно указать несколько имен таблиц, в этом случае *mysqldump* создаст файлы для каждой из них. Если не указать ни одной таблицы, *mysqldump* создаст дампы всех таблиц базы данных.

Программа *mysqldump* по умолчанию создает файлы данных с символами табуляции и конца строки в качестве разделителей полей и строк. Для управления форматом вывода используйте опции *--fields-enclosed-by*, *--fields-terminated-by* и *--lines-terminated-by* (то есть те же спецификаторы формата, что и в программе *mysqlimport*). Например, для экспорта таблицы `states` в формате CSV с разделителями строк CRLF можно выполнить такую команду:

```
% mysqldump --no-create-info --tab=/tmp \
```

```
--fields-enclosed-by="\"" --fields-terminated-by="," \
--lines-terminated-by="\r\n" cookbook states
```

Созданный таким образом файл данных можно импортировать как предложением LOAD DATA, так и утилитой *mysqlimport*. Проверяйте соответствие спецификаторов формата, если при экспорте не используется формат по умолчанию.

10.15. Экспорт содержимого таблиц или определений в SQL-формат

Задача

Вы хотите экспортировать таблицы или базы данных в SQL-предложения, чтобы упростить их последующий импорт.

Решение

Используйте утилиту *mysqldump* без опции *--tab*.

Обсуждение

Как уже упоминалось в рецепте 10.14, утилита *mysqldump*, запущенная с опцией *--tab*, заставляет сервер MySQL записывать таблицы в файлы с «сырыми» данными на хосте сервера. Если же опустить опцию *--tab*, сервер форматирует записи таблицы как предложения INSERT и возвращает их в *mysqldump*. Также можно сформировать предложение CREATE TABLE для каждой таблицы. Такая форма вывода удобна для сохранения в файле с последующим использованием для пересоздания одной или нескольких таблиц. Подобные дампы часто используются как резервные копии или при копировании таблиц на другой сервер MySQL. В этом разделе показано, как сохранить вывод дампа в файл или непосредственно отправить по сети на другой сервер.

Чтобы экспортировать таблицу в файл в формате SQL, выполните такую команду:

```
% mysqldump cookbook states > dump.txt
```

Создается файл вывода *dump.txt*, содержащий как предложение CREATE TABLE, так и набор предложений INSERT:

```
# MySQL dump 8.16
#
# Host: localhost    Database: cookbook
#--------------------------------------------------------
# Server version    3.23.46-log

#
# Table structure for table 'states'
#

CREATE TABLE states (
```

```
  name varchar(30) NOT NULL default '',
  abbrev char(2) NOT NULL default '',
  statehood date default NULL,
  pop bigint(20) default NULL,
  PRIMARY KEY  (abbrev)
) TYPE=MyISAM;

#
# Dumping data for table 'states'
#

INSERT INTO states VALUES ('Alaska','AK','1959-01-03',550043);
INSERT INTO states VALUES ('Alabama','AL','1819-12-14',4040587);
INSERT INTO states VALUES ('Arkansas','AR','1836-06-15',2350725);
INSERT INTO states VALUES ('Arizona','AZ','1912-02-14',3665228);
INSERT INTO states VALUES ('California','CA','1850-09-09',29760021);
INSERT INTO states VALUES ('Colorado','CO','1876-08-01',3294394);
...
```

Для того чтобы вывести содержимое нескольких таблиц, укажите их имена после аргумента базы данных. Чтобы сделать дамп всей базы данных, ничего не указывайте после имени базы данных. Чтобы выполнить дамп всех таблиц всех баз данных, вызовите *mysqldump* так:

```
% mysqldump --all-databases > dump.txt
```

В этом случае файл вывода будет содержать предложения `CREATE DATABASE` и `USE` *имя_БД*, расположенные определенным образом, так что при последующем считывании файла каждая таблица будет создана в соответствующей базе данных. Опция *--all-databases* появилась в версии MySQL 3.23.12.

Доступны и другие опции, определяющие формат вывода:

--no-create-info

Не формировать предложения `CREATE TABLE`. Используйте опцию, если вы хотите выгрузить только содержимое таблицы.

--no-data

Не формировать предложения `INSERT`. Используйте опцию, если вы хотите выгрузить только определение таблицы.

--add-drop-table

Предварить каждое предложение `CREATE TABLE` предложением `DROP TABLE`. Это удобно для формирования файла, который может использоваться для повторного создания таблиц с нуля.

--no-create-db

Не формировать предложения `CREATE DATABASE`, которые обычно выводит опция *--all-databases*.

Теперь предположим, что вы используете *mysqldump* для создания файла дампа в формате SQL. Как осуществить обратный импорт в MySQL? Распространенной ошибкой является использование *mysqlimport*. Кажется логичным, что раз *mysqldump* экспортирует таблицы, *mysqlimport* должна их им-

портировать, не так ли? К сожалению, нет. Может быть, это и логично, но не всегда правильно. Правда в том, что если вы используете опцию *--tab* для *mysqldump*, то можете импортировать результирующие файлы данных при помощи *mysqlimport*. Но если вы выполняете дамп файла в формате SQL, *mysqlimport* не сможет корректно его обработать. Используйте лучше программу *mysql*. Конкретный способ выполнения операции зависит от того, что содержится в файле дампа. Если вы выгружали несколько баз данных при помощи *--all-databases*, то файл будет содержать соответствующие предложения `USE` *имя_БД* для выбора базы данных, к которой относится каждая таблица, и аргумент базы данных в командной строке не потребуется:

```
% mysql < dump.txt
```

Если таблицы выгружались из одной базы данных, необходимо указать *mysql*, в какую базу их следует импортировать:

```
% mysql имя_БД < dump.txt
```

Обратите внимание на то, что, используя вторую команду, вы можете загрузить таблицы не в ту базу данных, к которой они относились изначально. Такую возможность можно использовать, например, для создания копий таблицы или таблиц тестовой базы данных для опробования некоторых предложений, манипулирующих данными, во время отладки, чтобы не заботиться о сохранности исходных данных.

10.16. Копирование таблиц и баз данных на другой сервер

Задача

Вы хотите скопировать таблицы или базы данных с одного сервера MySQL на другой.

Решение

Используйте вместе *mysqldump* и *mysql*, соединив их каналом (pipe).

Обсуждение

Вывод *mysqldump* в формате SQL можно использовать для копирования таблиц и баз данных с одного сервера на другой. Предположим, что вы хотите копировать таблицу `states` базы данных `cookbook` локального хоста в базу данных `cb`, работающую на хосте *other-host.com*. В качестве одного из способов сделать это можно предложить выгрузить вывод в файл (см. рецепт 10.15):

```
% mysqldump cookbook states > dump.txt
```

Затем скопируем *dump.txt* на *other-host.com* и запустим там команду импортирования таблицы в базу данных `cb` сервера:

```
% mysql cb < dump.txt
```

Есть и другой способ, не требующий использования промежуточного файла, – отправка вывода *mysqldump* по сети непосредственно удаленному серверу MySQL. Если с хоста, на котором расположена база данных `cookbook`, вы можете соединиться с обоими серверами, то выполните такую команду:

```
% mysqldump cookbook states | mysql -h other-host.com cb
```

Часть команды, относящаяся к *mysqldump*, устанавливает соединение с локальным сервером и пишет вывод дампа в канал. Вторая половина команды, относящаяся к *mysql*, соединяется с удаленным сервером MySQL *other-host.com*. Она считывает ввод из канала и отправляет каждое предложение серверу *other-host.com*.

Если вы не можете напрямую соединиться с удаленным сервером, применяя *mysql* на локальной машине, отправьте вывод дампа в канал, который использует *ssh* для удаленного вызова *mysql* на *other-host.com*:

```
% mysqldump cookbook states | ssh other-host.com mysql cb
```

ssh соединяется с сервером *other-host.com* и запускает на нем *mysql*. Затем из канала считывается вывод *mysqldump* и передается удаленному процессу *mysql*. Использование *ssh* удобно для отправки дампа по сети машине, MySQL-порт которой блокирован брандмауэром, но соединения с SSH-портом разрешены.

Если у вас нет доступа к *ssh*, может быть, вы можете использовать *rsh*. Однако программа *rsh* более уязвима, так что предпочтительнее применять *ssh*.

Чтобы копировать на другой сервер несколько таблиц, укажите их имена после аргумента базы данных в команде *mysqldump*. Чтобы копировать всю базу данных, ничего не указывайте после имени базы данных, *mysqldump* скопирует все таблицы.

Если вы подумываете о том, чтобы вызвать *mysqldump* с опцией *--all-databases* для отправки всех баз данных на другой сервер, помните, что вывод будет включать таблицы базы данных *mysql*, в том числе таблицы привилегий. Если удаленный сервер работает с другим множеством пользователей, то вы навряд ли захотите заменить таблицы его привилегий!

10.17. Создание собственных программ экспорта

Задача

Вам не хватает встроенных возможностей экспортирования MySQL.

Решение

Напишите собственные утилиты.

Обсуждение

Если существующее программное обеспечение не делает того, чего бы хотелось вам, напишите собственные программы, экспортирующие данные. Этот

раздел посвящен написанию на Perl сценария *mysql_to_text.pl*, который выполняет произвольный запрос и экспортирует его в указанном формате. Он записывает вывод на клиентский хост и может содержать строку заголовков столбцов (ничего из этого `SELECT ... INTO OUTFILE` не умеет). Он формирует различные форматы вывода, причем проще, чем при использовании *mysql* с пост-процессором, и записывает вывод на клиентский хост, в отличие от *mysqldump*, которая может записывать на клиентскую машину только вывод в формате SQL.

Сценарий *mysql_to_text.pl* овнован на модуле *Text::CSV_XS*, который должен быть установлен в вашей системе. После установки модуля вы можете прочитать соответствующую документацию так:

```
% perldoc Text::CSV_XS
```

Этот модуль удобен, потому что вам нужно только предоставить ему массив значений, а он уже упакует их в корректно отформатированную строку вывода. Поэтому преобразование вывода запроса в формат CSV становится тривиальной задачей. Но главное достоинство модуля *Text::CSV_XS* в том, что его можно конфигурировать – вы можете указать, какие символы следует использовать для разделителей и заключения в кавычки. То есть по умолчанию модуль выдает формат CSV, но вы можете настроить его так, чтобы выводилось множество форматов. Например, если задать в качестве разделителя символ табуляции, а кавычкой назначить `undef`, *Text::CSV_XS* сформирует вывод, элементы которого разделяются знаками табуляции. Мы воспользуемся этой гибкостью при написании *mysql_to_text.pl*, а также впоследствии при создании утилиты обработки файлов, преобразующей файлы из одного формата в другой.

Сценарий *mysql_to_text.pl* принимает ряд опций командной строки. Некоторые из них используются для определения параметров соединения MySQL (такие, как *--user*, *--password* и *--host*). С ними вы уже знакомы, поскольку они применяются стандартными клиентскими приложениями MySQL, такими как *mysql*. Сценарий также может получать параметры соединения из файла опций, если задать в файле группу `[client]`. Кроме того, *mysql_to_text.pl* принимает следующие опции:

*--**execute**=запрос*, *-**e** запрос*

Выполнить *запрос* и экспортировать его вывод.

*--**table**=имя_таблицы*, *-**t** имя_таблицы*

Экспортировать содержимое указанной таблицы. Эквивалентно использованию *--**execute*** со значением запроса `SELECT * FROM` *имя_таблицы*.

*--**labels***

Вывести строку заголовков столбцов.

*--**delim**=строка*

Установить последовательность разделителя столбцов в значение *строка*. Значение опции может состоять из одного или нескольких символов. По умолчанию используются символы табуляции.

--quote=c

Установить символ кавычки в значение *c*. По умолчанию – ничего не заключать в кавычки.

--eol=строка

Установить последовательность конца строки в значение *строка*. Значение опции может состоять из одного или нескольких символов. По умолчанию используются символы перевода строки.

Значения по умолчанию для опций *--delim*, *--quote* и *--eol* соответствуют применяемым для предложений `LOAD DATA` и `SELECT ... INTO OUTFILE`.

Последним аргументом командной строки должно быть имя базы данных, если только оно не указывается неявно (в запросе). Например, две приведенные ниже команды эквиваленты, они экспортируют таблицу `passwd` в формате значений, разделенных символом двоеточия, в файл *tmp*:

```
% mysql_to_text.pl --delim=":" --table=passwd cookbook > tmp
% mysql_to_text.pl --delim=":" --table=cookbook.passwd > tmp
```

Чтобы сформировать вывод CSV с парой CRLF в качестве признака конца строки, выполните такую команду:

```
% mysql_to_text.pl --delim="," --quote="\"" --eol="\r\n" \
    --table=cookbook.passwd > tmp
```

Это было общее описание использования *mysql_to_text.pl*. Теперь давайте поговорим о том, как работает сценарий. В первой части сценария *mysql_to_text.pl* объявляется несколько переменных, затем аргументы командной строки обрабатываются при помощи приемов, описанных в рецепте 2.10. (Как это часто бывает, большая часть кода сценария посвящена обработке аргументов командной строки и настройке работы сценария!)

```
#! /usr/bin/perl -w
# mysql_to_text.pl - экспорт вывода запроса MySQL
# в указанном пользователем текстовом формате

# Использование: mysql_to_text.pl [ опции ] [имя базы данных] > текстовый файл

use strict;
use DBI;
use Text::CSV_XS;
use Getopt::Long;
$Getopt::Long::ignorecase = 0; # опции чувствительны к регистру
$Getopt::Long::bundling = 1;   # позволяет объединить короткие опции

my $prog = "mysql_to_text.pl";

# ... создание переменной с сообщением об использовании $usage (не приводится) ...

# Переменные для опций командной строки - все изначально не определены,
# кроме структуры вывода, которая установлена в формат значений,
# разделенных табуляциями, признак конца строки - LF.
my $help;
my ($host_name, $password, $port_num, $socket_name, $user_name, $db_name);
```

```
my ($query, $tbl_name);
my $labels;
my $delim = "\t";
my $quote;
my $eol = "\n";

GetOptions (
    # =i означает, что после опции требуется целый аргумент
    # =s означает, что после опции требуется строковый аргумент
    # :s означает, что после опции возможен строковый аргумент
    "help"          => \$help,          # вывести справочное сообщение
    "host|h=s"      => \$host_name,     # хост сервера
    "password|p:s"  => \$password,      # пароль
    "port|P=i"      => \$port_num,      # номер порта
    "socket|S=s"    => \$socket_name,   # имя сокета
    "user|u=s"      => \$user_name,     # имя пользователя
    "execute|e=s"   => \$query,         # выполняемый запрос
    "table|t=s"     => \$tbl_name,      # экспортируемая таблица
    "labels|l"      => \$labels,        # формирование строки заголовков столбцов
    "delim=s"       => \$delim,         # разделитель столбцов
    "quote=s"       => \$quote,         # символ заключения столбца в кавычки
    "eol=s"         => \$eol            # признак конца строки (записи)
) or die "$usage\n";

die  "$usage\n" if defined $help;

$db_name = shift (@ARGV) if @ARGV;

# Должна быть указана только одна из опций --execute и --table, но не обе
die "You must specify a query or a table name\n\n$usage\n"
    if !defined ($query) && !defined ($tbl_name);
die "You cannot specify both a query and a table name\n\n$usage\n"
    if defined ($query) && defined ($tbl_name);

# Если было указано имя таблицы, преобразовать его в запрос, выбирающий всю таблицу
$query = "SELECT * FROM $tbl_name" if defined ($tbl_name);

# преобразовать состояние определено/не определено в истина/ложь
$labels = defined ($labels);

# интерпретировать специальные символы в опциях структуры файла
$quote = interpret_option ($quote);
$delim = interpret_option ($delim);
$eol = interpret_option ($eol);
```

Функция `interpret_option()` обрабатывает экранирующие и шестнадцатеричные последовательности для опций *--delim*, *--quote* и *--eol*. Последовательности `\n`, `\r`, `\t` и `\0` интерпретируются как перевод строки, возврат каретки, табуляция и ASCII-символ NUL. Шестнадцатеричные значения могут указываться в формате `0x`*nn* (например, `0x0d` означает возврат каретки). Функция здесь не приводится; чтобы посмотреть, как она работает, обратитесь к исходному тексту сценария.

После обработки опций командной строки сценарий формирует имя источника данных (DSN) и устанавливает соединение с сервером:

```
my $dsn = "DBI:mysql:";
$dsn .= ";database=$db_name" if $db_name;
$dsn .= ";host=$host_name" if $host_name;
$dsn .= ";port=$port_num" if $port_num;
$dsn .= ";mysql_socket=$socket_name" if $socket_name;
# читать группу параметров [client] из стандартного файла опций
$dsn .= ";mysql_read_default_group=client";

my $dbh = DBI->connect ($dsn, $user_name, $password,
                        {PrintError => 0, RaiseError => 1});
```

Имя базы данных берется из командной строки. Параметры соединения могут задаваться в командной строке или в файле опций. Использование файла опций MySQL описано в рецепте 2.10.

После установки соединения с MySQL сценарий готов к выполнению запроса и формированию вывода. Тут в игру вступает модуль *Text::CSV_XS*. Сначала создаем объект CSV, вызывая функцию `new()`, которая принимает необязательный хеш опций, управляющих обработкой строк данных. Затем сценарий подготавливает и выполняет запрос, выводит строку заголовков столбцов (если пользователь указал опцию *--labels*) и выводит строки результирующего множества:

```
my $csv = Text::CSV_XS->new ({
                            sep_char => $delim,
                            quote_char => $quote,
                            escape_char => $quote,
                            eol => $eol,
                            binary => 1
                        });

my $sth = $dbh->prepare ($query);
$sth->execute ();
if ($labels)                                # вывод строки заголовков столбцов
{
    $csv->combine (@{$sth->{NAME}}) or die "cannot process column labels\n";
    print $csv->string ();
}

my $count = 0;
while (my @val = $sth->fetchrow_array ())
{
    ++$count;
    $csv->combine (@val) or die "cannot process column values, row $count\n";
    print $csv->string ();
}
```

Опции `sep_char` и `quote_char` вызова функции `name()` устанавливают последовательность разделителя столбцов и символ кавычки. Опция `escape_char` устанавливается в то же значение, что и `quote_char`, так что вхождения символа кавычки в значения данных будут продублированы в выводе. Опция `eol` задает последовательность конца строки. Обычно *Text::CSV_XS* поручает вам печать завершающих символов для строк вывода. Передавая `new()` значение `eol` не-`undef`, модуль автоматически добавляет это значение в каждую

строку вывода. Опция `binary` используется для обработки значений данных, содержащих двоичные символы.

Заголовки столбцов доступны в `$sth->{NAME}` после вызова `execute()`. Каждая строка вывода формируется с использованием `combine()` и `string()`. Метод `combine()` принимает массив значений и преобразует их в соответствующим образом отформатированную строку. Метод `string()` возвращает строку, и мы можем печатать ее.

10.18. Преобразование файлов данных из одного формата в другой

Задача

Вы хотите преобразовать файл в другой формат для упрощения работы с ним или чтобы сделать его доступным какой-то другой программе.

Решение

Используйте описанный в разделе сценарий конвертера *cvt_file.pl*.

Обсуждение

Сценарий *mysql_to_text.pl*, приведенный в разделе 10.17, использует MySQL как источник данных и формирует вывод в формате, указанном при помощи опций *--delim*, *--quote* и *--eol*. В этом разделе будет описана утилита *cvt_file.pl*, предоставляющая похожие возможности форматирования не только для вывода, но и для ввода. Она читает данные из файла, а не из MySQL, и преобразует их из одного формата в другой. Например, чтобы прочитать файл *data.txt*, элементы которого разделены символами табуляции, преобразовать его в формат значений, разделенных двоеточиями и записать результат в *tmp*, следует вызвать *cvt_file.pl* так:

```
% cvt_file.pl --idelim="\t" --odelim=":" data.txt > tmp
```

У сценария *cvt_file.pl* есть отдельные опции для ввода и вывода. То есть в то время как *mysql_to_text.pl* для указания разделителей столбцов использует опцию *--delim*, у *cvt_file.pl* есть две опции *--idelim* и *--odelim* для установки разделителей столбцов ввода и вывода. Но в качестве короткой записи поддерживается и *--delim*: разделитель столбцов устанавливается как для вывода, так и для ввода. Полный набор опций *cvt_file.pl* таков:

--idelim=строка, *--odelim=строка*, *--delim=строка*

Установить последовательность разделителя столбцов для ввода, вывода или ввода и вывода одновременно. Опция может состоять из одного или нескольких символов.

--iquote=с, *--oquote=с*, *--quote=с*

Установить символ заключения столбца в кавычки для ввода, вывода или ввода и вывода одновременно.

--ieol=строка, *--oeol=строка*, *--eol=строка*

Установить последовательность конца строки для ввода, вывода или ввода и вывода одновременно. Опция может состоять из одного или нескольких символов.

--iformat=формат, *--oformat=формат*, *--format=формат*,

Указать формат ввода, вывода или ввода и вывода одновременно. Эта опция является краткой записью установки значений кавычки и разделителя. Например, *--iformat*=csv устанавливает символы кавычки и разделителя ввода в двойные кавычки и запятую, а *--iformat*=tab указывает формат «без кавычек» с символом табуляции в качестве разделителя.

--ilabels, *--olabels*, *--labels*

Ожидать первую строку заголовков столбцов во вводе, записывать первую строку с заголовками столбцов при выводе или делать и то и другое. Если вы не считываете заголовки из ввода, но требуете их присутствия в выводе, *cvt_file.pl* использует заголовки c1, c2 и т. д.

Формат файла, ожидаемый *cvt_file.pl* по умолчанию, совпадает с форматом по умолчанию для предложений LOAD DATA и SELECT INTO ... OUTFILE, то есть разделители – символы табуляции, признак конца строки – символ перевода строки.

Сценарий *cvt_file.pl* включен в каталог *transfer* дистрибутива recipes. Если вы предполагаете регулярно его использовать, то следует установить его в каком-то из каталогов, имя которого указано в пути поиска, чтобы сценарий можно было вызывать откуда угодно. Большая часть исходного текста сценария похожа на *mysql_to_text.pl*, поэтому вместо того чтобы показывать вам код и рассказывать, как он работает, я просто приведу несколько примеров его использования:

- Читаем файл в формате CSV с парой CRLF в качестве признака конца строки, пишем элементы, разделенные символами табуляции, строки завершаются символом перевода строки:

```
% cvt_file.pl --iformat=csv --ieol="\r\n" --oformat=tab --oeol="\n" \
    data.txt > tmp
```

- Читаем и пишем формат CSV, преобразуя завершающую строки пару CRLF в символ возврата каретки:

```
% cvt_file.pl --format=csv --ieol="\r\n" --oeol="\r" data.txt > tmp
```

- Создаем файл элементов, разделенных символами табуляции, из файла */etc/passwd* с разделителями – символами двоеточия:

```
% cvt_file.pl --idelim=":" /etc/passwd > tmp
```

- Преобразуем в формат CSV вывод запроса из mysql, элементы которого разделены символами табуляции:

```
% mysql -e "SELECT * FROM profile" cookbook \
    | cvt_file.pl --oformat=csv > profile.csv
```

10.19. Извлечение и перестановка столбцов файлов данных

Задача

Вы хотите извлечь столбцы из файла данных или расположить их в другом порядке.

Решение

Используйте утилиту, которая по требованию может выводить столбцы файла.

Обсуждение

Сценарий *cvt_file.pl* служит инструментом преобразования целого файла из одного формата в другой. Еще одной распространенной операцией над файлом данных является манипулирование его столбцами. Это необходимо, например, при импортировании файла в программу, которая сама не понимает, как извлечь или изменить порядок столбцов. Возможно, вам потребуется пропустить столбцы из середины файла, чтобы использовать его в предложении `LOAD DATA`, которое не умеет перескакивать через столбцы, находящиеся не с края. Или, например, ваша версия *mysqlimport* старше, чем 3.23.17, и не поддерживает опцию *--columns*, позволяющую указать порядок столбцов в файле. Чтобы обойти эту проблему, можно произвести перестановку в файле данных.

Вспомните, что глава начиналась с описания ситуации с 12-столбцовым файлом в формате CSV *somedata.csv*, из которого нас интересовали только столбцы 2, 11, 5 и 9. Можно преобразовать файл в формат значений, разделенных табуляциями:

```
% cvt_file.pl --iformat=csv somedata.csv > somedata.txt
```

И что дальше? Если вы просто хотите создать небольшой сценарий, извлекающий перечисленные четыре столбца, это нетрудно: пишем цикл, который читает строки ввода и записывает только те столбцы, которые нужно, в указанном порядке. Если считать, что на входе использованы табуляции как разделители и символы перевода строки как признак конца строки, то можно написать простую программу на Perl, извлекающую четыре столбца:

```
#! /usr/bin/perl -w
# yank_4col.pl - пример извлечения четырех столбцов

# Извлечь столбцы 2, 11, 5 и 9 из 12-столбцового ввода в указанном порядке.
# Предполагаем, что столбцы разделены символами табуляции,
# признак конца строки – перевод строки.

use strict;

while (<>)
{
```

```
    chomp;
    my @in = split (/\t/, $_);          # разделить по символам табуляции
    # извлечь столбцы 2, 11, 5 и 9
    print join ("\t", $in[1], $in[10], $in[4], $in[8]) . "\n";
}

exit (0);
```

Запустите сценарий для чтения файла, содержащего 12 столбцов данных, и записи вывода, включающего только четыре столбца в указанном порядке:

```
% yank_4col.pl somedata.txt > tmp
```

Но *yank_4col.pl* – это сценарий, который может использоваться только для решения очень узкого круга задач. Давайте еще чуть-чуть поработаем и напишем более универсальную утилиту *yank_col.pl*, которая обеспечивает извлечение любого набора столбцов. Используя такое средство, вы могли бы указывать в командной строке список столбцов:

```
% yank_col.pl --columns=2,11,5,9 somedata.txt > tmp
```

Поскольку список столбцов не «зашит» в сценарий, можно выводить произвольное множество столбцов в любом порядке. Столбцы можно задавать списком номеров, разделенных запятыми, или диапазоном. (Например, *--columns=1,4-7,10* подразумевает столбцы 1, 4, 5, 6, 7 и 10.) Утилита будет выглядеть так:

```
#! /usr/bin/perl -w
# yank_col.pl - извлечение столбцов из ввода

# Пример: yank_col.pl --columns=2,11,5,9 filename

# Предполагаем, что строки ввода разделены символами табуляции,
# признак конца строки - перевод строки.

use strict;
use Getopt::Long;
$Getopt::Long::ignorecase = 0; # опции чувствительны к регистру

my $prog = "yank_col.pl";
my $usage = <<EOF;
Usage: $prog [options] [data_file]

Options:
--help
    Print this message
--columns=column-list
    Specify columns to extract, as a comma-separated list of column positions
EOF

my $help;
my $columns;

GetOptions (
    "help"      => \$help,          # вывести справочное сообщение
    "columns=s" => \$columns        # указать список столбцов
```

```
) or die "$usage\n";

die  "$usage\n" if defined $help;

my @col_list = split (/,/, $columns) if defined ($columns);
@col_list or die "$usage\n";            # требуется непустой список

# убедиться в том, что номера столбцов представляют собой целые положительные числа,
и сдвинуть начало отсчета с 1 в 0

my @tmp;
for (my $i = 0; $i < @col_list; $i++)
{
    if ($col_list[$i] =~ /^\d+$/)               # отдельный номер столбца
    {
        die "Column specifier $col_list[$i] is not a positive integer\n"
                unless $col_list[$i] > 0;
        push (@tmp, $col_list[$i] - 1);
    }
    elsif ($col_list[$i] =~ /^(\d+)-(\d+)$/)    # диапазон номеров m-n
    {
        my ($begin, $end) = ($1, $2);
        die "$col_list[$i] is not a valid column specifier\n"
                unless $begin > 0 && $end > 0 && $begin <= $end;
        while ($begin <= $end)
        {
            push (@tmp, $begin - 1);
            ++$begin;
        }
    }
    else
    {
        die "$col_list[$i] is not a valid column specifier\n";
    }
}
@col_list = @tmp;

while (<>)                      # читать ввод
{
    chomp;
    my @val = split (/\t/, $_, 10000);  # разбить на части, сохраняя все поля
    # извлечь нужные столбцы, отображая undef на пустую строку
    # (возможно, если индекс больше количества столбцов в строке)
    @val = map { defined ($_) ? $_ : "" } @val[@col_list];
    print join ("\t", @val) . "\n";
}

exit (0);
```

Цикл обработки ввода преобразует каждую строку в массив значений, затем извлекает из массива значения, соответствующие запрошенным столбцам. Чтобы не использовать цикл для массива, применяется нотация Perl, разрешающая указывать список индексов для одновременного доступа к несколь-

ким элементам массива. Например, если `@col_list` содержит значения `2`, `6` и `3`, то два таких выражения эквивалентны:

```
($val[2] , $val[6], $val[3])
@val[@col_list]
```

Но что делать, если вы хотите извлечь столбцы из файла, элементы которого разделены не символами табуляции, или сформировать вывод в другом формате? Используйте *yank_col.pl* в сочетании с *cvt_file.pl*. Предположим, что вы хотите извлечь все, кроме столбца паролей из файла */etc/passwd*, элементы которого разделены символами двоеточия, и записать результат в формате CSV. С помощью *cvt_file.pl* выполните предварительную обработку */etc/passwd*, преобразовав его в формат значений, разделенных табуляциями для *yank_col.pl*, и с помощью *cvt_file.pl* же произведите преобразование извлеченных значений в формат CSV:

```
% cvt_file.pl --idelim=":" /etc/passwd \
    | yank_col.pl --columns=1,3-7 \
    | cvt_file.pl --oformat=csv > passwd.csv
```

Если вы не хотите вводить одну длинную команду, то можете использовать для промежуточных операций временные файлы:

```
% cvt_file.pl --idelim=":" /etc/passwd > tmp1
% yank_col.pl --columns=1,3-7 tmp1 > tmp2
% cvt_file.pl --oformat=csv tmp2 > passwd.csv
% rm tmp1 tmp2
```

Как заставить функцию split() выводить все поля?

Функция Perl `split()` чрезвычайно полезна, но обычно она не возвращает завершающие пустые поля. То есть если вы выводите ровно столько полей, сколько возвращает `split()`, может оказаться, что количество столбцов в строках вывода не совпадает с аналогичным параметром для строк ввода. Чтобы этого не было, передайте третий аргумент, указывающий максимальное количество возвращаемых полей. Тогда `split()` будет возвращать столько полей, сколько фактически содержится в строке, или их запрошенное количество, в зависимости от того, какое из чисел меньше. Если значение третьего аргумента достаточно велико, то в результате будут возвращены все значения, вне зависимости от того, пустые они или нет. Сценарии данной главы используют значение счетчика полей, равное 10 000:

```
# разбиение строки по символам табуляции с извлечением всех полей
my @val = split (/\t/, $_, 10000);
```

В случае (маловероятном) если строка ввода имеет больше полей, чем 10 000, она будет усечена. Если вы полагаете, что это создаст проблемы, то можете еще увеличить значение счетчика.

10.20. Проверка корректности и преобразование данных

Задача

Вам необходимо убедиться в том, что данные, содержащиеся в файле, корректны.

Решение

Проверьте их, возможно, преобразовав в более подходящий формат.

Обсуждение

Рассмотренные ранее в главе рецепты показывают, как работать со структурными характеристиками файлов, читая строки и разделяя их на отдельные столбцы. Это очень полезные операции, но часто приходится иметь дело не только со структурой файла данных, но и с его содержимым:

- Иногда разумно проверить значения данных на предмет их разрешенности для типов столбцов, в которых их планируется хранить. Например, можно проверить, являются ли значения, предназначенные для столбцов `INT`, `DATE` и `ENUM`, соответственно целыми числами, датами в формате *CCYY-MM-DD* и разрешенными членами перечислимого типа.
- Значения данных могут потребовать переформатирования. Особенно часто встречается перезапись дат из одного формата в другой. Например, если вы импортируете файл FileMaker Pro в MySQL, то, вероятно, понадобится преобразование дат из формата *MM-DD-YY* в формат ISO. Если же действие выполняется в обратном направлении, из MySQL в FileMaker Pro, потребуется обратное преобразование дат, а также разбиение столбцов `DATETIME` и `TIMESTAMP` на отдельные столбцы даты и времени.
- Может возникнуть необходимость распознавания специальных значений. Значения `NULL` часто представляются значением, которое больше не встречается в файле, таким как `-1`, `Unknown` или `N/A`. Если вы не хотите, чтобы эти значения были переданы буквально, необходимо распознавать их и обрабатывать специальным образом.

Этот раздел начинает ряд рецептов, описывающих проверку корректности данных и способы форматирования, полезные в таких ситуациях. Рассматриваются: непосредственное сравнение, сравнение с образцом (pattern matching), проверка с использованием информации из базы данных. Некоторые операции проверки корректности встречаются снова и снова, так что, возможно, имеет смысл создать библиотеку функций. Упаковка операций проверки корректности в библиотеку упрощает создание использующих их утилит, а утилиты в свою очередь облегчают выполнение операций командной строки для целых файлов, избавляя вас от необходимости редактирования вручную.

Создание цикла обработки ввода

Многие из обсуждаемых рецептов являются типичными представителями проверок, выполняемых в программах, которые читают файл и проверяют значения отдельных столбцов. В общем виде утилиту подобной обработки файла можно записать так:

```
#! /usr/bin/perl -w
# loop.pl - Стандартный цикл обработки ввода

# Предполагаем, что строки ввода разделены символами табуляции,
# признак конца строки - перевод строки.

use strict;

while (<>)                              # читать каждую строку
{
    chomp;
    # разбить на части по символам табуляции, сохраняя все поля
    my @val = split (/\t/, $_, 10000);
    for my $i (0 .. @val - 1)   # перебор столбцов строки
    {
        # ... test $val[$i] here ...
    }
}

exit (0);
```

Цикл `while()` читает каждую строку ввода и разбивает ее на поля. Внутри цикла каждая строка разбита на поля. Затем внутренний цикл `for()` перебирает поля строки, обеспечивая их последовательную обработку. Если вы выполняете проверку не для всех полей одинаково, то следует заменить цикл `for()` отдельными проверками по столбцам.

Цикл работает с вводом, элементы которого разделены символами табуляции, а признаком конца строки является символ перевода строки. Таким же предполагают ввод и большинство утилит, создаваемых в оставшейся части главы. Чтобы применять эти программы для файлов данных другого формата, можно преобразовать их в файл, элементы которого разделены символами табуляции, используя сценарий *cvt_file.pl* из рецепта 10.18.

Размещение часто выполняемых проверок в библиотеке

Часто выполняемый тест можно оформить как библиотечную функцию. Операцию будет легко выполнять, кроме того, у нее появится имя, которое будет более понятным описанием производимого действия, чем сам код сравнения. Например, следующий тест выполняет сравнение с образцом, чтобы узнать, состоит ли `$val` целиком из цифр (возможно, предваренных знаком), затем проверяет, положительно ли значение:

```
$valid = ($val =~ /^\+?\d+$/ && $val > 0);
```

Другими словами, производится поиск строк, представляющих положительные целые числа. Чтобы упростить использование теста и сделать более

понятным его предназначение, можно поместить его в функцию, которая будет применяться так:

```
$valid = is_positive_integer ($val);
```

Сама функция определяется следующим образом:

```
sub is_positive_integer
{
my $s = shift;

    return ($s =~ /^\+?\d+$/ && $s > 0);
}
```

Теперь поместим определение функции в библиотечный файл, чтобы ее могли использовать различные сценарии. Файл модуля *Cookbook_Utils.pm* из каталога *lib* дистрибутива `recipes` является примером библиотечного файла, содержащего ряд функций проверки корректности данных. Посмотрите, может быть, какие-то его функции пригодятся вам при создании собственных программ (или в качестве модели для написания своих библиотечных функций). Чтобы обратиться к модулю из сценария, включите в него предложение `use`:

```
use Cookbook_Utils;
```

Естественно, необходимо установить файл модуля в каталоге, где Perl сможет его обнаружить. Подробная информация об установке библиотек приведена в рецепте 2.3.

Важное преимущество размещения набора утилит в библиотечном файле заключается в возможности использования их всяческими программами. Задачи манипулирования данными редко бывают абсолютно уникальными. Если вы сможете использовать хотя бы несколько тестов на корректность из библиотеки, вполне вероятно, что объем создаваемого кода уменьшится, даже для очень специфичных программ.

10.21. Проверка корректности. Прямое сравнение

Задача

Необходимо убедиться в том, что значение равно или не равно некоторому указанному значению или входит в указанный диапазон значений.

Решение

Выполните сравнение.

Обсуждение

Простейшим видом проверки корректности является выполнение сравнения с литералом:

```
# непустое значение
$valid = ($val ne "");
```

```
# непустое указанное значение
$valid = ($val eq "abc");
# одно из нескольких значений
$valid = ($val eq "abc" || $val eq "def" || $val eq "xyz");
# значение в диапазоне (1 to 10)
$valid = ($val >= 1 && $val <= 10);
```

Большая часть таких проверок выполняет строковые сравнения. Последний тест – числовой, однако выполнению числовых проверок часто предшествуют предварительные тесты на отсутствие в значении нецифровых символов (например, сравнение с образцом, которому посвящен следующий раздел).

По умолчанию строковые сравнения чувствительны к регистру. Чтобы сделать операцию нечувствительной к регистру, преобразуйте оба операнда к одному регистру:

```
# указанное непустое значение вне зависимости от регистра
$valid = (lc ($val) eq lc ("AbC"));
```

10.22. Проверка корректности. Сравнение с образцом

Задача

Вам нужно сравнить значение с набором значений, который сложно задать буквально, не используя по-настоящему ужасающее выражение.

Решение

Используйте сравнение с образцом.

Обсуждение

Сравнение с образцом – мощное средство проверки корректности данных: оно позволяет тестировать целые группы значений при помощи одного-единственного выражения. Кроме того, можно использовать сравнение с образцом для разбиения подходящих значений на части для последующих индивидуальных проверок, а также в операциях замены для перезаписи соответствующих значений. Например, можно разбить дату на части, чтобы проверить, находится ли месяц в диапазоне от 1 до 12, а день – в рамках месяца. Или можно использовать замену для преобразования значений *MM-DD-YY* и *DD-MM-YY* в формат *YY-MM-DD*.

Следующие несколько разделов описывают применение шаблонов для проверки различных типов значений, пока же давайте поговорим об общих принципах работы с образцами. Будем рассматривать возможности регулярных выражений Perl, сравнение с образцом в PHP и Python выполняется аналогично (различия описаны в специальной документации). Что касается Java, библиотека класса ORO предоставляет возможности сравнения с образцом, подобные поддерживаемым в Perl; в приложении A указано, как можно получить доступ к этой библиотеке.

В Perl конструктором образца является */образец/*:

```
$it_matched = ($val =~ /образец/);     # соответствие образцу
```

Чтобы сделать сравнение с образцом нечувствительным к регистру, добавьте `i` после конструктора */образец/*:

```
$it_matched = ($val =~ /образец/i);    # соответствие образцу, нечувствительное к
регистру
```

Чтобы использовать символ, отличный от слэша, начните конструктор с `m` (применяется, если сам образец содержит слэши):

```
$it_matched = ($val =~ m|образец|);    # другой символ конструктора
```

Чтобы выполнить проверку на несоответствие, замените оператор `=~` оператором `!~`:

```
$no_match = ($val !~ /образец/);       # несоответствие образцу
```

Чтобы выполнить в `$val` замену по результатам сравнения с образцом, используйте `s/образец/замена/`. Если *образец* встречается в `$val`, производится замена на *замена*. Чтобы выполнить сравнение с образцом, нечувствительное к регистру, добавьте `i` после последнего слэша. Чтобы выполнить глобальную замену – заменить все вхождения *образец*, а не только первое, добавьте `g` после последнего слэша:

```
$val =~ s/образец/замена/;     # замена
$val =~ s/образец/замена/i;    # замена, нечувствительная к регистру
$val =~ s/образец/замена/g;    # глобальная замена
$val =~ s/образец/замена/ig;   # глобальная замена, нечувствительная к регистру
```

Перечень некоторых специальных элементов, используемых в регулярных выражениях Perl, приведен в табл. 10.1:

Таблица 10.1. Специальные символы регулярных выражений Perl

Образец	Описание соответствия
`^`	Начало строки
`$`	Конец строки
`.`	Любой символ
`\s`, `\S`	Пробельный или непробельный символ
`\d`, `\D`	Цифровой или нецифровой символ
`\w`, `\W`	Символ, используемый в словах (буквенно-цифровой символ или подчеркивание) или не используемый в словах
`[...]`	Любой из символов, перечисленных в квадратных скобках
`[^...]`	Любой из символов, не перечисленных в квадратных скобках
p1`|`*p2*`|`*p3*	Дизъюнкция; соответствие любому из образцов *p1*, *p2* или *p3*
`*`	Ноль или более вхождений предыдущего элемента
`+`	Одно или более вхождений предыдущего элемента

Образец	Описание соответствия
{*n*}	*n* вхождений предыдущего элемента
{*m*, *n*}	От *m* до *n* вхождений предыдущего элемента

Многие из таких элементов совпадают с доступными в операторе регулярных выражений MySQL – `REGEXP` (см. рецепт 4.7).

Чтобы выполнить сравнение со специальным символом образца, таким как `*`, `^` или `$`, предварите его символом обратного слэша. Аналогично, чтобы включить в класс символов символ, имеющий специальное значение в конструкторе класса (`[`, `]` или `-`), поставьте перед ним символ обратного слэша. Чтобы включить в класс символов литерал `^`, укажите его не первым в скобках.

Многие из образцов, используемых в последующих разделах, имеют форму `/^`*образец*`$/`. Такое начало и окончание шаблона приводят к тому, что *образец* сравнивается со всей исследуемой строкой целиком. Такие проверки наиболее часто встречаются в контексте проверки корректности данных, так как обычно необходима информация о том, совпадает ли образец со всем значением ввода, а не его частью. (Если вы хотите, например, убедиться в том, что значение является целым числом, вам вряд ли пригодятся сведения о том, что оно включает в себя целое число.) Однако это не непреложное правило, и иногда стоит использовать более мягкий тест, опустив символы `^` и `$`. Например, если вы хотите пропустить начальный и конечный пробелы в значениях, используйте образец, закрепленный только в начале строки, и второй образец, в котором закреплен только конец:

```
$val =~ s/^\s+//;       # отрезаем начальный пробел
$val =~ s/\s+$//;       # отрезаем конечный пробел
```

На самом деле эта операция используется так часто, что является хорошим кандидатом на помещение в библиотечную функцию. Файл *Cookbook_Utils.pm* содержит функцию `trim_whitespace()`, которая выполняет обе замены и возвращает результат:

```
$val = trim_whitespace ($val);
```

Чтобы запомнить подразделы строки, совпадающей с образцом, используйте скобки для соответствующих частей образца. В случае совпадения вы сможете ссылаться на подстроки при помощи переменных `$1`, `$2` и т. д.:

```
if ("abcdef" =~ /^(ab)(.*)$/)
{
    $first_part = $1;   # это будет ab
    $the_rest = $2;     # это будет cdef
}
```

Чтобы показать, что элемент образца является необязательным, поставьте после него символ `?`. Для сравнения значений с последовательностью цифр, возможно, со знаком «минус», которая может заканчиваться точкой, используйте такой образец:

```
/^-?\d+\.?$/
```

Скобки можно использовать и для группировки дизъюнкций внутри образца. Следующий образец соответствует значениям времени в формате *hh:mm*, за которыми могут (необязательно) следовать AM или PM:

```
/^\d{1,2}:\d{2}\s*(AM|PM)?$/i
```

Скобки в этом образце имеют и побочный эффект – необязательная часть запоминается в $1. Чтобы такого не было, используйте (?:*образец*):

```
/^\d{1,2}:\d{2}\s*(?:AM|PM)?$/i
```

Теперь у вас достаточно базовых знаний о сравнениях с образцами в Perl для создания тестов на корректность различных типов данных. В следующих разделах предлагаются образцы, которые можно использовать для широкой классификации значений, сравнения чисел, временных типов, адресов электронной почты и URL.

Каталог *transfer* дистрибутива recipes содержит сценарий *test_pat.pl*, который читает значения ввода, сравнивает их с различными образцами и сообщает о том, каким образцам соответствует каждое из значений. Сценарий поддается наращиванию, так что вы можете использовать его как средство тестирования для собственных образцов.

10.23. Образцы для широкой классификации

Задача

Вы хотите разбить значения на широкие категории.

Решение

Используйте достаточно общий образец.

Обсуждение

Если вам необходимо знать только, являются значения пустыми или нет, или состоят только из определенных видов символов, вам будет достаточно образцов из табл. 10.2:

Таблица 10.2. Образцы для широкой классификации

Образец	Описание соответствия
/^$/	Пустая строка
/./	Непустая строка
/^\s*$/	Пробельные символы, возможно, пустая строка
/^\s+$/	Непустая строка пробельных символов
/\S/	Непустая строка не только из пробельных символов
/^\d+$/	Только цифры, непустая строка

Образец	Описание соответствия
/^[a-z]+$/i	Только буквенные символы (нечувствительные к регистру), непустая строка
/^\w+$/	Только буквенно-цифровые символы или символ подчеркивания, непустая строка

10.24. Образцы для числовых значений

Задача

Вам необходимо убедиться в том, что строка представляет собой число.

Решение

Используйте образец, соответствующий интересующему вас типу чисел.

Обсуждение

Образцы могут использоваться для распределения значений по различным категориям чисел (табл. 10.3):

Таблица 10.3. Образцы для числовых значений

Образец	Описание соответствия
/^\d+$/	Целое число без знака
/^-?\d+$/	Целое отрицательное число или целое число без знака
/^[-+]?\d+$/	Целое число со знаком или без знака
/^[-+]?(\d+(\.\d*)?|\.\d+)$/	Число с плавающей точкой

Образец /^\d+$/ соответствует целым числам без знака, требуя непустого значения, состоящего от начала и до конца из цифр. Если вам нужно только, чтобы значение начиналось с целого числа, вы можете сравнивать с образцом начальную числовую часть и извлекать ее. Для этого опустите символ $, требующий соответствия до самого конца строки, и заключите в скобки часть \d+. В случае совпадения можно ссылаться на соответствующее образцу число как на $1:

```
if ($val =~ /^(\d+)/)
{
    $val = $1;  # установить значение в совпавшую составляющую
}
```

Вы также можете добавить к значению ноль, тогда Perl выполнит неявное преобразование строки в число, отбросив нечисловой суффикс:

```
if ($val =~ /^\d+/)
{
    $val += 0;
}
```

Однако если запустить Perl с опцией *-w* (что я лично рекомендую), такое преобразование приведет к выводу предупреждений для значений, которые действительно содержат нечисловую часть. Кроме того, строковые значения типа `0013` будут преобразованы в число `13`, что не всегда допустимо.

Некоторые числовые значения имеют специальный формат или подчиняются особым правилам. Рассмотрим несколько примеров и поговорим о том, как с ними работать:

Почтовые индексы

Почтовые индексы Zip и Zip+4 используются для доставки корреспонденции в США. Они имеют значения типа `12345` или `12345-6789` (то есть пять цифр, после которых могут идти дефис и еще четыре цифры). Для поиска соответствия одной или другой форме используются образцы, приведенные в табл. 10.4:

Таблица 10.4. Образцы для почтовых индексов

Образец	Описание соответствия
`/^\d{5}$/`	Почтовый индекс Zip, только пять цифр
`/^\d{5}-\d{4}$/`	Почтовый индекс Zip+4
`/^\d{5}(-\d{4})?$/`	Почтовый индекс Zip или Zip+4

Номера кредитных карт

Номера кредитных карт обычно состоят из цифр, но между ними часто встречаются пробелы, дефисы или другие символы, разделяющие группы цифр. Например, следующие номера должны восприниматься как одинаковые:

```
0123456789012345
0123 4567 8901 2345
0123-4567-8901-2345
```

Для таких значений используйте образец:

```
/^[- \d]+/
```

(Обратите внимание, что Perl разрешает использовать внутри классов символов спецификатор цифры `\d`.) Но такой образец не распознает значения неправильной длины (это свойство можно использовать для удаления посторонних символов). Если вам нужно, чтобы значения кредитных карт содержали ровно 16 цифр, используйте замену для удаления всех посторонних символов, затем проверьте длину результата:

```
$val =~ s/\D//g;
$valid = (length ($val) == 16);
```

Иннинги

В бейсболе для подающих ведется такая статистика, как количество результативных иннингов (inning, период игры), измеряемое в третях ин-

нингов (что соответствует числу зарегистрированных аутов). На эти числовые значения накладываются дополнительные ограничения: в них допустима дробная часть, которая (если имеется) должна состоять из одной цифры: 0, 1 или 2. То есть разрешенные значения имеют вид `0`, `.1`, `.2`, `1`, `1.1`, `1.2`, `2` и т. д. Для установления соответствия целому числу без знака (за которым может следовать десятичная точка и, возможно, дробная часть, представленная цифрой `0`, `1` или `2`) или цифре дробной части без начальной целой части используйте такой образец:

```
/^(\d+(\.[012]?)?|\.[012])$/
```

Варианты внутри образца сгруппированы в скобках, так как в противном случае символ `^` закрепит первый из них за началом строки, а символ `$` – второй за концом строки.

10.25. Образцы для дат и времени

Задача

Вы хотите убедиться в том, что строка представляет собой дату или время.

Решение

Используйте образец для значения времени интересующего вас типа. Не забудьте о проверке таких деталей, как разделители элементов и длины этих элементов.

Обсуждение

Проверка корректности дат – это сплошная головная боль, так как они могут быть представлены во множестве форматов. Сравнение с образцом чрезвычайно полезно для удаления недействительных значений, но часто его бывает недостаточно для полной проверки: дата может содержать число там, где должен быть месяц, но это может быть число 13. В этом рецепте мы обсудим несколько образцов для распространенных форматов дат. Рецепт 10.30 более подробно рассматривает данный вопрос и показывает, как использовать сравнение с образцом в сочетании с проверкой содержимого.

Формату дат ISO (*CCYY-MM-DD*) отвечает такой образец:

```
/^\d{4}-\d{2}-\d{2}$/
```

Он требует, чтобы разделителем частей даты был символ `-`. Для того чтобы наряду с дефисом разделителем мог служить и символ `/`, используйте между числовыми составляющими класс символов (слэш экранируется обратным слэшем, чтобы он не интерпретировался как конец конструктора образца):

```
/^\d{4}[-\/]\d{2}[-\/]\d{2}$/
```

А можно использовать другой ограничитель вокруг образца, тогда удастся избежать применения обратного слэша:

```
m|^\d{4}[-/]\d{2}[-/]\d{2}$|
```

Чтобы разрешить использование любого нечислового разделителя (а именно так действует MySQL, интерпретируя строки как даты), воспользуйтесь таким образцом:

```
/^\d{4}\D\d{2}\D\d{2}$/
```

Если необязательно, чтобы в каждой части присутствовали все четыре цифры (чтобы можно быть опустить начальные нули в значениях типа 03), будем искать числовые трехзначные непустые последовательности:

```
/^\d+\D\d+\D\d+$/
```

Конечно, это очень общий образец, и ему будут соответствовать и другие значения, например, номера социального страхования США, имеющие формат 012-34-5678. Чтобы наложить ограничение на длины составляющих значений, потребовав, чтобы часть года имела не менее двух разрядов, часть дня и месяца – один или два разряда, используйте следующий образец:

```
/^\d{2,4}?\D\d{1,2}\D\d{1,2}$/
```

Для дат в форматах *MM-DD-YY* и *DD-MM-YY* применяются похожие образцы, только составляющие располагаются в другом порядке. Предлагаемый образец соответствует обоим форматам:

```
/^\d{2}-\d{2}-\d{2}$/
```

Если необходимо проверить значения отдельных составляющих даты, используйте в образце скобки и после установления соответствия извлекайте подстроки. Например, для дат в формате ISO можно сделать нечто подобное:

```
if ($val =~ /^(\d{2,4})\D(\d{1,2})\D(\d{1,2})$/)
{
    ($year, $month, $day) = ($1, $2, $3);
}
```

Библиотечный файл *lib/Cookbook_Utils.pm* дистрибутива recipes содержит несколько таких сравнений с образцом, оформленных как вызовы функции. Если дата не соответствует образцу, они возвращают undef. В противном случае они возвращают ссылку на массив, содержащий выделенные значения года, месяца и дня, которые могут пригодиться для дальнейших проверок компонентов даты. Например, функция is_iso_date() ищет даты в формате ISO:

```
sub is_iso_date
{
my $s = shift;

    return undef unless $s =~ /^(\d{2,4})\D(\d{1,2})\D(\d{1,2})$/;
    return [ $1, $2, $3 ];  # вернуть год, месяц, день
}
```

Использовать функцию можно так:

```
my $ref = is_iso_date ($val);
if (defined ($ref))
{
    # $val соответствует образцу формата ISO;
```

```
    # проверять его составляющие, используя $ref->[0] - $ref->[2]
}
else
{
    # $val не соответствует образцу формата ISO
}
```

Дополнительная проверка дат проводится достаточно часто, так как несмотря на то что образцы и помогают отбросить синтаксически неправильные значения, они не оценивают корректность отдельных составляющих. Необходимо выполнять проверку вхождения значений в диапазон. Об этом мы поговорим в рецепте 10.30.

Если вы хотите пропустить проверку составляющих и просто перезаписать значения, используйте подстановку. Например, чтобы преобразовать значения из формата *MM-DD-YY* в формат *YY-MM-DD*, сделайте следующее:

```
$val =~ s/^(\d+)\D(\d+)\D(\d+)$/$3-$1-$2/;
```

Значения времени организованы немного более упорядоченно, чем даты; обычно первыми записываются часы, а последними – секунды, при этом каждая часть состоит из двух цифр:

```
/^\d{2}:\d{2}:\d{2}$/
```

Можно несколько смягчить требования, разрешая представлять составляющую часов одним знаком или допуская отсутствие составляющей секунд:

```
/^\d{1,2}:\d{2}(:\d{2})?$/
```

Составляющие времени можно поместить в скобки, если вы планируете осуществлять для них отдельные проверки на принадлежность диапазону или, например, переформатировать значения, добавляя составляющую секунд `00` туда, где она отсутствует. Но будьте внимательны и не забывайте о символе `?` в тех случаях, когда составляющая секунд является необязательной. Хотелось бы сделать так, чтобы все выражение `:\d{2}` в конце образца было необязательным, и при этом символ `:` не сохранялся бы в `$3`, если третий раздел времени все-таки присутствует. Для этого используем `(?:`*образец*`)`, нотацию группировки, которая не сохраняет подобранную подстроку. В такой нотации можно заключать цифры в скобки, чтобы сохранить их. Тогда `$3` будет содержать `undef`, если составляющая секунд отсутствует, а в противном случае будет хранить эту составляющую:

```
if ($val =~ /^(\d{1,2}):(\d{2})(?::(\d{2}))?$/)
{
    my ($hour, $min, $sec) = ($1, $2, $3);
    $sec = "00" if !defined ($sec); # секунды отсутствуют, используем 00
    $val = "$hour:$min:$sec";
}
```

Чтобы преобразовать значения из 12-часового формата с суффиксами AM и PM в 24-часовой формат, сделаем так:

```
if ($val =~ /^(\d{1,2}):(\d{2})(?::(\d{2}))?\s*(AM|PM)?$/i)
```

```
{
    my ($hour, $min, $sec) = ($1, $2, $3);
    # добавляем отсутствующие секунды
    $sec = "00" unless defined ($sec);
    # преобразуем 0 .. 11 -> 12 .. 23 для времен PM
    $hour += 12 if defined ($4) && uc ($4) eq "PM";
    $val = "$hour:$min:$sec";
}
```

Составляющие времени помещаются в `$1`, `$2` и `$3`, при этом `$3` устанавливается в `undef`, если значение не содержит секунд. Если есть суффикс, он попадает в `$4`. Если суффикс – это `AM` или его вообще нет (`undef`), значение интерпретируется как время до полудня (AM). Если же суффикс – это `PM`, значение интерпретируется как время после полудня.

См. также

Этот раздел представляет только азы того, что можно делать при обработке дат в процессе передачи данных. Проверка и преобразование дат и времени могут быть весьма специфическими, а общий объем всего этого разнообразия просто ошеломляет. Приведем некоторые вопросы, ответ на которые может вас заинтересовать:

- Каков базовый формат даты? Даты могут быть представлены в различных общепринятых форматах, таких как ISO (*CCYY-MM-DD*), принятый в США (*MM-DD-YY*) и в Великобритании (*DD-MM-YY*). И это только несколько наиболее распространенных форматов, возможны и другие. Например, файл данных может включать даты, записанные как `June 17, 1959` или как `17 Jun '59`.
- Разрешено ли в датах замыкающее время или, может быть, необходимо? Если предполагается наличие времени, должны ли присутствовать все его компоненты или только часы и минуты?
- Разрешены ли такие значения, как `now` и `today`?
- Должны ли составляющие даты разделяться каким-то определенным символом, например `-` или `/`, или разрешены и другие разделители?
- Должны ли составляющие даты иметь определенную длину? Разрешено ли отбрасывать начальные нули в составляющих месяца и года?
- Записываются ли месяцы в числовом виде или представляются названиями в форме `January` или `Jan`?
- Разрешены ли двузначные значения для года? Следует ли преобразовывать их в четырехзначные? Если да, то по каким правилам? (Какова точка перехода для диапазона `00-99`, в которой изменяется значение века?)
- Должны ли проверяться на корректность составляющие дат? Образцы могут распознать строки, которые выглядят как даты и время, но, несмотря на всю их полезность для отсева неправильно построенных значений, их может быть недостаточно. Значение типа `1947-15-99` может соответствовать образцу, но недопустимо для даты. Поэтому сравнение с об-

разцом наиболее эффективно используется в сочетании с проверками на вхождение в диапазон отдельных составляющих даты.

Поскольку все эти вопросы действительно возникают постоянно, вы, вероятно, придете к тому, чтобы написать несколько собственных тестов для проверки дат, представленных в специальных форматах. В последующих разделах вам в помощь будут предложены некоторые средства, например, в рецепте 10.29 представлено преобразование двузначных значений года в четырехзначные, а рецепт 10.30 рассказывает о проверках на корректность составляющих значений дат и времени.

10.26. Образцы для адресов электронной почты и URL

Задача

Вы хотите определить, содержится ли в значении адрес электронной почты или URL.

Решение

Используйте образец необходимого уровня строгости.

Обсуждение

В предыдущих разделах образцы использовались для идентификации таких классов значений, как числа и даты, и надо сказать, что регулярные выражения очень часто применяются именно так. Но вообще сравнение с образцом имеет настолько широкую область возможного применения, что невозможно даже просто перечислить все варианты его использования для проверки корректности данных. Для того чтобы дать общее представление о том, для каких еще типов значений может применяться сравнение с образцом, приведем в этом разделе несколько тестов для адресов электронной почты (email) и URL.

Чтобы проверить, являются ли значения адресами электронной почты, образец должен требовать как минимум наличия символа `@`, с двух сторон окруженного непустыми строками:

```
/.@./
```

Это минимальная проверка. Придумать общий образец, охватывающий все разрешенные значения и отбрасывающий все неразрешенные, нелегко, так что создадим хотя бы образец, налагающий чуть более строгие ограничения.[1]

[1] Чтобы посмотреть, насколько сложным может быть сравнение с образцом для адреса электронной почты, обратитесь к приложению E книги Джеффри Фридла (Jeffrey Friedl) «Mastering Regular Expressions», O'Reilly («Регулярные выражения», издательство «Питер», 2003) .

Например, имя пользователя и имя домена должны не просто быть непустыми, а целиком состоять из символов, отличных от @ и пробелов:

```
/^[^@ ]+@[^@ ]+$/
```

Кроме того, можно потребовать, чтобы доменное имя состояло как минимум из двух частей, разделенных точкой:

```
/^[^@ ]+@[^@ .]+\.[^@ .]+/
```

Чтобы найти значения URL, которые начинаются с указателя протокола `http://`, `ftp://` или `mailto:`, используйте дизъюнкцию, устанавливающую соответствие с любым из них в начале строки. Такие значения содержат символы слэша, поэтому разумно заключить образец в другие символы, чтобы избежать экранирования слэша символом обратного слэша:

```
m#^(http://|ftp://|mailto:)#i
```

Варианты в образце группируются при помощи скобок, так как в противном случае только первый из них оказался бы закрепленным в начале строки посредством ^. Модификатор `i`, следующий за образцом, обозначает нечувствительность указателя протокола к регистру. Больше образец никаких ограничений не накладывает, разрешая чему угодно следовать за указателем протокола. Если хотите, можете сами выполнить дальнейшее усовершенствование этого образца.

10.27. Проверка корректности при помощи метаданных таблицы

Задача

Вам необходимо проверить, являются ли вводимые значения разрешенными для столбца `ENUM` или `SET`.

Решение

Получите определение столбца, извлеките список его членов и сверьте вводимое значение со списком.

Обсуждение

Некоторые виды проверки корректности данных требуют использования информации, хранящейся в базе данных. Это относится и к значениям, предполагаемым для хранения в столбце `ENUM` или `SET`, которые можно сравнить с допустимыми членами из определений таких столбцов. Проверки с использованием информации из базы данных также проводятся для значений, которые должны совпадать с перечисленными в справочных таблицах. Например, можно потребовать, чтобы записи, содержащие идентификаторы клиентов, соответствовали записям таблицы `customers`, а аббревиатуры названий штатов в адресах можно сравнивать с таблицей штатов. В этом разде-

ле описаны проверки для столбцов ENUM и SET, а рецепт 10.28 посвящен использованию справочных таблиц.

Одним из способов проверки вводимых значений на соответствие разрешенным значениям для столбцов ENUM или SET является получение списка допустимых значений в массиве с помощью информации, возвращаемой запросом SHOW COLUMNS, а затем — выполнение проверки на принадлежность массиву. Например, столбец любимого цвета color таблицы profile относится к типу ENUM и определен так:

```
mysql> SHOW COLUMNS FROM profile LIKE 'color'\G
*************************** 1. row ***************************
  Field: color
   Type: enum('blue','red','green','brown','black','white')
   Null: YES
    Key:
Default: NULL
  Extra:
```

Если извлечь список перечислимых значений из Type и сохранить в массиве @members, то проверку на членство можно будет выполнить так:

```
$valid = grep (/^$val$/i, @members);
```

Конструктор образца начинается и заканчивается символами ^ и $, которые требуют полного совпадения $val с членом перечисления (а не просто подстрокой). За конструктором следует модификатор i, задающий сравнение, нечувствительное к регистру, так как столбцы ENUM не чувствительны к регистру.

В рецепте 9.6 была написана функция get_enumorset_info(), возвращающая метаданные столбцов ENUM и SET. Метаданные включали и список членов, так что используем эту функцию для создания другой полезной программы, check_enum_value(), которая извлекает разрешенные значения перечислимого типа и выполняет тест на принадлежность. Программа принимает четыре аргумента: дескриптор базы данных, имя таблицы, имя столбца ENUM и значение для проверки. В зависимости от допустимости значения возвращается «истина» или «ложь»:

```
sub check_enum_value
{
my ($dbh, $tbl_name, $col_name, $val) = @_;

    my $valid = 0;
    my $info = get_enumorset_info ($dbh, $tbl_name, $col_name);
    if ($info && $info->{type} eq "enum")
    {
        # сравнение будет нечувствительным к регистру,
        # т. к. столбцы ENUM нечувствительны к регистру.
        $valid = grep (/^$val$/i, @{$info->{values}});
    }
    return ($valid);
}
```

Такой тест хорошо подходит для проверки отдельных значений, например, для значения, передаваемого через веб-форму. Но если вы собираетесь тестировать ряд значений (например, целый столбец файла данных), лучше один раз считать значения перечислимого типа в память, а затем многократно использовать их для проверки каждого значения данных. Более того, гораздо более эффективными являются справочные хеши, а не массивы (по крайней мере, в Perl). Разрешенные значения перечислимого типа извлекаются и хранятся как ключи хеша. Затем для каждого вводимого значения проверяется, существует ли такой ключ хеша. Создать хеш чуть сложнее, поэтому `check_enum_value()` этим не занимается. Но для пакетной проверки корректности выигрыш от ускорения просмотра больше, чем расходы на формирование хеша.[1]

Начнем с получения метаданных для столбца, затем преобразуем список разрешенных значений перечислимого типа в хеш:

```
my $ref = get_enumorset_info ($dbh, $tbl_name, $col_name);
my %members;
foreach my $member (@{$ref->{values}})
{
    # преобразуем ключи хеша к одному регистру; ENUM нечувствителен к регистру
    $members{lc ($member)} = 1;
}
```

Цикл делает каждый член перечислимого типа ключом элемента хеша. В данном случае важен сам ключ, сопоставленное ему значение к делу не относится. (В приведенном примере значение устанавливается в `1`, но вы можете использовать `undef`, `0` или любое другое значение.) Обратите внимание на то, что код перед сохранением преобразует ключи хеша к нижнему регистру. Дело в том, что ключи хеша в Perl чувствительны к регистру. Это удобно, если проверяемые значение тоже чувствительны к регистру, но столбцы `ENUM` не обладают таким свойством. Преобразуя перед сохранением в хеш значения перечислимого типа к определенному регистру, а затем аналогичным образом преобразуя проверяемые значения, вы тем самым делаете нечувствительным к регистру тест на существование ключа:

```
$valid = exists ($members{lc ($val)});
```

В нашем примере значения приводились к нижнему регистру, но можно выбрать и верхний, важно лишь единообразно применить преобразование ко всем значениям.

Заметьте, что проверка на существование может оказаться неудачной, если вводимое значение является пустой строкой. Вам придется придумать, как обрабатывать такую ситуацию в зависимости от столбца. Например, если столбец допускает использование значений `NULL`, можно интерпретировать пустую строку как эквивалент `NULL`, то есть как разрешенное значение.

---

[1] Если вы хотите самостоятельно проверить относительную эффективность проверки принадлежности массиву и просмотра хеша, выполните сценарий *lookup_time.pl* из каталога *transfer* дистрибутива `recipes`.

Для столбцов SET процедура проверки корректности аналогична процедуре для значений ENUM, только вводимое значение может состоять из любого количества членов SET, разделенных запятыми. Для того чтобы значение было разрешенным, каждый его элемент должен быть разрешен. Кроме того, поскольку «любое количество» включает в себя и «ноль», пустая строка является разрешенным значением для любого столбца SET.

Для разовой проверки отдельных значений ввода можно использовать служебную процедуру check_set_value(), напоминающую check_enum_value():

```
sub check_set_value
{
my ($dbh, $tbl_name, $col_name, $val) = @_;

    my $valid = 0;
    my $info = get_enumorset_info ($dbh, $tbl_name, $col_name);
    if ($info && $info->{type} eq "set")
    {
        return 1 if $val eq "";        # пустая строка - допустимый элемент
        # использовать нечувствительное к регистру сравнение
        # столбцы SET не чувствительны к регистру
        $valid = 1;         # считать разрешенным, пока не выявлено обратное
        foreach my $v (split (/,/, $val))
        {
            if (!grep (/^$v$/i, @{$info->{values}}))
            {
                $valid = 0; # значение содержит неразрешенный элемент
                last;
            }
        }
    }
    return ($valid);
}
```

Для массового тестирования создайте хеш из разрешенных членов SET. Процедура повторяет формирование хеша из элементов ENUM:

```
my $ref = get_enumorset_info ($dbh, $tbl_name, $col_name);
my %members;
foreach my $member (@{$ref->{values}})
{
    # преобразовать ключи хеша к одному регистру; SET не чувствителен к регистру
    $members{lc ($member)} = 1;
}
```

Чтобы сравнить вводимое значение с хешем членов SET, преобразуйте его к тому же регистру, что и ключи хеша, разделите на части по запятым, чтобы получить список отдельных элементов значения, и проверьте каждое из них. Если один из элементов не разрешен, не разрешено и все значение:

```
$valid = 1;         # считать разрешенным, пока не выявлено обратное
foreach my $elt (split (/,/, lc ($val)))
{
    if (!exists ($members{$elt}))
```

```
        {
            $valid = 0; # значение содержит неразрешенный элемент
            last;
        }
    }
```

После завершения цикла `$valid` — «истина», если значение допустимо для столбца `SET`, и «ложь» в противном случае. Пустые строки всегда разрешены для столбцов `SET`, но этот код не выполняет никаких специальных проверок на пустую строку. В этом нет необходимости, так как если операция `split()` возвращает пустой список, то цикл ни разу не выполняется, и значение `$valid` остается «истиной».

10.28. Проверка корректности при помощи справочной таблицы

Задача

Вам нужно убедиться в том, что вводимые значения имеются в справочной таблице.

Решение

Выполните запросы, чтобы проверить, содержатся ли значения в таблице. Конкретный способ выполнения зависит от количества вводимых значений и размера таблицы.

Обсуждение

Для того чтобы проверить соответствие вводимых значений содержимому справочной таблицы, можно использовать приемы, подобные представленным в рецепте 10.27 для тестирования столбцов `ENUM` и `SET`. Но если количество членов в столбцах `ENUM` и `SET` ограничено максимальным значением в 65 536 и 64 соответственно, справочная таблица может включать практически неограниченное количество записей. Вероятно, вы не захотите считывать их все в память.

Как будет показано ниже, сравнение вводимых значений с содержимым справочной таблицы можно выполнять различными способами. Приведенные в качестве примеров проверки выполняют точное сравнение с теми значениями, которые хранятся в таблице. Если вас интересуют сравнения, не чувствительные к регистру, не забудьте о преобразовании всех значений к одному регистру.

Создание отдельных запросов

Для одноразовых операций можно тестировать значение, проверяя, входит ли оно в справочную таблицу. Следующий запрос возвращает значение «истина» (не ноль), если значение найдено, и «ложь» в противном случае:

```
$valid = $dbh->selectrow_array (
                "SELECT COUNT(*) FROM $tbl_name WHERE val = ?",
                undef, $val);
```

Такая проверка подходит, например, для тестирования значения, переданного через веб-форму, но для больших объемов данных она не эффективна. Результаты уже выполненных проверок не запоминаются, так что придется выполнять запросы для каждого отдельного вводимого значения.

Формирование хеша из всей справочной таблицы

Если вы планируете проводить массовые проверки больших объемов данных, следует извлечь значения из справочной таблицы в память, сохранить их как структуру данных и сопоставлять каждое вводимое значение содержимому созданной структуры. Хранение справочника в памяти избавляет от необходимости выполнения запроса для каждого значения.

Сначала выполним запрос, извлекающий все значения справочной таблицы и сформируем из них хеш:

```
my %members;    # хеш для справочных значений
my $sth = $dbh->prepare ("SELECT val FROM $tbl_name");
$sth->execute ();
while (my ($val) = $sth->fetchrow_array ())
{
    $members{$val} = 1;
}
```

Использование справочных таблиц в других языках

Рассмотренный пример массовой проверки значений использует хеш языка Perl для определения того, содержится ли указанное значение в множестве:

```
$valid = exists ($members{$val});
```

Аналогичные структуры данных есть и в других языках. В PHP можно использовать ассоциативный массив и выполнять поиск ключа так:

```
$valid = isset ($members[$val]);
```

В Python используйте словарь и проверяйте значения ввода, применяя метод `has_key()`:

```
valid = members.has_key (val)
```

Для поиска в Java используйте `HashMap` и тестируйте значения при помощи метода `containsKey()`:

```
valid = members.containsKey (val);
```

Каталог *transfer* дистрибутива `recipes` содержит несколько примеров операций со справочными таблицами на каждом из рассматриваемых языков.

Затем проверим каждое значение, выполняя тест на существование ключа хеша:

```
$valid = exists ($members{$val});
```

Взаимодействие с базой данных сводится к единственному запросу. Однако если справочная таблица очень велика, то и такой объем трафика весьма велик, и вы можете не захотеть хранить всю таблицу в памяти.

Хеш как кэш для уже просмотренных справочных значений

Можно сочетать два описанных приема: индивидуальные запросы и использование хеша для хранения информации о существовании значения. Такой подход особенно удобен для очень больших справочных таблиц. Начнем с пустого хеша:

```
my %members;    # хеш для справочных значений
```

Затем для каждого проверяемого значения определяем, присутствует ли оно в хеше. Если нет, выдаем индивидуальный запрос, чтобы проверить, содержится ли оно в справочной таблице, и записываем результат запроса в хеш. Допустимость значения ввода определяется значением, сопоставленным ключу, а не существованием ключа:

```
if (!exists ($members{$val}))   # такое значение еще не встречалось
{
    my $count = $dbh->selectrow_array (
                    "SELECT COUNT(*) FROM $tbl_name WHERE val = ?",
                    undef, $val);
    # хранить истину/ложь как признак найденного значения
    $members{$val} = ($count > 0);
}
$valid = $members{$val};
```

В данном случае хеш действует как кэш, так что запрос для каждого определенного значения выполняется только единожды, вне зависимости от того, сколько раз оно присутствует во вводе. Для множеств данных с достаточно большим количеством повторяющихся значений этот метод позволяет избежать запуска отдельных запросов для каждого значения, так как в хеше нужна запись только для каждого уникального значения. Данный подход представляет собой компромисс между объемом данных, извлекаемых из базы данных, и объемом памяти, необходимой для хранения хеша.

Обратите внимание на то, что хеш используется не совсем так, как в предыдущем случае. Раньше корректность значения определялась существованием вводимого значения как ключа хеша, а сопоставленное ключу значение было несущественным. А для случая «хеш как кэш» наличие ключа хеша означает не «значение разрешено», а «значение уже проверено». Для каждого ключа сопоставленное ему значение показывает, присутствует ли вводимое значение в справочной таблице. (Если хранить как ключи только значения, найденные в справочной таблице, то придется выполнять запрос для

каждого экземпляра недопустимого значения входных данных, что неэффективно.)

10.29. Преобразование двузначных значений года в четырехзначные

Задача

Вам нужно преобразовать составляющую года в значениях дат из двузначной в четырехзначную.

Решение

Доверьтесь MySQL. Если же правила преобразования MySQL вас не устраивают, выполните операцию самостоятельно.

Обсуждение

Двузначные значения года создают проблемы, потому что в значениях дат явно не указывается век. Если вам известен диапазон годов, охватываемый вводом, вы можете однозначно определить и добавить к значениям век. В противном случае остается только гадать. Например, большинство американцев, вероятно, восприняло бы дату 2/10/69 как 2 октября 1969 года. Но если речь идет о дне рождения Махатмы Ганди, то имеется в виду 1869 год.

Одним из способов преобразования годов к четырехзначному виду является использование возможностей MySQL. Если вы сохраняете дату, содержащую двузначный год, то MySQL автоматически преобразует его в четырехзначную форму. Для MySQL переходной точкой является 1970 год; значения от 00 до 69 интерпретируются как годы с 2000 по 2069, а значения от 70 до 99 – как годы с 1970 по 1999. Такие правила верны для значений годов в диапазоне от 1970 до 2069. Если ваши значения выходят за рамки данного диапазона, вам следует добавлять соответствующий век самостоятельно перед сохранением значений в MySQL.

Чтобы использовать другую точку перехода, преобразуйте годы к четырехзначному формату вручную. Напишем универсальную программу, которая преобразует двузначные значения года в четырехзначные, поддерживая произвольную точку перехода:

```
sub yy_to_ccyy
{
my ($year, $transition_point) = @_;

    $transition_point = 70 unless defined ($transition_point);
    $year += ($year >= $transition_point ? 1900 : 2000) if $year < 100;
    return ($year);
}
```

По умолчанию эта функция использует точку перехода MySQL (70). Можно задать необязательный второй аргумент для определения другой точки пере-

хода. Перед выполнением преобразования функция yy_to_ccyy() проверяет, действительно ли оно необходимо (меньше ли значение, чем 100). Поэтому вы можете передавать ей значения, не проверяя предварительно, содержат ли они значение века. Вызовы с точкой перехода по умолчанию имеют такие результаты:

```
$val = yy_to_ccyy (60);          # возвращает 2060
$val = yy_to_ccyy (1960);        # возвращает 1960 (преобразование не проводилось)
```

Но предположим, что вы хотите преобразовать значения годов следующим образом, используя в качестве точки перехода 50:

```
00 .. 49 -> 2000 .. 2049
50 .. 99 -> 1950 .. 1999
```

Передайте аргумент точки перехода в yy_to_ccyy():

```
$val = yy_to_ccyy (60, 50);      # возвращает 1960
$val = yy_to_ccyy (1960, 50);    # возвращает 1960 (преобразование не проводилось)
```

Функция yy_to_ccyy() включена в библиотечный файл *Cookbook_Utils.pm*.

10.30. Проверка корректности составляющих даты и времени

Задача

Строка сравнивалась с образцом и оказалась похожа на дату или время, но вы хотите выполнить дополнительную проверку, чтобы убедиться в том, что строка допустима.

Решение

Разбейте значение на части и выполните для каждой составляющей проверку на вхождение в соответствующий диапазон.

Обсуждение

Сравнения с образцом может оказаться недостаточно для проверки значений дат и времени. Например, значение типа 1947-15-19 может соответствовать образцу даты, но если вставить такое значение в столбец DATE, MySQL преобразует его в 0000-00-00. Если вы хотите узнать, корректно ли значение, до того, как оно попадет в базу данных, используйте сравнение с образцом в сочетании с проверкой на принадлежность диапазону.

Чтобы убедиться в правильности даты, разбейте ее на составляющие года, месяца и дня и проверьте, входят ли они в соответствующие диапазоны. Год должен быть меньше 9999 (в MySQL верхняя граница дат равна 9999-12-31), значение месяца должно быть от 1 до 12, а дни должны входить в диапазон от 1 до количества дней в данном месяце. Последнее условие несколько сложнее, так как зависит от месяца, кроме того, для февральских дат оно зависит еще и от года (високосный он или нет).

Предположим, что вы проверяете входные даты в формате ISO. Ранее в рецепте 10.25 мы использовали функцию `is_iso_date()` из библиотечного файла *Cookbook_Utils.pm* для выполнения сравнения строки даты с образцом и разбиения ее на составляющие:

```
my $ref = is_iso_date ($val);
if (defined ($ref))
{
    # $val соответствует образцу формата ISO;
    # проверять составляющие, используя $ref->[0] - $ref->[2]
}
else
{
    # $val не соответствует образцу формата ISO
}
```

Функция `is_iso_date()` возвращает `undef`, если значение не совпадает с образцом формата даты ISO. В противном случае она возвращает ссылку на массив, содержащий значения года, месяца и дня.[1] Чтобы выполнить дополнительную проверку составляющих даты, передайте их другой библиотечной функции, `is_valid_date()`:

```
$valid = is_valid_date ($ref->[0], $ref->[1], $ref->[2]);
```

Или, если короче:

```
$valid = is_valid_date (@{$ref});
```

Функция `is_valid_date()` проверяет составляющие даты так:

```
sub is_valid_date
{
my ($year, $month, $day) = @_;

    # год должен быть неотрицательным, месяц и год - положительными
    return (0) if $year < 0 || $month < 1 || $day < 1;
    # проверка допустимых границ составляющих
    return (0) if $year > 9999;
    return (0) if $month > 12;
    return (0) if $day > days_in_month ($year, $month);
    return (1);
}
```

Для `is_valid_date()` требуется не строка даты, а отдельные значения года, месяца и дня. Поэтому прежде чем вызывать функцию, вам необходимо разбить тестируемые значения на составляющие. Такое требование делает функцию более универсальной. Например, ее можно использовать для проверки

[1] Файл *Cookbook_Utils.pm* также содержит программы `is_mmddyy_date()` и `is_ddmmyy_date()`, проверяющие соответствие дат формату, принятому в США и Великобритании, и возвращающие `undef` или ссылку на массив составляющих даты. (Составляющие всегда расположены в порядке «год, месяц, день» вне зависимости от того, в каком порядке они были указаны в строке даты.)

дат типа 12 February 2003, если перед вызовом is_valid_date() сопоставить названию месяца его номер. Если бы is_valid_date() принимала строковый аргумент, он должен был бы иметь какой-то предопределенный формат, и функция стала бы более узко специализированной.

Для вычисления количества дней месяца, представленного датой, функция is_valid_date() применяет вспомогательную функцию days_in_month(). Функция days_in_month() требует в качестве аргументов и год, и месяц, так как если месяц – февраль (2), то количество его дней зависит от того, високосный ли год. То есть *необходимо* передавать значение года в четырехзначном формате. Корректно определить високосный год в двузначном значении невозможно, как это было показано в рецепте 5.27. Функции days_in_month() и is_leap_year() базируются на приемах, взятых непосредственно оттуда:

```
sub is_leap_year
{
my $year = shift;

    return (($year % 4 == 0) && ((($year % 100) != 0) || ($year % 400) == 0));
}

sub days_in_month
{
my ($year, $month) = @_;
my @day_tbl = (31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31);
my $days = $day_tbl[$month-1];

    # добавить один день для февраля високосного года
    $days++ if $month == 2 && is_leap_year ($year);
    return ($days);
}
```

Чтобы выполнить проверку корректности значений времени, можно использовать аналогичную процедуру, изменив диапазоны для составляющих: от 0 до 24 для часов и от 0 до 59 для минут и секунд. Функция is_24hr_time() преобразует значения в 24-часовой формат:

```
sub is_24hr_time
{
my $s = shift;

    return undef unless $s =~ /^(\d{1,2})\D(\d{2})\D(\d{2})$/;
    return [ $1, $2, $3 ];  # вернуть часы, минуты, секунды
}
```

Следующая функция is_ampm_time() ищет значения времени в 12-часовом формате с необязательным суффиксом AM или PM, преобразуя время после полудня (PM) к 24-часовому формату:

```
sub is_ampm_time
{
my $s = shift;

    return undef unless $s =~ /^(\d{1,2})\D(\d{2})\D(\d{2})(?:\s*(AM|PM))?$/i;
```

```
    my ($hour, $min, $sec) = ($1, $2, $3);
    $hour += 12 if defined ($4) && uc ($4) eq "PM";
    return [ $hour, $min, $sec ];   # вернуть часы, минуты, секунды
}
```

Обе функции возвращают `undef` для значений, не совпадающих с образцом. В противном случае они возвращают ссылку на трехэлементный массив, содержащий часы, минуты и секунды.

10.31. Создание утилит для обработки дат

Задача

Вам часто приходится выполнять определенную операцию обработки дат, поэтому вы хотите написать утилиту, которая делала бы это за вас.

Решение

В этом разделе приведено несколько примеров таких утилит.

Обсуждение

Даты бывают очень разными, поэтому вам, вероятно, периодически придется писать конвертеры, преобразующие даты. В разделе приведено несколько конвертеров, выполняющих разнообразные операции:

- *isoize_date.pl* читает файл, отыскивая даты в формате, принятом в США (*MM-DD-YY*) и преобразует их в формат ISO.
- *cvt_date.pl* преобразует даты в/из формата ISO и форматов дат, принятых в США и Великобритании. Это более универсальная программа, чем *isoize_date.pl*, но она требует указать, какой ввод вы ожидаете получить и какой вывод следует формировать.
- *monddccyy_to_iso.pl* ищет даты типа `Feb. 6, 1788` и преобразует их в формат ISO. Программа показывает, как отображать даты с нечисловыми составляющими в формат, понятный MySQL.

Все три сценария находятся в каталоге *transfer* дистрибутива `recipes`. Они предполагают, что элементы файлов разделены символами табуляции, а признаком конца строки является символ перевода строки. (Используйте *cvt_file.pl* для работы с файлами в другом формате.)

Наша первая утилита обработки дат, *isoize_date.pl*, ищет даты в формате, принятом в США, и преобразует их в формат ISO. Вы должны заметить, что в ее основе лежит общий цикл обработки ввода из рецепта 10.20, к которому добавлено немного кода для выполнения конкретного преобразования:

```
#! /usr/bin/perl -w
# isoize_date.pl - Читает входные данные, ищет значения, соответствующие образцу
# даты, преобразует их в формат ISO. Кроме того преобразует двузначные года
# в четырехзначные, используя точку перехода 70.

# По умолчанию ищет даты в формате MM-DD-[CC]YY.
```

```
# Предполагает, что строки ввода разделены символами табуляции,
# признак конца строки – перевод строки.

# Не проверяет, действительно ли допустимы даты
# (например, не будет жаловаться на значение 13-49-1928).

use strict;

# точка перехода, в которой двузначные года рассматриваются как 19XX
# (более ранние рассматриваются как 20XX)
my $transition = 70;

while (<>)
{
    chomp;
    my @val = split (/\t/, $_, 10000);  # разбить, сохраняя все поля
    for my $i (0 .. @val - 1)
    {
        my $val = $val[$i];
        # искать строки в формате MM-DD-[CC]YY
        next unless $val =~ /^(\d{1,2})\D(\d{1,2})\D(\d{2,4})$/;

        my ($month, $day, $year) = ($1, $2, $3);
        # чтобы интерпретировать даты как DD-MM-[CC]YY,
        # заменить предыдущую строку следующей:
        #my ($day, $month, $year) = ($1, $2, $3);

        # преобразовать двузначный год в четырехзначный,
        # затем обновить значение в массиве
        $year += ($year >= $transition ? 1900 : 2000) if $year < 100;
        $val[$i] = sprintf ("%04d-%02d-%02d", $year, $month, $day);
    }
    print join ("\t", @val) . "\n";
}

exit (0);
```

Если вы подаете на вход *isoize_date.pl* такой файл:

```
Fred    04-13-70
Mort    09-30-69
Brit    12-01-57
Carl    11-02-73
Sean    07-04-63
Alan    02-14-65
Mara    09-17-68
Shepard 09-02-75
Dick    08-20-52
Tony    05-01-60
```

то будет сформирован следующий вывод:

```
Fred    1970-04-13
Mort    2069-09-30
Brit    2057-12-01
Carl    1973-11-02
Sean    2063-07-04
```

```
Alan    2065-02-14
Mara    2068-09-17
Shepard 1975-09-02
Dick    2052-08-20
Tony    2060-05-01
```

Сценарий *isoize_date.pl* имеет специальное назначение. Он конвертирует только формат дат США в формат ISO. Он не выполняет проверку корректности составляющих даты и не позволяет задать точку перехода для добавления века. Более универсальное средство было бы более полезным. Следующий сценарий, *cvt_date.pl*, расширяет возможности *isoize_date.pl*; он распознает даты в формате ISO, США и Великобритании и преобразует любой из этих форматов в любой другой. Кроме того, сценарий конвертирует двузначные года в четырехзначные, позволяя указать точку перехода, и может выводить предупреждения о неподходящих датах. Поэтому его можно использовать для предварительной обработки ввода перед загрузкой в MySQL или для пост-обработки данных, экспортированных из MySQL для использования в других программах.

Сценарий *cvt_date.pl* распознает следующие опции:

--iformat=формат, --oformat=формат, --format=формат,

Установить формат даты для ввода, вывода или ввода и вывода. Значение *формат* по умолчанию – `iso`; кроме того, *cvt_date.pl* воспринимает любую строку, начинающуюся с `us` или `br`, как задающую формат даты США или Великобритании соответственно.

--add-century

Преобразовать двузначные значения года в четырехзначные.

--columns=список_столбцов

Преобразовать даты только в указанных столбцах. По умолчанию *cvt_date.pl* ищет даты во всех столбцах. Если опция указана, то *список_столбцов* – это список из одной или более позиций столбцов, разделенных запятыми. Нумерация позиций начинается с 1.

--transition=n

Установить точку перехода для преобразования двузначных значений годов в четырехзначные. Значение по умолчанию равно 70. Эта опция включает *--add-century*.

--warn

Предупреждает о плохих датах. (Обратите внимание на то, что эта опция может выводить ложные предупреждения, если даты имеют двузначные составляющие года, а опция *--add-century* не указана, так как в этом случае проверка на високосный год не всегда будет корректной.)

Я не буду приводить здесь код *cvt_date.pl* (его большая часть посвящена обработке опций командной строки), если хотите, можете сами просмотреть исходные тексты. Для того чтобы посмотреть, как *cvt_date.pl* работает, предположим, что у вас есть файл *newdata.txt* с таким содержимым:

```
name1   01/01/99    38
name2   12/31/00    40
name3   02/28/01    42
name4   01/02/03    44
```

Запустим для этого файла *cvt_date.pl* с опциями, указывающими, что даты должны быть в формате США, а к годам должен быть добавлен век:

```
% cvt_date.pl --iformat=us --add-century newdata.txt
name1   1999-01-01  38
name2   2000-12-31  40
name3   2001-02-28  42
name4   2003-01-02  44
```

Чтобы вывести даты в формате, принятом в Великобритании, без преобразования года:

```
% cvt_date.pl --iformat=us --oformat=br newdata.txt
name1   01-01-99    38
name2   31-12-00    40
name3   28-02-01    42
name4   02-01-03    44
```

Сценарий *cvt_date.pl* не имеет информации о содержимом каждого столбца. Если у вас есть столбец со значениями, которые соответствуют образцу, но не являются датами, он перезапишет и этот столбец. Чтобы избежать подобных ситуаций, используйте опцию *--columns* для определения того, на какие столбцы должен воздействовать *cvt_date.pl*.

Сценарии *isoize_date.pl* и *cvt_date.pl* работают с датами, записанными в любом числовом формате. Но в файлах данных даты часто представлены по-другому, и может потребоваться создание специального сценария для их обработки. Предположим, что файл ввода содержит даты в следующем формате (это даты вступления отдельных штатов в США):

```
Delaware        Dec. 7, 1787
Pennsylvania    Dec 12, 1787
New Jersey      Dec. 18, 1787
Georgia         Jan. 2, 1788
Connecticut     Jan. 9, 1788
Massachusetts   Feb. 6, 1788
Maryland        Apr. 28, 1788
South Carolina  May 23, 1788
New Hampshire   Jun. 21, 1788
Virginia        Jun 25, 1788
...
```

Даты состоят из трехсимвольного сокращенного названия месяца (за которым может следовать точка), дня месяца в числовом формате, запятой и года, тоже в числовом формате. Чтобы импортировать такой файл в MySQL, необходимо преобразовать даты в формат ISO и получить такой файл:

```
Delaware        1787-12-07
Pennsylvania    1787-12-12
```

```
New Jersey      1787-12-18
Georgia         1788-01-02
Connecticut     1788-01-09
Massachusetts   1788-02-06
Maryland        1788-04-28
South Carolina  1788-05-23
New Hampshire   1788-06-21
Virginia        1788-06-25
...
```

Это весьма специфическое преобразование, хотя тип задачи (преобразование специального формата даты) достаточно распространен. Чтобы выполнить преобразование, идентифицируйте даты как значения, совпадающие с соответствующим образцом, затем сопоставьте названиям месяцев их номера и переформатируйте результат. Рассмотрим сценарий *monddccyy_to_iso.pl*, который показывает, как все это сделать:

```
#! /usr/bin/perl -w
# monddccyy_to_iso.pl - преобразование дат из формата mon[.] dd, ccyy в ISO

# Предполагается, что строки ввода разделены символами табуляции,
# а признак конца строки - перевод строки.

use strict;

my %map =       # сопоставить названию месяца его номер
(
    "jan" => 1, "feb" => 2, "mar" => 3, "apr" => 4, "may" => 5, "jun" => 6,
    "jul" => 7, "aug" => 8, "sep" => 9, "oct" => 10, "nov" => 11, "dec" => 12
);

while (<>)
{
    chomp;
    my @val = split (/\t/, $_, 10000);       # разбить на части, сохраняя все поля
    for my $i (0 .. @val - 1)
    {
        # переформатировать значение, если оно совпадает с образцом, иначе считать,
        # что это не дата в нужном формате и не обрабатывать ее
        if ($val[$i] =~ /^([^.]+)\.? (\d+), (\d+)$/)
        {
            # использовать название месяца в нижнем регистре
            my ($month, $day, $year) = (lc ($1), $2, $3);
            if (exists ($map{$month}))
            {
                $val[$i] = sprintf ("%04d-%02d-%02d", $year, $map{$month}, $day);
            }
            else
            {
                # предупреждать, но не переформатировать
                warn "$val[$i]: bad date?\n";
            }
        }
    }
```

```
    print join ("\t", @val) . "\n";
}
exit (0);
```

Сценарий выполняет только переформатирование, не проверяя корректность дат. Для проверки корректности сделайте так, чтобы сценарий использовал модуль *Cookbook_Utils.pm*: добавьте после строки `use strict` такое предложение:

```
use Cookbook_Utils;
```

Тогда сценарий получит доступ к программе модуля – `is_valid_date()`. Чтобы использовать ее, измените раздел сценария, в котором выполняется переформатирование, следующим образом:

```
if (exists ($map{$month})
        && is_valid_date ($year, $map{$month}, $day))
{
    $val[$i] = sprintf ("%04d-%02d-%02d",
                        $year, $map{$month}, $day);
}
else
{
    # предупреждать, но не переформатировать
    warn "$val[$i]: bad date?\n";
}
```

10.32. Использование дат с недостающими частями

Задача

Даты в вашем файле данных являются неполными, то есть в них отсутствуют какие-то составляющие.

Решение

MySQL может представить их в формате ISO, используя для недостающих составляющих нули.

Обсуждение

Некоторые приложения используют неполные даты. Например, вам может потребоваться обработка таких значений ввода, как `Mar/2001`, которые содержат только месяц и день. Начиная с версии MySQL 3.23 можно представлять такие значения датами в формате ISO, которые содержат нули вместо недостающих составляющих. (Значение `Mar/2001` может храниться как `2001-03-00`.) Чтобы преобразовать значения месяц/год в формат ISO для импортирования в MySQL, создайте хеш для сопоставления названиям месяцев их числовых значений:

```
my %map =       # сопоставить названию месяца его номер
(
    "jan" => 1, "feb" => 2, "mar" => 3, "apr" => 4, "may" => 5, "jun" => 6,
    "jul" => 7, "aug" => 8, "sep" => 9, "oct" => 10, "nov" => 11, "dec" => 12
);
```

Тогда каждое значение ввода будет преобразовано следующим образом:

```
if ($val =~ /^([a-z]{3})\/(\d{4})$/i)
{
    my ($m, $y) = (lc ($1), $2); # использовать название месяца в нижнем регистре
    $val = sprintf ("%04d-%02d-00", $y, $map{$m})
}
```

После сохранения результирующих значений в MySQL можно извлекать их для отображения в исходном формате месяц/год, выполняя предложение `SELECT`, форматирующее даты при помощи выражения `DATE_FORMAT()`:

```
DATE_FORMAT(date_val,'%b/%Y')
```

10.33. Преобразование дат при помощи SQL

Задача

Вы хотите преобразовать даты, используя предложения SQL.

Решение

При экспорте используйте функцию `DATE_FORMAT()` для преобразования значений. При импорте считайте значения в строковый столбец и преобразуйте в настоящие значения `DATE`.

Обсуждение

Предположим, что вы хотите экспортировать данные из MySQL в приложение, которое не понимает даты в формате ISO. Можно экспортировать данные в файл, оставляя даты в формате ISO, затем пропустить этот файл через утилиту типа *cvt_date.pl* для преобразования значений в нужный формат даты.

Есть и другой способ – экспортировать даты непосредственно в требуемый формат, преобразуя их посредством `DATE_FORMAT()`. Предположим, что вам нужно экспортировать данные из таблицы, при этом даты должны быть в формате США (*MM-DD-CCYY*). Ниже приведен сценарий, выполняющий такую операцию. Он принимает в качестве аргументов имена базы данных и таблицы, затем выгружает таблицу в формате значений, разделенных символами табуляции, с переформатированием столбцов `DATE`, `DATETIME` и `TIMESTAMP`. Сценарий исследует метаданные таблицы для получения типов столбцов, затем формирует предложение `SELECT`, использующее функцию `DATE_FORMAT()` для форматирования дат. Другие столбцы таблицы записываются без изменения:

```
#! /usr/bin/perl -w
# iso_to_us.pl - Экспорт таблицы с датами, преобразованными из формата ISO
```

```
# (CCYY-MM-DD) в формат США (MM-DD-CCYY). Для этого формируется предложение SELECT,
# выбирающее все столбцы таблицы и использующее DATE_FORMAT() для перезаписи дат.

# Записывает каждую строку как значения, разделенные символами табуляции,
# признак конца строки – символ перевода строки.

use strict;
use DBI;

# ...обработка опций командной строки (не приводится) ...

@ARGV == 2 or die "Usage: $0 [options] db_name tbl_name\n";
my $db_name = shift (@ARGV);
my $tbl_name = shift (@ARGV);

# ... соединение с базой данных (не приводится) ...

# Читать из MySQL метаданные таблицы для получения имен и типов столбцов.
# Информация о типах используется для выявления столбцов DATE, DATETIME и TIMESTAMP
# и перезаписи их содержимого с помощью DATE_FORMAT().

my @col;

my $sth = $dbh->prepare ("SHOW COLUMNS FROM $tbl_name");
$sth->execute ();
while (my @row = $sth->fetchrow_array ())
{
    if ($row[1] =~ /^datetime|timestamp/)
    {
        $row[0] = "DATE_FORMAT($row[0], '%m-%d-%Y %T') AS $row[0]";
    }
    elsif ($row[1] =~ /^date/)
    {
        $row[0] = "DATE_FORMAT($row[0], '%m-%d-%Y') AS $row[0]";
    }
    push (@col, $row[0]);
}
my $query = "SELECT\n\t" . join (",\n\t", @col) . "\nFROM $tbl_name";

# Выполнить предложение SELECT и выгрузить результат

$sth = $dbh->prepare ($query);
$sth->execute ();
while (my @val = $sth->fetchrow_array ())
{
    # преобразовать значения NULL (undef) в пустые строки
    @val = map { defined ($_) ? $_ : "" } @val;
    print join ("\t", @val) . "\n";
}

$dbh->disconnect ();

exit (0);
```

Чтобы посмотреть, как работает сценарий, создадим такую таблицу:

```
CREATE TABLE datetbl
(
```

```
    i   INT,
    c   CHAR(10),
    d   DATE,
    dt  DATETIME,
    ts  TIMESTAMP
);
```

Для экспорта содержимого `datetbl` сценарий формирует такое предложение `SELECT`:

```
SELECT
    i,
    c,
    DATE_FORMAT(d, '%m-%d-%Y') AS d,
    DATE_FORMAT(dt, '%m-%d-%Y %T') AS dt,
    DATE_FORMAT(ts, '%m-%d-%Y %T') AS ts
FROM datetbl
```

То есть если `datetbl` содержит следующие строки:

```
3       abc     2001-12-31      2001-12-31 12:05:03     20011231120503
4       xyz     2002-01-31      2002-01-31 12:05:03     20020131120503
```

Сценарий генерирует такой вывод:

```
3       abc     12-31-2001      12-31-2001 12:05:03     12-31-2001 12:05:03
4       xyz     01-31-2002      01-31-2002 12:05:03     01-31-2002 12:05:03
```

При импортировании дат не-ISO в MySQL обычно сначала выполняется преобразование в формат ISO. Иначе пришлось бы импортировать их как символьные строки, что делает невозможным их использование во временном контексте. Однако в некоторых случаях можно импортировать даты не-ISO как строки, а затем преобразовать их в значения `DATE` стандарта ISO. Подобный пример рассмотрен в рецепте 10.34.

См. также

Одна из разновидностей перезаписи дат при экспорте применяется в рецепте 10.40, в котором описан сценарий *mysql_to_filemaker.pl*, экспортирующий таблицы MySQL для использования в FileMaker Pro. Сценарий использует `DATE_FORMAT()` для преобразования дат к формату *MM-DD-CCYY*, воспринимаемому FileMaker Pro. Кроме того, функция `DATE_FORMAT()` используется и для разбиения значений дата-и-время на отдельные столбцы даты и времени, так как в FileMaker Pro нет аналога столбцов `DATETIME` и `TIMESTAMP` MySQL.

10.34. Использование временных таблиц для преобразования дат

Задача

Вы хотите выполнить предварительную обработку входных данных для MySQL, но у вас нет доступа к соответствующим внешним утилитам.

Решение

Загрузите данные во временную таблицу, переформатируйте ее, используя предложения SQL, затем скопируйте записи в таблицу назначения.

Обсуждение

Для работы с информацией, которую необходимо проверить или преобразовать перед добавлением в таблицу, часто бывает полезно сначала загрузить файл данных во временную таблицу для проверки корректности (обычно проще обрабатывать множество данных, которое изолировано в отдельной таблице, а не перемешано с другими записями). После того, как вы убедитесь в том, что содержимое временной таблицы вас удовлетворяет, скопируйте ее строки в основную таблицу и удалите временную. (Надо сказать, что «временность» таблицы не обязательно подразумевает использование ключевого слова `TEMPORARY` при ее создании.[1] Если вы обрабатываете таблицу в несколько этапов в рамках нескольких соединений с сервером, то нужно будет создать таблицу не-`TEMPORARY`, а затем явно удалить ее, когда работа с ней будет закончена.)

Следующий пример показывает, как использовать временную таблицу для решения общей проблемы: загрузки в таблицу данных, формат значений которых не соответствует структуре таблицы. Предположим, что у вас есть таблица `main`, содержащая три столбца `name`, `date` и `value`, где `date` – столбец типа `DATE` со значениями в формате ISO (*CCYY-MM-DD*). Предположим также, что у вас есть файл данных *newdata.txt*, который нужно импортировать в таблицу, а его содержимое таково:

```
name1   01/01/99    38
name2   12/31/00    40
name3   02/28/01    42
name4   01/02/03    44
```

Даты имеют формат *MM/DD/YY* и должны быть преобразованы к формату ISO для хранения в виде значений `DATE` в MySQL. Можно применить к файлу рассмотренный ранее в главе сценарий *cvt_date.pl*:

```
% cvt_date.pl --iformat=us --add-century newdata.txt >tmp
```

Затем можно загружать файл *tmp* в таблицу `main`. Но можно выполнить задачу целиком в MySQL, не обращаясь к внешним утилитам, а импортируя данные во временную таблицу и используя предложения SQL для выполнения следующих операций:

1. Создать пустую таблицу, в которую будут загружаться тестовые данные. Следующие предложения создают таблицу `tmp` как пустую копию таблицы `main`, к которой добавлен столбец `cdate` для хранения дат файла данных в виде символьных строк:

[1] Предложение `CREATE TEMPORARY TABLE` обсуждалось в рецепте 3.24.

```
mysql> CREATE TABLE tmp SELECT * FROM main WHERE 1 < 0;
mysql> ALTER TABLE tmp ADD cdate CHAR(8);
```

2. Загрузить файл данных во временную таблицу, сохраняя значения данных в столбце `cdate`, а не `date`:

```
mysql> LOAD DATA LOCAL INFILE 'newdata.txt' INTO TABLE tmp (name,cdate,value);
```

3. Преобразовать значения `cdate` из формата *MM/DD/YY* в *YY-MM-DD* и сохранить результаты в столбце `date`:

```
mysql> UPDATE tmp
    -> SET date = CONCAT(RIGHT(cdate,2),'-',LEFT(cdate,2),'-',MID(cdate,4,2));
```

MySQL автоматически преобразует двузначные значения года в четырехзначные, так что исходные значения *MM/DD/YY* столбца `cdate` превратятся в столбце `date` в значения ISO в формате *CCYY-MM-DD*. Следующий запрос показывает, как выглядят исходные значения `cdate` и преобразованные значения `date` после выполнения предложения `UPDATE`:

```
mysql> SELECT cdate, date FROM tmp;
+----------+------------+
| cdate    | date       |
+----------+------------+
| 01/01/99 | 1999-01-01 |
| 12/31/00 | 2000-12-31 |
| 02/28/01 | 2001-02-28 |
| 01/02/03 | 2003-01-02 |
+----------+------------+
```

4. Наконец, скопировать записи из таблицы `tmp` в `main` (используя преобразованные значения дат, а не исходные значения `cdate`) и удалить временную таблицу:

```
mysql> INSERT INTO main (name, date, value)
    -> SELECT name, date, value FROM tmp;
mysql> DROP TABLE tmp;
```

Предполагается, что автоматическое преобразование MySQL двузначных значений года в четырехзначные выводит корректные значения века. То есть составляющая года должна соответствовать годам из диапазона с 1970 по 2069 год. Если это не так, необходимо выполнить преобразование года каким-то другим способом (см. рецепт 10.29).

Кроме того, предполагается, что значения `cdate` всегда состоят ровно из восьми символов, поэтому для извлечения составляющих можно использовать функции `LEFT()`, `MID()` и `RIGHT()`. Если такое предположение неверно, необходимо изменить процедуру конвертации. Можно, например, применить `SUBSTRING_INDEX()` для разбиения строк на части по разделителям /:

```
mysql> UPDATE tmp
    -> SET date =
    -> CONCAT(SUBSTRING_INDEX(cdate,'/',-1),'-',
    -> SUBSTRING_INDEX(cdate,'/',1),'-',
    -> SUBSTRING_INDEX(SUBSTRING_INDEX(cdate,'/',2),'/',-1));
```

Еще одним приложением обработки данных после импортирования является разбиение имен. Если вы импортируете значения, включающие имя, пробел и фамилию, в столбец `full_name`, то можете разбить его на два отдельных столбца `first_name` и `last_name` при помощи таких предложений:

```
UPDATE имя_таблицы SET first_name = SUBSTRING_INDEX(full_name,' ',1);
UPDATE имя_таблицы SET last_name = SUBSTRING_INDEX(full_name,' ',-1);
```

Задача может усложниться, если любое из имен содержит инициалы или слова типа `Jr.` и `Sr`. Если дело обстоит именно так, лучше выполнить предварительную обработку имен до импортирования, используя утилиту сравнения с образцом, которая лучше умеет разбивать полные имена на составляющие.

10.35. Обработка значений NULL

Задача

Вы не знаете, как представить значения `NULL` в файле данных.

Решение

Попробуйте использовать значение, которое точно не присутствует в файле, чтобы отличать `NULL` от всех других допустимых значений не-`NULL`.

Обсуждение

Отсутствие стандарта для представления значений `NULL` в файлах данных создает некоторые проблемы при импорте и экспорте. Они объясняются тем, что `NULL` означает *отсутствие* значения, что не так-то легко буквально отобразить в файле данных. Первое, что приходит на ум, – использовать пустое значение столбца, но тогда будет невозможно отличить в строковом столбце `NULL` от настоящей пустой строки. Пустые значения не очень удачны и для других типов столбцов. Например, если вы загружаете пустое значение в числовой столбец при помощи `LOAD DATA`, оно будет сохранено как `0`, а не `NULL`, и при вводе его не удастся отличить от настоящего `0`.

Обычно проблему решают, сопоставляя значениям `NULL` значение, про которое известно, что оно точно не встречается в файле данных. Именно так поступают предложение `LOAD DATA` и программа *mysqlimport*, используя `\N` как значение, которое принимается за `NULL`. Теперь, когда вы знаете это, можно преобразовать пустые поля в файле данных в значения `\N`, чтобы предложение `LOAD DATA` интерпретировало их как `NULL`. Давайте напишем сценарий, который выполняет такую операцию:

```
#! /usr/bin/perl -w
# empty_to_null.pl - Преобразование пустых полей ввода в \N.

# \N - это условное обозначение MySQL, используемое в предложении LOAD DATA
# для NULL. Пропускаем файл через сценарий и выводим в результат обозначение NULL,
# а не пустую строку.

# Предполагается, что строки ввода разделены символами табуляции,
```

```
# а признак конца строки – символ перевода строки.

use strict;

while (<>)
{
    chomp;
    my @val = split (/\t/, $_, 10000);  # разбить на части, сохраняя все поля
    # сопоставить пустым полям \N, вывести как строку с разделителями-табуляциями
    print join ("\t", map { /^$/ ? "\\N" : $_ } @val) . "\n";
}

exit (0);
```

Можно использовать сценарий так:

```
% empty_to_null.pl mytbl.txt > mytbl.txt2
% mysqlimport --local cookbook mytbl.txt2
```

Загрузка файла, к которому был применен сценарий *empty_to_null.pl*, часто дает лучшие результаты для столбцов, допускающих значения NULL. В табл. 10.5 приведено сравнение результатов использования LOAD DATA или *mysqlimport* для загрузки пустой строки и \N (NULL) в столбцы различных типов, которые могут содержать значения NULL:

Таблица 10.5. Столбцы, допускающие значения NULL

Тип столбца	Результат загрузки пустой строки	Результат загрузки \N
CHAR	Пустая строка	NULL
INT	0	NULL
DATE	0000-00-00	NULL

Но что будет, если загрузить \N вместо пустой строки в столбцы, определенные как NOT NULL? Как показывает табл. 10.6, ничего не изменится:

Таблица 10.6. Столбцы, не допускающие значения NULL

Тип столбца	Результат загрузки пустой строки	Результат загрузки \N
CHAR	Пустая строка	Пустая строка
INT	0	0
DATE	0000-00-00	0000-00-00

То есть не стоит тратить время и силы на создание более «умной» версии *empty_to_null.pl*, проверяющей структуру таблицы, в которую планируется загрузка данных, и преобразовывающей пустые строки в \N только для столбцов, допускающих использование значений NULL.

Если же вы хотите интерпретировать как NULL не только пустые значения, но и какие-то еще, или использовать разные правила для разных столбцов, то можно-таки усовершенствовать сценарий. Рассмотрим такой файл данных, *has_nulls.txt*:

```
str1    13
str2    0
Unknown 15
Unknown 0
```

Первый столбец содержит строки, а `Unknown` обозначает `NULL`. Второй столбец содержит целые значения, а `0` соответствует `NULL`. Что делать в такой ситуации? Для обработки подобных файлов каталог *transfer* дистрибутива `recipes` предлагает сценарий *to_null.pl*. Он поддерживает опции, позволяющие указать, какие столбцы следует просматривать и какие значения в них искать:

--columns=список_столбцов

Преобразовывать значения только в указанных столбцах. По умолчанию просматриваются все столбцы. Если опция указана, то *список_столбцов* – это список из одной или более позиций столбцов, разделенных запятыми. Нумерация позиций начинается с 1.

--null=значение

Интерпретировать *значение* как индикатор значения `NULL` и преобразовывать его вхождения в `\N`. По умолчанию преобразуются пустые значения, как в *empty_to_null.pl*.

--case-insensitive, *-i*

При поиске индикатора значений `NULL` выполнять нечувствительное к регистру сравнение.

Поскольку в файле данных *has_nulls.txt* присутствуют два разных индикатора значения `NULL`, необходимо обрабатывать его с помощью двух вызовов *to_null.pl*:

```
% to_null.pl --columns=1 --null=Unknown has_nulls.txt \
    | to_null.pl --columns=2 --null=0 > tmp
```

Результирующий файл *tmp* выглядит так:

```
str1    13
str2    \N
\N      15
\N      \N
```

В некоторых случаях можно не заниматься предварительной обработкой файла ввода, если у вас есть возможность обработки его после импортирования. Например, если файл данных содержит числовой столбец, в котором `-1` представляет значение `NULL`, вы вполне можете преобразовать все значения `-1` после загрузки файла, используя простое предложение `UPDATE`:

```
UPDATE имя_таблицы SET имя_столбца = NULL WHERE имя_столбца = -1;
```

Итак, мы обсудили интерпретацию значений `NULL` при импорте в MySQL. Пришло время подумать о том, что делать с значениями `NULL` при передаче данных в обратном направлении – из MySQL в другие программы, например:

- `SELECT INTO ... OUTFILE` выводит значения `NULL` как `\N`. Придерживается ли другая программа тех же правил? Если нет, необходимо преобразовать `\N` в нечто, что программа сможет распознать.
- Вы можете использовать *mysql* в пакетном режиме в качестве простого способа получения вывода, разделенного табуляциями (рецепт 10.13), но в этом случае значение `NULL` выводится как слово «NULL». Если это слово больше не встречается в выводе, можно выполнить его пост-обработку и преобразовать все вхождения слова «NULL» во что-то более подходящее. Можно написать простой сценарий, аналогичный *empty_to_null.pl*, или использовать однострочную команду *sed*:

  ```
  % sed -e "s/NULL/\\N/g" data.txt > tmp
  ```

  Если же слово «NULL» может присутствовать в выводе, представляя нечто отличное от значения `NULL`, возникает неоднозначность, и, вероятно, вам придется использовать другой способ экспорта данных.

10.36. Определение структуры таблицы для файла данных

Задача

Вам дают файл с данными и говорят: «Вот файл, перенеси его в MySQL». Таблицы для хранения данных еще нет.

Решение

Напишите самостоятельно предложение `CREATE TABLE`. Или используйте утилиту, которая определяет структуру таблицы, исследуя содержимое файла данных.

Обсуждение

Иногда бывает так, что вам нужно импортировать данные в MySQL, а таблицы для них еще нет. Вы можете сами создать таблицу, основываясь на имеющейся информации о содержимом файла. Или же можете доверить часть работы утилите *guess_table.pl* из каталога *transfer* дистрибутива `recipes`. Утилита *guess_table.pl* читает файл данных, чтобы понять, какую информацию он содержит, затем пытается сформировать предложение `CREATE TABLE`, соответствующее содержимому файла. Конечно же, такой сценарий далек от совершенства, так как содержимое столбцов может быть неоднозначным. Например, столбец, содержащий небольшое количество различных строк, может относиться к типу как `CHAR`, так и `ENUM`. Но все же проще несколько подправить предложение, формируемое *guess_table.pl*, чем писать с нуля собственное. Утилиту можно использовать и для диагностики, хотя это не главное ее назначение. Например, вам может казаться, что столбец содержит только числа, но если *guess_table.pl* предложит для него тип `CHAR`, это будет означать, что столбец содержит хотя бы одно нечисловое значение.

Утилита *guess_table.pl* работает в предположении, что ее ввод разделен символами табуляции, а признаком конца строки является символ перевода строки. Она также считает, что вводятся разрешенные значения, так как любая попытка определения типа столбца на основе некорректных данных обречена на провал. То есть, например, для того, чтобы столбец дат был распознан как таковой, даты должны быть в формате ISO. В противном случае *guess_table.pl* может определить столбец как имеющий тип `CHAR`. Если для файла данных не выполнены описанные условия, можно предварительно переформатировать его при помощи утилит *cvt_file.pl* и *cvt_date.pl* из рецептов 10.18 и 10.31.

Утилита *guess_table.pl* понимает такие опции:

--labels

Интерпретировать первую строку ввода как строку заголовков столбцов и использовать их для именования столбцов таблицы. Если опция не задана, то *guess_table.pl* использует имена столбцов по умолчанию: `c1`, `c2`, и т. д. Обратите внимание на то, что если файл содержит строку заголовков, а вы не указываете эту опцию, то заголовки будут интерпретированы как значения данных. В результате сценарий может ошибочно определить все числовые столбцы как имеющие тип `CHAR` за счет присутствия нечислового значения – имени столбца.

--lower, *--upper*

Указывать имена столбцов в предложении `CREATE TABLE` в нижнем или верхнем регистре.

--quote-names

Заключать имена таблиц и столбцов в предложении `CREATE TABLE` в кавычки, используя символы ` (например, `` `mytbl` ``). Используется, если имя является зарезервированным словом. Получившееся предложение требует версии MySQL 3.23.6, так как более ранние версии не поддерживают заключение имен в кавычки.

--report

Формировать отчет, а не предложение `CREATE TABLE`. Сценарий выводит информацию, которую он собрал о каждом столбце.

--tbl_name=имя_таблицы

Указывает имя таблицы, которое должно использоваться в предложении `CREATE TABLE`. По умолчанию используется `t`.

Рассмотрим пример работы *guess_table.pl* для файла *managers.csv* из дистрибутива бейсбольной базы данных *baseball1.com*. Этот файл содержит записи для руководителей команд. Он начинается со строки заголовков столбцов, за которой следуют строки значений данных:

```
LahmanID,Year,Team,Lg,DIV,G,W,L,Pct,Std,Half,Order,PlyrMgr,PostWins,PostLosses
cravebi01,1871,TRO,NA,,25,12,12,0.5,6,0,2,,,
deaneha01,1871,KEK,NA,,5,2,3,0.4,8,0,2,,,
hastisc01,1871,ROK,NA,,25,4,21,0.16,9,0,0,,,
paborch01,1871,CLE,NA,,29,10,19,0.345,7,0,0,,,
```

```
wrighha01,1871,BOS,NA,,31,20,10,0.667,3,0,0,,,
youngni99,1871,OLY,NA,,32,15,15,0.5,5,0,0,,,
clappjo01,1872,MAN,NA,,24,5,19,0.208,8,0,0,,,
clintji01,1872,ECK,NA,,11,0,11,0,9,0,1,,,
fergubo01,1872,BRA,NA,,37,9,28,0.243,6,0,0,,,
...
```

Первая строка приводит заголовки столбцов, последующие строки содержат записи данных, по одной на строку. Сценарий *guess_table.pl* требует ввода, разделенного символами табуляции, с символом перевода строки в качестве признака конца строки, поэтому для обработки файла *managers.csv* необходимо сначала преобразовать его при помощи *cvt_file.pl*, записав результат во временный файл *managers.txt*:

```
% cvt_file.pl --iformat=csv --ieol="\r\n" managers.csv > managers.txt
```

Затем для временного файла выполним *guess_table.pl* (используем опцию *--lower*, так как мне больше нравятся имена столбцов в нижнем регистре):

```
% guess_table.pl --table=managers --labels --lower managers.txt > managers.sql
```

Предложение `CREATE TABLE`, которое сценарий *guess_table.pl* записывает в *managers.sql*, выглядит так:

```
CREATE TABLE managers
(
    lahmanid CHAR(9) NOT NULL,
    year INT UNSIGNED NOT NULL,
    team CHAR(3) NOT NULL,
    lg CHAR(2) NOT NULL,
    div CHAR(1) NULL,
    g INT UNSIGNED NOT NULL,
    w INT UNSIGNED NOT NULL,
    l INT UNSIGNED NOT NULL,
    pct FLOAT NOT NULL,
    std INT UNSIGNED NOT NULL,
    half INT UNSIGNED NOT NULL,
    order INT UNSIGNED NOT NULL,
    plyrmgr CHAR(1) NULL,
    postwins INT UNSIGNED NULL,
    postlosses INT UNSIGNED NULL
);
```

А получается такое предложение на основе следующих предположений:

- Если столбец содержит только целые значения, считается, что он имеет тип `INT`. Если ни одно из его значений не отрицательно, вероятно, он также является столбцом без знака.
- Если столбец не содержит пустых значений, предполагается, что он `NOT NULL`.

- Столбцы, которые не удается отнести к числовым или датам, принимаются за столбцы CHAR, длина которых совпадает с самым длинным значением столбца.

Вы можете захотеть отредактировать предложение CREATE TABLE, построенное утилитой *guess_table.pl*, чтобы, например, увеличить размер символьных полей, изменить CHAR на VARCHAR или добавить индексы. Есть и другая причина редактирования предложения: если имя столбца является зарезервированным словом MySQL, можно заменить его. Например, определение таблицы managers, созданное *guess_table.pl*, включает столбец order, имя которого – зарезервированное ключевое слово. Столбец представляет порядок руководителей в течение сезона (если они менялись), так что в качестве разумной альтернативы названия можно предложить имя mgrorder. После редактирования предложения в файле *managers.sql* с изменением имени, выполните его для создания таблицы:

```
% mysql cookbook < managers.sql
```

Теперь можно загружать в таблицу данные (пропуская первую строку заголовков):

```
mysql> LOAD DATA LOCAL INFILE 'managers.txt' INTO TABLE managers
    -> IGNORE 1 LINES;
```

(Когда вы будете это делать, LOAD DATA будет выводить предупреждения. О них мы поговорим в рецепте 10.37.)

База данных *baseball1.com* существует и в формате Access. База данных Access содержит явную информацию о структуре таблицы managers, и эта информация доступна таким утилитам, как DBTools и MySQLFront, которые могут использовать ее для создания таблицы MySQL (программы описаны в рецепте 10.38). Так вы получаете возможность проверить, насколько хорошо *guess_table.pl* угадывает структуру таблицы по сведениям только из файла данных по сравнению с программами, обладающими большим объемом информации.

Проблема утилит типа DBTools и MySQLFront в том, что если имя столбца таблицы Access является зарезервированным словом, невозможно импортировать его в MySQL, не изменив таблицу Access так, чтобы она использовала другие имя. Зарезервированным словом является имя столбца Order таблицы managers. При использовании *guess_table.pl* никаких проблем не возникает, так как вы просто редактируете формируемое предложение CREATE TABLE так, чтобы имя было разрешенным.[1] Но чтобы обрабатывать столбец Order таблицы managers при помощи DBTools или MySQLFront, необходимо изменить саму базу данных Access, переименовав ее столбец (например, в MgrOrder.)

Структура таблицы managers, предлагаемая DBTools, такова:

---

[1] Можно также использовать опцию *-quote-name* при запуске *guess_table.pl*. Тогда вы сможете создавать таблицу, не меняя имен столбцов, но не забывайте заключать имя в кавычки при каждой ссылке на него.

```
CREATE TABLE managers (
  LahmanID char(9) NOT NULL default '',
  Year int(11) default NULL,
  Team char(3) default NULL,
  Lg char(2) default NULL,
  Div char(2) default NULL,
  G int(11) default NULL,
  W int(11) default NULL,
  L int(11) default NULL,
  Pct double default NULL,
  Std int(11) default NULL,
  Half int(11) default NULL,
  MgrOrder int(11) default NULL,
  PlyrMgr char(1) default NULL,
  PostWins int(11) default NULL,
  PostLosses int(11) default NULL,
  KEY LahmanID (LahmanID)
);
```

MySQLFront создает такую таблицу:

```
CREATE TABLE managers (
  LahmanID longtext,
  Year int(11) default NULL,
  Team longtext,
  Lg longtext,
  Div longtext,
  G int(11) default NULL,
  W int(11) default NULL,
  L int(11) default NULL,
  Pct float default NULL,
  Std int(11) default NULL,
  Half int(11) default NULL,
  MgrOrder int(11) default NULL,
  PlyrMgr longtext,
  PostWins int(11) default NULL,
  PostLosses int(11) default NULL
);
```

Утилита DBTools определяет структуру таблицы MySQL более точно, чем две другие программы. Она использует индексную информацию, предоставляемую файлом Access, для вывода определения `KEY` и для создания строковых столбцов с соответствующими длинами. MySQLFront не определяет ключ и определяет строки как столбцы `LONGTEXT`, даже столбец `PlyrMgr`, длины значений которого никогда не превышают один символ. Качество вывода *guess_table.pl* находится где-то посередине. Утилита не определяет ключ, но и не сопоставляет каждому строковому столбцу наибольшую длину. (С другой стороны, с длинами столбцов надо соблюдать осторожность.) В общем, неплохо, учитывая то, что *guess_table.pl* не имеет доступа ко всей информации, содержащейся в исходном файле Access. К тому же этот сценарий можно использовать на разных платформах.

Результаты показывают, что если вы используете Windows и записи хранятся в файле Access, то лучше всего доверить создание таблицы MySQL утилите DBTools. В других случаях (при работе в UNIX или если ваши файлы данных поступают не из Access) утилита *guess_table.pl* может оказаться более эффективной.

10.37. Диагностическая утилита для LOAD DATA

Задача

`LOAD DATA` или *mysqlimport* при загрузке файла данных в MySQL выводит ненулевой счетчик предупреждений, но вы не представляете, какие строки или столбцы вызвали проблемы.

Решение

Запустите для файла утилиту, которая определяет, какие значения данных привели к выводу сообщений.

Обсуждение

Предложение `LOAD DATA` очень эффективно для массовой загрузки, оно работает во много раз быстрее, чем набор предложений `INSERT`, добавляющих те же самые строки. Однако это предложение не очень информативно. Оно возвращает только сообщение, указывающее количество обработанных записей и несколько других счетчиков. Например, в предыдущем разделе был создан файл данных *managers.txt*, используемый утилитой *guess_table.pl* для определения структуры таблицы `managers` базы данных *baseball1.com*. Если создать эту таблицу, используя результирующее предложение `CREATE TABLE`, и загрузить в нее файл данных, то результат будет таким:

```
mysql> LOAD DATA LOCAL INFILE 'managers.txt' INTO TABLE managers
    -> IGNORE 1 LINES;
Query OK, 2841 rows affected (0.06 sec)
Records: 2841  Deleted: 0  Skipped: 0  Warnings: 5082
```

Очевидно, что с файлом что-то не так. К сожалению, выводимое `LOAD DATA` сообщение ничего не говорит о том, в каких строках и столбцах возникли проблемы. Программа *mysqlimport* так же немногословна, ее сообщение совпадает с возвращаемым `LOAD DATA`.

Мы вернемся к этому примеру в конце раздела, пока же поговорим о манере вывода `LOAD DATA`. С одной стороны, минимальный вывод сообщений – это разумный подход. Если бы предупреждения возвращались клиенту, теоретически они могли бы содержать диагностическое сообщение для каждой строки ввода или даже для каждого столбца! Их было бы несметное количество, и они бы значительно снизили эффективность `LOAD DATA`. С другой стороны, больший объем информации об источнике ошибок мог бы оказаться полезен для корректировки файла.

В списке функциональностей MySQL, запланированных для будущих версий, стоит возможность регистрации ошибок LOAD DATA в другой таблице, с тем чтобы вы могли получать расширенную диагностическую информацию. Пока же вы можете использовать утилиту *load_diag.pl* из каталога *transfer* дистрибутива recipes. Утилита *load_diag.pl* удобна для предварительного просмотра файла данных с тем, чтобы понять, хорошо ли файл будет загружен в предназначенную для него таблицу и чтобы выявить проблемы, дабы разобраться с ними до выполнения «настоящей» загрузки в MySQL.

Утилита *load_diag.pl* также может помочь в определении типов проблем для ситуаций, в которых может оказаться полезным применение фильтра. Предположим, что вы периодически получаете файлы с данными для загрузки в определенную таблицу MySQL. Чем чаще это происходит, тем сильнее вы хотите автоматизировать (насколько это возможно) процесс переноса данных. Может потребоваться создать фильтр для преобразования значений данных из формата, в котором они поступают, в формат, более подходящий для MySQL. Обработка файлов данных утилитой *load_diag.pl* может помочь в оценке того, какие столбцы могут вызывать проблемы, и тем самым способствовать определению того, на чем стоит сосредоточить свои усилия по созданию программы преобразования файлов для их корректной загрузки в MySQL.

Чтобы запустить *load_diag.pl*, укажите имена базы данных и таблицы, в которую планируется загрузить файл данных, а также имя самого файла:

```
% load_diag.pl имя_БД имя_таблицы имя_файла
```

На самом деле *load_diag.pl* не будет ничего загружать в таблицу, имя которой указано в командной строке, но ей необходимо знать, что это за таблица, чтобы она могла создать временную таблицу, имеющую такую же структуру столбцов, в целях тестирования.

Сначала *load_diag.pl* загружает весь файл данных во временную таблицу, чтобы посмотреть, будут ли выведены какие-то предупреждения. Если нет, делать больше нечего, поэтому *load_diag.pl* удаляет временную таблицу и завершается. В противном случае она по отдельности загружает в таблицу каждую строку файла данных, чтобы определить, где возникла проблема:

- Строка записывается во временный файл, выдается предложение LOAD DATA для загрузки файла в таблицу. Если счетчик предупреждений равен нулю, то считается, что со строкой все хорошо.
- Если счетчик предупреждений не равен нулю, *load_diag.pl* по очереди исследует каждый из столбцов, используя серию одностолбцовых предложений LOAD DATA для выявления столбцов, генерирующих предупреждения.
- Если для столбца выводится сообщение, а значение данных пусто, то *load_diag.pl* определяет, исчезнет ли сообщение при загрузке значения NULL вместо пустого значения. Если файл данных содержит пустые значения, то часто можно добиться лучших результатов, загружая вместо пустых строк значения NULL. (Например, если вы загружаете пустую строку в столбец INT, MySQL преобразует ее в 0 и выдает сообщение.) Если окажется, что количество сообщений для файла значительно уменьшится при

загрузке NULL вместо пустых строк, то, возможно, следует перед загрузкой файла выполнить для него *to_null.pl*.

- Предупреждения также могут появляться, если строка содержит больше или меньше столбцов, чем количество столбцов таблицы, поэтому *load_diag.pl* проверяет и это.

Утилита *load_diag.pl* выводит диагностическую информацию о своих находках при тестировании каждой строки ввода, затем выводит итоговый отчет после завершения обработки файла. В отчете приводится общее количество строк файла, количество сообщений, порожденных загрузкой всего исходного файла, и количество строк, в которых было слишком много или слишком мало столбцов. Отчет также содержит список, указывающий для каждого столбца сколько значений отсутствует, сколько предупреждений выведено, сколько из этих сообщений вызвано пустыми значениями, и количество сообщений, вызванных пустыми значениями, которые исчезли при загрузке вместо них значения NULL.

Как вы понимаете, вся эта активность означает, что *load_diag.pl* далеко не так эффективна, как LOAD DATA. На самом деле она может очень сильно загрузить сервер! Но ее цель – это предоставление максимума информации, а не обеспечение минимального времени исполнения. (Отметьте, что если ваш сервер MySQL разрешает протоколирование, то использование *load_diag.pl* для больших файлов данных может привести к быстрому росту объема журнала.)

Чтобы посмотреть, как работает *load_diag.pl*, предположим, что у вас есть простая таблица diag_test, которая содержит строковый столбец, столбец дат и числовой столбец:

```
CREATE TABLE diag_test
(
    str     CHAR(10),
    date    DATE,
    num     INT
);
```

Предположим, что у вас также есть файл данных *diag_sample.dat*, который планируется загрузить в таблицу:

```
str1    01-20-2001      97
str2    02-28-2002
        03-01-2002      64      extra junk
```

Чтобы посмотреть, возникнут ли у файла проблемы при загрузке, проверим его так:

```
% load_diag.pl cookbook diag_test diag_sample.dat
line 1: 1 warning
  column 2 (date): bad value = (01-20-2001)
line 2: 2 warnings
  too few columns
  column 2 (date): bad value = (02-28-2002)
  column 3 (num): missing from input line
```

```
  column 3 (num): bad value = () (inserting NULL worked better)
line 3: 1 warning
  excess number of columns

Number of lines in file: 3
Warnings found when loading entire file: 4
Lines containing too few column values: 1
Lines containing excess column values: 1

Warnings per column:

Column    Times      Total     Warnings for     Improved
         missing    warnings   empty columns    with NULL
str            0          0              0            0
date           0          2              0            0
num            1          1              1            1
```

Похоже на то, что даты загружаются не очень хорошо. Это и неудивительно, ведь они имеют формат США и должны быть преобразованы в формат ISO. Преобразование пустых значений в `\N` также может сослужить хорошую службу. Кроме того, избавимся от лишнего значения в строке 3. Выполним все эти преобразования, используя некоторые утилиты, созданные ранее в этой главе, и запишем результат во временный файл:

```
% yank_col.pl --columns=1-3 diag_sample.dat \
    | cvt_date.pl --iformat=us --oformat=iso \
    | to_null.pl > tmp
```

Команда формирует такой файл *tmp*:

```
str1    2001-01-20      97
str2    2002-02-28      \N
\N      2002-03-01      64
```

Используем *load_diag.pl* для проверки нового файла:

```
% load_diag.pl cookbook diag_test tmp
File loaded with no warnings, no per-record tests performed
```

То есть если загрузить *tmp* в таблицу `diag_test`, результат должен быть хорошим, и так это и есть:

```
mysql> LOAD DATA LOCAL INFILE 'tmp' INTO TABLE diag_test;
Records: 3  Deleted: 0  Skipped: 0  Warnings: 0
```

Может показаться, что ради улучшения загрузки в MySQL какого-то файла из трех строк возникает слишком много ненужной суматохи. Но ситуация рассматривается лишь как иллюстрация использования *load_diag.pl* для выявления того, что неладно в файле данных, и как это можно исправить.

В дополнение к необходимым аргументам имен базы данных, таблицы и файла данных, *load_diag.pl* понимает ряд опций:

*--**columns**=имя1, имя2, имя3,...*

По умолчанию *load_diag.pl* считает, что файл данных содержит столбцы, количество и порядок которых совпадают со столбцами таблицы. Если

это не так, используйте опцию для указания имен столбцов, присутствующих в файле, и их порядка.

--labels

Опция указывает на то, что файл данных содержит начальную строку заголовков, которую следует пропустить. (Загрузка заголовков в таблицу обычно приводит к выводу ложных предупреждений.)

--skip-full-load

Пропустить начальный тест, загружающий весь файл данных.

--tmp-table=имя_таблицы

Указывает имя для временной таблицы. По умолчанию `_load_diag_n`, где *n* – это идентификатор процесса *load_diag.pl*.

При необходимости также можно указать стандартные опции параметров соединения, такие как *--user* или *--host*. Все опции должны стоять перед аргументом имени базы данных.

На использование *load_diag.pl* налагаются такие ограничения:

- Ввод должен быть разделен символами табуляции, признак конца строки – символ перевода строки.
- Загрузка записи выполняется при помощи предложения `LOAD DATA` с опцией `LOCAL`. Применение `LOCAL` возможно начиная с версии MySQL 3.22.15 и выше (и начиная с версии 3.23.49 требуется, чтобы эта возможность не отключалась при сборке).
- Когда *load_diag.pl* создает временную таблицу, все имеющиеся в исходной таблице индексы не учитываются, что уменьшает время загрузки записи (особенно для первичной проверки, загружающей весь файл данных). С другой стороны, неиспользование индексов означает, что *load_diag.pl* не обнаружит предупреждений о повторяющихся значениях уникальных ключей.

Вернемся к примеру из начала раздела: так чем же объясняются все эти предупреждения, появившиеся при загрузке файла *managers.txt* в таблицу `managers`? Утилита *load_diag.pl* идентифицировала все их как результат отсутствующих или пустых столбцов в конце некоторых строк:

```
% load_diag.pl --labels cookbook managers managers.txt
line 2: 2 warnings
  column 14 (postwins): bad value = () (inserting NULL worked better)
  column 15 (postlosses): bad value = () (inserting NULL worked better)
line 3: 2 warnings
  column 14 (postwins): bad value = () (inserting NULL worked better)
  column 15 (postlosses): bad value = () (inserting NULL worked better)
...
line 2839: 2 warnings
  column 14 (postwins): bad value = () (inserting NULL worked better)
  column 15 (postlosses): bad value = () (inserting NULL worked better)
line 2842: 2 warnings
```

```
  column 14 (postwins): bad value = () (inserting NULL worked better)
  column 15 (postlosses): bad value = () (inserting NULL worked better)

Number of lines in file: 2842
Warnings found when loading entire file: 5082
Lines containing too few column values: 416
Lines containing excess column values: 0

Warnings per column:

Column        Times      Total     Warnings for     Improved
             missing    warnings   empty columns    with NULL
lahmanid           0           0               0           0
year               0           0               0           0
team               0           0               0           0
lg                 0           0               0           0
div                0           0               0           0
g                  0           0               0           0
w                  0           0               0           0
l                  0           0               0           0
pct                0           0               0           0
std                0           0               0           0
half               0           0               0           0
mgrorder           0           0               0           0
plyrmgr           16           0               0           0
postwins         416        2533            2533        2533
postlosses       416        2533            2533        2533
```

Из результата видно, что в 416 строках отсутствуют столбцы `postwins` и `postlosses` (а в 16 строках – и столбец `plyrmgr`). Остальные ошибки возникли в строках, в которых столбцы `postwins` и `postlosses` присутствуют, но содержат пустые значения. Общее количество предупреждений для файла равно 5082 и может рассматриваться как количество отсутствующих значений `plyrmgr` плюс общее количество предупреждений о столбцах `postwins` и `postlosses` (16 + 2533 + 2533 = 5082).

Значение `Total warnings` для столбца `plyrmgr` равно нулю, так как это столбец типа `CHAR`, и в него разрешено загружать пустые значения. Значение `Total warnings` для `postwins` и `postlosses` не равно нулю, так как они относятся к типу `INT`, и загрузка в них пустых значений инициирует операции преобразования в ноль. Все проблемы могут быть решены за счет преобразования пустых и отсутствующих значений в `\N`. Обработайте файл утилитой *yank_col.pl*, тогда в каждой строке будет по 15 столбцов, затем примените к результату сценарий *to_null.pl* для преобразования пустых значений в `\N`:

```
% yank_col.pl --columns=1-15 managers.txt | to_null.pl > tmp
```

Посмотрим, что *load_diag.pl* скажет о получившемся файле:

```
% load_diag.pl --labels cookbook managers tmp
File loaded with no warnings, no per-record tests performed
```

Если загрузить *tmp* в таблицу `managers`, никаких проблем не возникнет:

```
mysql> LOAD DATA LOCAL INFILE 'tmp' INTO TABLE managers IGNORE 1 LINES;
Query OK, 2841 rows affected (0.13 sec)
Records: 2841  Deleted: 0  Skipped: 0  Warnings: 0
```

10.38. Обмен данными между MySQL и Microsoft Access

Задача

Вы хотите осуществлять обмен информацией между MySQL и Access.

Решение

Чтобы использовать информацию, хранящуюся в MySQL, установите соединение с сервером MySQL непосредственно из Access. Чтобы передать данные из Access в MySQL, используйте утилиту, которая может выполнять прямую передачу данных, или экспортируйте таблицы из Access в файлы и импортируйте файлы в MySQL.

Обсуждение

И MySQL, и Access понимают ODBC, так что вы можете соединиться с MySQL прямо из Access. Установив ODBC-соединение, Access становится внешним интерфейсом, через который вы можете использовать базу данных MySQL. Много полезной информации можно найти на сайте *mysql.com* в разделе, посвященном MyODBC:

http://www.mysql.com/products/myodbc/

Замечательное описание процедур настройки ODBC и соединения из Access с MySQL по ODBC приведено в статье В. Дж. Гилмора (W. J. Gilmore) на сайте DevShed:

http://www.devshed.com/Server_Side/MySQL/ODBC/

Если ваши таблицы хранятся в Access, а вы хотите перенести их в MySQL, необходимо создать в MySQL таблицы и импортировать данные Access в эти таблицы. Есть пара хороших бесплатных инструментов, которые могут вам помочь: DBTools и MySQLFront. Они могут исследовать структуру таблиц в базе данных Access, создать соответствующие таблицы в MySQL и скопировать туда данные.

Можно также экспортировать таблицы Access в файлы, затем импортировать эти файлы в MySQL. (Это необходимо, если, например, сервер MySQL работает на другом хосте, который не разрешает устанавливать соединения с вашей машиной под Windows.) Если вы пойдете именно этим путем, вам следует подумать о выборе формата файла, преобразовании формата дат и о том, как создать таблицы MySQL, если они еще не существуют. В этом вам могут помочь сценарии, описанные ранее в этой главе (такие как *cvt_file.pl*, *cvt_date.pl* и *guess_table.pl*). Процедура импорта таблицы Access в MySQL может быть, например, такой:

1. Экспортировать таблицу из Access в какой-то текстовый формат, возможно, вместе с заголовками столбцов. Если вам нужно будет преобразовывать файл при помощи утилит, которым требуется на входе файл в формате с символами табуляции в качестве разделителя и символом перевода строки в качестве признака конца строки, наиболее разумным будет экспорт именно в такой формат.
2. Если таблица содержит даты, и вы не экспортируете их в формат ISO, необходимо будет преобразовать их для MySQL. Можно воспользоваться утилитой *cvt_date.pl*.
3. Если таблица MySQL, в которую вы планируете импортировать данные Access, не существует, нужно создать ее. Используйте утилиту *guess_table.pl* для формирования предложения `CREATE TABLE`.
4. Импортировать файл данных в MySQL при помощи `LOAD DATA` или *mysqlimport*.

10.39. Обмен данными между MySQL и Microsoft Excel

Задача

Вы хотите осуществлять обмен информацией между MySQL и Excel.

Решение

Используйте такие утилиты, как DBTools и MySQLFront. Или примените модули Perl, которые читают и записывают файлы электронных таблиц Excel, для создания собственных утилит передачи данных.

Обсуждение

Один из способов передачи файлов Excel в MySQL – применение утилит DBTools и MySQLFront, упомянутых в рецепте 10.38 при обсуждении обработки файлов Access. Обе программы умеют читать и файлы Excel. Но обе работают только в Windows, так что если вам нужно решение, работающее и в UNIX, и в Windows, вы можете читать и записывать электронные таблицы Excel из сценариев Perl, установив предварительно несколько модулей:

- Модуль *Spreadsheet::ParseExcel::Simple* предоставляет удобный интерфейс для чтения таблиц Excel.
- Модуль *Spreadsheet::WriteExcel::Simple* позволяет создавать файлы в формате электронных таблиц Excel.

Эти модули доступны в Perl CPAN. (Они фактически являются внешними интерфейсами к другим модулям, которые также нужно будет предварительно установить.) После установки модулей для обращения к их документации используйте такие команды:

```
% perldoc Spreadsheet::ParseExcel::Simple
% perldoc Spreadsheet::WriteExcel::Simple
```

Эти модули значительно упрощают написание двух небольших сценариев (приведенных ниже) для преобразования электронных таблиц в формат с разделителями-символами табуляции и обратно. Используя эти сценарии в сочетании с приемами импорта и экспорта данных MySQL, вы сможете перемещать содержимое электронных таблиц в таблицы MySQL и наоборот. Можно применять их «как есть» или адаптировать к вашим конкретным требованиям.

Сценарий *from_excel.pl* читает электронные таблицы Excel и преобразует их в формат значений, разделенных символами табуляции:

```
#! /usr/bin/perl -w
# from_excel.pl - читает электронную таблицу Excel, записывает
# в стандартный вывод в формате: символы табуляции в качестве разделителей,
# символ перевода строки - признак конца строки.

use strict;
use Spreadsheet::ParseExcel::Simple;

@ARGV or die "Usage: $0 excel-file\n";

my $xls = Spreadsheet::ParseExcel::Simple->read ($ARGV[0]);
foreach my $sheet ($xls->sheets ())
{
    while ($sheet->has_data ())
    {
        my @data = $sheet->next_row ();
        print join ("\t", @data) . "\n";
    }
}

exit (0);
```

Сценарий *to_excel.pl* выполняет обратную операцию чтения файла с разделителями-табуляциями и записывает его в формате Excel:

```
#! /usr/bin/perl -w
# to_excel.pl - читает ввод, разделенный символами табуляции, признак конца
# строки - символ перевода строки, записывает в стандартный вывод в формате Excel.

use strict;
use Spreadsheet::WriteExcel::Simple;

my $ss = Spreadsheet::WriteExcel::Simple->new ();

while (<>)                              # читать каждую строку ввода
{
    chomp;
    my @data = split (/\t/, $_, 10000); # разбить на части, сохраняя все поля
    $ss->write_row (\@data);            # записать строку в таблицу
}

print $ss->data (); # вывести таблицу

exit (0);
```

Сценарий *to_excel.pl* принимает ввод в формате значений, разделенных символами табуляции, с символом перевода строки в качестве признака конца строки. Для того чтобы он мог работать с файлами в другом формате, используйте его в сочетании с *cvt_file.pl*.

Еще один связанный с Excel модуль Perl, *Spreadsheet::WriteExcel::FromDB*, читает данные из таблицы, используя соединение DBI, и записывает их в формате Excel. Рассмотрим небольшой сценарий, который экспортирует таблицу MySQL в виде электронной таблицы Excel:

```
#! /usr/bin/perl -w
# mysql_to_excel.pl - получив имена базы данных и таблицы,
# выгружает таблицу в стандартный вывод в формате Excel.

use strict;
use DBI;
use Spreadsheet::ParseExcel::Simple;
use Spreadsheet::WriteExcel::FromDB;

# ... обработка опций командной строки (не приводится) ...

@ARGV == 2 or die "Usage: $0 [options] db_name tbl_name\n";
my $db_name = shift (@ARGV);
my $tbl_name = shift (@ARGV);

# ... соединение с базой данных (не приводится) ...

my $ss = Spreadsheet::WriteExcel::FromDB->read ($dbh, $tbl_name);
print $ss->as_xls ();

exit (0);
```

Все три утилиты выводят результат в стандартный вывод, откуда его можно перенаправить, например, в файл:

```
% from_excel.pl data.xls > data.txt
% to_excel.pl data.txt > data.xls
% mysql_to_excel.pl cookbook profile > profile.xls
```

10.40. Обмен данными между MySQL и FileMaker Pro

Задача

Вы хотите осуществлять обмен информацией между MySQL и FileMaker Pro.

Решение

В Windows вы можете установить из FileMaker Pro ODBC-соединение с сервером MySQL. Можно и экспортировать таблицы из MySQL в файлы, а затем импортировать их в FileMaker Pro, или наоборот. Но не забывайте о возможной несовместимости типов данных.

Обсуждение

Если вы можете установить соединение из FileMaker Pro с сервером MySQL по ODBC, то можете так получить доступ к таблицам MySQL. Процедура аналогична описанной для соединения с MySQL из Access (см. рецепт 10.38).

Также вы можете экспортировать данные из одной программы в файл, а затем импортировать их в другую программу. Каталог *transfer* дистрибутива `recipes` содержит утилиту *mysql_to_filemaker.pl*, экспортирующую таблицу MySQL в файл, который можно импортировать в FileMaker Pro. Это сценарий предназначен для обработки следующих ситуаций, возможных при работе с FileMaker Pro:

- Формат дат по умолчанию для FileMaker Pro – это *MM-DD-CCYY*. Сценарий перезаписывает содержимое любых столбцов таблицы MySQL, содержащих даты так, чтобы они соответствовали формату даты FileMaker Pro.
- FileMaker Pro поддерживает такие типы столбцов, как дата и время, но не комбинированный тип дата-и-время. Сценарий *mysql_to_filemaker.pl* экспортирует столбцы `DATETIME` и `TIMESTAMP` как отдельные значения `DATE` и `TIME`. (Например, столбец таблицы с именем `c` экспортируется как два столбца – `c_date` и `c_time`.)
- Все внутренние символы перевода строки и возврата каретки внутри значений столбцов отображаются в Ctrl-K, именно так FileMaker Pro представляет разделители строк внутри значений данных.

Для обработки значений данных *mysql_to_filemaker.pl* использует подход, подобный изложенному в рецепте 10.33 для формирования предложения `SELECT`, когда таблица экспортируется с перезаписью дат. То есть сценарий читает метаданные таблицы, чтобы выявить столбцы дат, и экспортирует их, используя вызовы `DATE_FORMAT()` для перезаписи значений столбцов в формат FileMaker Pro.

mysql_to_filemaker.pl записывает вывод в формате, который FileMaker Pro называет форматом слияния (merge), а по сути он является форматом CSV с начальной строкой заголовков столбцов. Файлы слияния удобны для FileMaker Pro по нескольким причинам:

- При создании новой таблицы из файлов в формате слияния FileMaker Pro автоматически использует заголовки для имен столбцов. Обеспечивается простой способ переноса имен столбцов таблицы MySQL в базу данных FileMaker Pro.
- При импорте файла слияния в существующую таблицу наличие строки заголовков столбцов упрощает сопоставление столбцов файла данных столбцам таблицы.

Сценарий *mysql_to_filemaker.pl* требует указания в командной строке аргументов имен базы данных и таблицы. Например, чтобы экспортировать содержимое таблицы `mail` в файл слияния *mail.mer*, вызываем сценарий так:

```
% mysql_to_filemaker.pl cookbook mail > mail.mer
```

Таблица `mail` содержит столбец `t` со значениями `DATETIME`. Если вы внимательно посмотрите на *mail.mer*, то заметите, что *mysql_to_filemaker.pl* экспортирует столбец `t` как два отдельных столбца, `t_date` и `t_time`, преобразуя порядок составляющих даты из формата ISO в *MM-DD-CCYY*:

```
t_date,t_time,srcuser,srchost,dstuser,dsthost,size
05-11-2001,10:15:08,barb,saturn,tricia,mars,58274
05-12-2001,12:48:13,tricia,mars,gene,venus,194925
05-12-2001,15:02:49,phil,mars,phil,saturn,1048
05-13-2001,13:59:18,barb,saturn,tricia,venus,271
05-14-2001,09:31:37,gene,venus,barb,mars,2291
...
```

Кроме того, сценарий *mysql_to_filemaker.pl* понимает обычные опции параметров соединения (такие, как *--user* и *--host*). Все опции должны быть указаны перед аргументом имени базы данных.

После создания файла слияния можно указать FileMaker Pro на необходимость открыть его (тогда будет создана новая таблица) или импортировать в существующую базу данных.

Для переноса данных в обратном направлении (импорта базы данных FileMaker Pro в MySQL) выполните такие операции:

1. Экспортируйте базу данных в какой-нибудь тестовый формат. Если вы хотите, чтобы файл содержал строку заголовков столбцов, экспортируйте базу данных в формат слияния. Затем можно будет обработать файл утилитой *cvt_file.pl*, чтобы вывести строки, элементы которых разделены символами табуляции, признак конца строки – символ перевода строки. Это пригодится, если понадобится преобразовать файл с помощью других утилит, которые предполагают такой формат ввода.
2. Если база данных содержит даты, то в зависимости от формата, который FileMaker Pro использует для их экспорта, может потребоваться преобразование дат для MySQL. FileMaker Pro использует формат по умолчанию *MM-DD-CCYY*, если указать для экспорта опцию «don't format output» (не форматировать вывод). Если же выбрать «display using current layout» (отображать в текущем формате), даты будут экспортированы в том формате, в котором их отображает FileMaker Pro. Вы можете использовать сценарий *cvt_date.pl* для перезаписи дат в формат ISO.
3. Если таблица MySQL, в которую вы планируете импортировать данные FileMaker Pro, не существует, создайте ее. Утилита *guess_table.pl* может пригодиться для формирования предложения `CREATE TABLE`.
4. Импортируйте файл данных в MySQL при помощи `LOAD DATA` и *mysqlimport*.

10.41. Экспорт результатов запроса в XML

Задача

Вы хотите экспортировать результат запроса в XML-документ.

Решение

Используйте *mysql* или напишите собственную программу экспорта.

Обсуждение

Чтобы сформировать вывод результата запроса в формате XML, используйте *mysql*, если вы работаете с версией MySQL 4.0 или выше (см. рецепт 1.24).

Можно написать собственную программу экспорта в XML. Для этого создайте запрос и выведите его, самостоятельно добавляя разметку XML. Или упростите себе жизнь, установив несколько модулей Perl и позволив им поработать:

- *XML::Generator::DBI* создает запрос по соединению DBI и передает результат соответствующей программе записи выходных данных.
- *XML::Handler::YAWriter* предоставляет такую программу записи выходных данных.

Следующий сценарий, *mysql_to_xml.pl*, напоминает *mysql_to_text.pl* (см. рецепт 10.17), но не принимает опций для символов разделения и заключения в кавычки. Вы можете указать следующие опции:

--execute=запрос, *-e запрос*

Выполнить запрос *запрос* и экспортировать вывод.

--table=имя_таблицы, *-t имя_таблицы*

Экспортировать содержимое указанной таблицы. Эквивалентно использованию *--execute* для указания значения *запрос* для `SELECT * FROM` *имя_таблицы*.

При необходимости можно указать стандартные опции параметров соединения, такие как *--user* или *--host*. Последним аргументом командной строки должно быть имя базы данных, если только оно не указано неявно в запросе.

Предположим, что вы хотите экспортировать содержимое такой таблицы с данными экспериментов:

```
mysql> SELECT * FROM expt;
+---------+------+-------+
| subject | test | score |
+---------+------+-------+
| Jane    | A    |    47 |
| Jane    | B    |    50 |
| Jane    | C    |  NULL |
| Jane    | D    |  NULL |
| Marvin  | A    |    52 |
| Marvin  | B    |    45 |
| Marvin  | C    |    53 |
| Marvin  | D    |  NULL |
+---------+------+-------+
```

Вызовем *mysql_to_xml.pl* при помощи одной из команд:

```
% mysql_to_xml.pl --execute="SELECT * FROM expt" cookbook > expt.xml
% mysql_to_xml.pl --table=cookbook.expt > expt.xml
```

Получившийся документ XML *expt.xml* будет выглядеть так:

```
<?xml version="1.0" encoding="UTF-8"?>
<rowset>
 <select query="SELECT * FROM expt">
  <row>
   <subject>Jane</subject>
   <test>A</test>
   <score>47</score>
  </row>
  <row>
   <subject>Jane</subject>
   <test>B</test>
   <score>50</score>
  </row>
  <row>
   <subject>Jane</subject>
   <test>C</test>
  </row>
  <row>
   <subject>Jane</subject>
   <test>D</test>
  </row>
  <row>
   <subject>Marvin</subject>
   <test>A</test>
   <score>52</score>
  </row>
  <row>
   <subject>Marvin</subject>
   <test>B</test>
   <score>45</score>
  </row>
  <row>
   <subject>Marvin</subject>
   <test>C</test>
   <score>53</score>
  </row>
  <row>
   <subject>Marvin</subject>
   <test>D</test>
  </row>
 </select>
</rowset>
```

Каждая строка записывается как элемент `<row>`. Внутри строки имена и значения столбцов используются в качестве имен и значений элементов, по одному элементу на столбец. Обратите внимание, что значения `NULL` в выводе не присутствуют.

Сценарий создает этот документ при помощи совсем небольшого фрагмента кода после обработки аргументов командной строки и установления соединения (код которых не приводится). Собственно с XML связаны части сценария *mysql_to_xml.pl*, содержащие предложения `use`, которые привлекают необходимые модули, а также создают и используют объекты XML. Когда заданы дескриптор базы данных `$dbh` и строка запроса `$query`, задача не так уж сложна. Код указывает объекту записи выходных данных на необходимость отправки результатов в стандартный вывод, устанавливает соединение объекта с DBI и запускает запрос:

```
#! /usr/bin/perl -w
# mysql_to_xml.pl - выгрузить указанную таблицу указанной базы данных
# в стандартный вывод в формате XML.

use strict;
use DBI;
use XML::Generator::DBI;
use XML::Handler::YAWriter;

# ... обработка опций командной строки (не приводится) ...

# ... соединение с базой данных (не приводится) ...

# создать программу записи вывода; "-" означает "стандартный вывод"
my $out = XML::Handler::YAWriter->new (AsFile => "-");
# установить соединение между DBI и программой записи вывода
my $gen = XML::Generator::DBI->new (
                            dbh => $dbh,             # дескриптор базы данных
                            Handler => $out,         # программа записи вывода
                            RootElement => "rowset" # корневые элементы документа
                          );
# запустить запрос и вывести XML
$gen->execute ($query);

$dbh->disconnect ();

exit (0);
```

10.42. Импорт XML в MySQL

Задача

Вы хотите импортировать XML-документ в таблицу MySQL.

Решение

Настройте программу синтаксического анализа XML для чтения документа. Затем используйте записи документа для формирования и выполнения предложений `INSERT`.

Обсуждение

Процедура импорта документа XML зависит от возможности синтаксического разбора документа и извлечения из него содержимого записей. Все определяется тем, как записан документ. Например, в одном формате имена и значения столбцов могут быть представлены как атрибуты элементов <column>:

```
<?xml version="1.0" encoding="UTF-8"?>
<rowset>
  <row>
   <column name="subject" value="Jane" />
   <column name="test" value="A" />
   <column name="score" value="47" />
  </row>
  <row>
   <column name="subject" value="Jane" />
   <column name="test" value="B />
   <column name="score" value="50" />
  </row>
...
</rowset>
```

Другой формат может использовать имена столбцов как имена элементов, а значения столбцов – как содержимое этих элементов:

```
<?xml version="1.0" encoding="UTF-8"?>
<rowset>
  <row>
   <subject>Jane</subject>
   <test>A</test>
   <score>47</score>
  </row>
  <row>
   <subject>Jane</subject>
   <test>B</test>
   <score>50</score>
  </row>
...
</rowset>
```

Поскольку существует несколько вариантов структуры, необходимо сделать какие-то предположения об ожидаемом формате XML-документа. В данном разделе я предполагаю наличие второго из только что описанных форматов. Для обработки такого документа можно использовать модуль *XML::XPath*, позволяющий ссылаться на элементы документа посредством путевых выражений. Например, путь //row выбирает все элементы <row> ниже корня документа, а путь * выбирает всех потомков указанного элемента. Пути можно использовать в *XML::XPath* для получения всех элементов <row>, а затем для получения списка всех столбцов каждой строки.

Сценарий *xml_to_mysql.pl* принимает три аргумента:

```
% xml_to_mysql.pl имя_БД имя_таблицы имя_файла_xml
```

Аргумент имени файла указывает, какой документ следует импортировать, а аргументы имен базы данных и таблицы – в какую таблицу импортировать документ.

Сценарий *xml_to_mysql.pl* обрабатывает аргументы командной строки и устанавливает соединение с MySQL (код не приводится), затем обрабатывает документ:

```
#! /usr/bin/perl -w
# xml_to_mysql.pl – чтение файла XML в MySQL

use strict;
use DBI;
use XML::XPath;

# ... обработка опций командной строки (не приводится) ...

# ... соединение с базой данных (не приводится) ...

# Открыть файл для чтения
my $xp = XML::XPath->new (filename => $file_name);
my $row_list = $xp->find ("//row");         # найти множество элементов <row>
print "Number of records: " . $row_list->size () . "\n";
foreach my $row ($row_list->get_nodelist ())        # цикл по строкам
{
    my @name;   # массив для имен столбцов
    my @val;    # массив для значений столбцов
    my $col_list = $row->find ("*");                # потомки (столбцы) строки
    foreach my $col ($col_list->get_nodelist ())    # цикл по столбцам
    {
        # сохранение имени и значения столбца
        push (@name, $col->getName ());
        push (@val, $col->string_value ());
    }
    # создание и выполнение предложения INSERT
    my $stmt = "INSERT INTO $tbl_name ("
                . join (",", @name)
                . ") VALUES ("
                . join (",", ("?") x scalar (@val))
                . ")";
    $dbh->do ($stmt, undef, @val);
}

$dbh->disconnect ();

exit (0);
```

Сценарий создает объект `XML::XPath`, который открывает и анализирует документ. Затем запрашивается набор элементов `<row>` объекта при помощи пути `//row`. Размер этого набора указывает количество записей, содержащихся в документе.

Для обработки каждой строки сценарий использует путь `*` для поиска всех потомков объекта строки. Каждый потомок соответствует столбцу строки.

Такое использование * в качестве пути для get_nodelist() удобно, так как нам не нужно заранее знать, какие столбцы следует ожидать. Сценарий *xml_to_mysql.pl* получает имя и значение каждого столбца и сохраняет их в массивах @name и @value. После того, как все столбцы обработаны, массивы используются для построения предложения INSERT, которое указывает имена столбцов, обнаруженных в строке, и включает заполнитель для каждого значения данных (формирование списка заполнителей рассматривается в рецепте 2.6). Затем сценарий запускает предложение, передавая значения столбцов в do() для связывания с заполнителями.

В предыдущем разделе сценарий *mysql_to_xml.pl* использовался для экспорта содержимого таблицы expt в документ XML. Сценарий *xml_to_mysql.pl* может применяться для выполнения обратной операции импорта документа в MySQL:

```
% xml_to_mysql.pl cookbook expt expt.xml
```

По мере обработки документа сценарий формирует и выполняет такой набор предложений:

```
INSERT INTO expt (subject,test,score) VALUES ('Jane','A','47')
INSERT INTO expt (subject,test,score) VALUES ('Jane','B','50')
INSERT INTO expt (subject,test) VALUES ('Jane','C')
INSERT INTO expt (subject,test) VALUES ('Jane','D')
INSERT INTO expt (subject,test,score) VALUES ('Marvin','A','52')
INSERT INTO expt (subject,test,score) VALUES ('Marvin','B','45')
INSERT INTO expt (subject,test,score) VALUES ('Marvin','C','53')
INSERT INTO expt (subject,test) VALUES ('Marvin','D')
```

Заметьте, что не все из этих выражений вставляют одно и то же количество столбцов. Предложения с «отсутствующими» значениями соответствуют строкам со значениями NULL.

10.43. Эпилог

Вспомним, с чего начиналась эта глава:

Предположим, у вас есть файл *somedata.csv* в формате CSV, содержащий 12 столбцов значений, разделенных запятыми (формат CSV). Вы хотите извлечь данные из столбцов 2, 11, 5, 9 и сохранить их в таблице MySQL, включающей столбцы name, birth, height и weight (имя, дата рождения, рост, вес). При этом необходимо убедиться в том, что рост и вес представлены целыми положительными числами, и преобразовать дату из формата *MM/DD/YY* в формат *CCYY-MM-DD*. Как можно это сделать?

Итак... как вы это *сделаете* после знакомства со всеми приемами, описанными в главе?

Большую часть работы могут выполнить созданные утилиты. Вы преобразуете файл к формату значений, разделенных символами табуляции, с помощью *cvt_file.pl*, извлекаете столбцы в нужном порядке с помощью *yank_col.pl* и перезаписываете столбец дат в формат ISO с помощью *cvt_date.pl*:

```
% cvt_file.pl --iformat=csv somedata.csv \
    | yank_col.pl --columns=2,11,5,9 \
    | cvt_date.pl --columns=2 --iformat=us --add-century > tmp
```

Полученный файл *tmp* будет содержать четыре столбца, представляющих значения `name`, `birth`, `height` и `weight` именно в таком порядке. Нужно лишь проверить столбцы роста и веса на предмет целых положительных значений. При помощи библиотечной функции `is_positive_integer()` файла модуля *Cookbook_Utils.pm* эта задача может быть решена путем использования небольшого специального сценария, который по сути является просто циклом обработки ввода:

```
#! /usr/bin/perl -w
# validate_htwt.pl - пример проверки корректности роста/веса

# Предполагается, что строки ввода разделены символами табуляции,
# признак конца строки - символ перевода строки.

# Столбцы ввода и выполняемые над ними действия таковы:
# 1: имя; отображать, как указано
# 2: дата рождения; отображать, как указано
# 3: рост; проверять на положительное целое
# 4: вес; проверять на положительное целое

use strict;
use lib qw(/usr/local/apache/lib/perl);
use Cookbook_Utils;

while (<>)
{
    chomp;
    my ($name, $birth, $height, $weight) = split (/\t/, $_, 4);
    warn "line $.:height $height is not a positive integer\n"
                                if !is_positive_integer ($height);
    warn "line $.:weight $weight is not a positive integer\n"
                                if !is_positive_integer ($weight);
}

exit (0);
```

Сценарий *validate_htwt.pl* не формирует никакого вывода (кроме предупреждений об ошибках), так как ему не нужно переформатировать ни одну из строк ввода. Если считать, что проверка корректности *tmp* прошла успешно, файл можно загрузить в MySQL при помощи простого предложения `LOAD DATA`:

```
mysql> LOAD DATA LOCAL INFILE 'tmp' INTO TABLE имя_таблицы;
```

11

Формирование и использование последовательностей

11.0. Введение

Последовательность (sequence) – это множество целых чисел (1, 2, 3, ...), генерируемых в требуемом порядке по запросу. Последовательности часто используются в базах данных, так как многие приложения требуют, чтобы каждая строка таблицы содержала уникальное значение, а это удобно делать при помощи последовательности. В этой главе рассказано о том, как работать с последовательностями в MySQL. Рассмотрены следующие вопросы:

Использование столбцов AUTO_INCREMENT для создания последовательностей. Столбец `AUTO_INCREMENT` – это тот механизм, который обеспечивает формирование последовательности для множества строк в MySQL. Каждый раз, когда вы создаете строку в таблице, содержащей столбец `AUTO_INCREMENT`, MySQL автоматически генерирует следующий номер последовательности и присваивает его значению столбца. Это значение служит уникальным идентификатором, так что благодаря последовательностям мы получаем простой способ создания таких элементов, как идентификационные номера клиентов, номера накладных на отгрузку, номера счетов или заказов на закупки, идентификаторы отчетов об ошибках, номера билетов или серийные номера продуктов.

Извлечение значений последовательности. Во многих приложениях недостаточно просто создать значения последовательности. Необходимо уметь определять значение последовательности для только что вставленной записи. Веб-приложению может понадобиться показать пользователю содержимое записи, созданной из содержимого формы, заполненной и переданной пользователем. Или же может потребоваться извлечь значение, чтобы его можно было хранить среди других записей в связанной таблице.

Способы повторного упорядочивания. Этот раздел описывает перенумерацию последовательности, содержащей «дыры», появившиеся из-за удаления записей, а также рассказывает о том, почему следует избегать повтор-

ной нумерации. Рассматривается возможность начала нумерации со значения, отличного от 1, и добавление столбца последовательности в таблицу, которая его не содержит.

Использование столбца AUTO_INCREMENT для создания нескольких последовательностей. Часто столбец `AUTO_INCREMENT` таблицы не зависит от других столбцов, и его значения монотонно возрастают на протяжении всей таблицы. Однако если вы создаете многостолбцовый индекс, который содержит столбец `AUTO_INCREMENT`, то можете использовать его для генерирования нескольких последовательностей. Например, если у вас есть доска объявлений, на которой сообщения распределяются по темам, вы можете последовательно нумеровать сообщения в рамках каждой темы, привязывая столбец `AUTO_INCREMENT` к столбцу индентификатора темы.

Управление несколькими значениями AUTO_INCREMENT одновременно. Будьте особенно внимательны при отслеживании нескольких значений последовательности. Такое возможно, например, при запуске ряда предложений, изменяющих одну таблицу, или при создании записей в нескольких таблицах, каждая из которых содержит столбец `AUTO_INCREMENT`. В разделе рассказано о том, как обрабатывать подобные ситуации.

Использование генераторов однострочных последовательностей. Последовательности могут использоваться как счетчики. Например, если у вас на сайте размещен баннер с рекламой, вы можете увеличивать счетчик при каждом нажатии (то есть при каждом переходе по рекламному объявлению). Количества нажатий для каждого конкретного баннера образуют последовательность, но интересует нас только текущий счетчик, так что нет необходимости в формировании новой строки для записи каждого нажатия. MySQL обеспечивает способ решения такой задачи за счет механизма, позволяющего последовательности генерироваться всего в одной строке таблицы. Чтобы хранить в таблице несколько счетчиков, добавьте столбец с уникальным идентификатором счетчика. Например, вы можете хранить в таблице произвольное количество счетчиков нажатий на баннеры. Каждая строка таблицы определяет какой-то баннер, а счетчик каждой строки увеличивается независимо от всех остальных. Аналогично

Генераторы последовательностей и переносимость

Большинство баз данных поддерживает формирование последовательностей, хотя их реализация обычно зависит от конкретной СУБД. Это касается и MySQL, так что материал раздела практически целиком является специфичным для MySQL, даже на уровне SQL. Другими словами, конструкции SQL, служащие для формирования последовательностей, сами по себе не переносимы, даже если использовать такие API, как DBI или JDBC, обеспечивающие абстрактный уровень. Абстрактные интерфейсы могут способствовать переносимой обработке предложений SQL, но сделать непереносимый SQL переносимым они не в силах.

можно создавать последовательности, которые увеличиваются не на единицу, а на какое-то другое значение, на непостоянную величину и даже на отрицательную величину.

Последовательная нумерация строк вывода. В разделе предлагаются способы формирования последовательностей только для отображения, используемых для нумерации строк результата запроса.

11.1. Использование AUTO_INCREMENT для создания столбца последовательности

Задача

Вы хотите включить в таблицу столбец последовательности.

Решение

Используйте столбец `AUTO_INCREMENT`.

Обсуждение

В разделе приводятся базовые сведения о работе столбцов `AUTO_INCREMENT`, начиная с простого примера, иллюстрирующего механизм формирования последовательности. Для примеров будем использовать ситуацию, сложившуюся вокруг коллекции жуков: вашему восьмилетнему сыну Джуниору в школе поручили собрать насекомых для школьного проекта. Для каждого насекомого Джуниор должен записать название («муравей», «пчела» и т. д.), дату появления в коллекции и происхождение экспоната. Вы с юных лет внушали Джуниору преимущества использования MySQL для хранения записей. И сегодня, как только вы пришли с работы, он тут же доложил о проекте и заявил, глядя на вас в упор, что MySQL отлично подходит для решения этой задачи. Спорить бесполезно. И вы вдвоем беретесь за работу. Джуниор уже нашел несколько экземпляров и, пока ждал вашего возвращения с работы, записал для них следующую информацию:

Name (Название)	Date (Дата)	Origin (Источник)
millipede (многоножка)	2001-09-10	driveway (подъездная дорожка)
housefly (муха)	2001-09-10	kitchen (кухня)
grasshopper (кузнечик)	2001-09-10	front yard (лужайка перед домом)
stink bug (лесной клоп)	2001-09-10	front yard
cabbage butterfly (бабочка-капустница)	2001-09-10	garden (сад)
ant (муравей)	2001-09-10	back yard (задний двор)
ant	2001-09-10	back yard
millbug (мокрица)	2001-09-10	under rock (под камнем)

Глядя на заметки Джуниора, вы радуетесь тому, что уже в таком нежном возрасте он умеет записывать даты в формате ISO. Однако вы также обращаете внимание на то, что в его коллекции присутствуют многоножка и мокрица, ни одна из которых на самом деле не является насекомым. Вы решаете отложить решение этого вопроса, так как Джуниор забыл принести домой письменные инструкции по выполнению задания и пока непонятно, можно ли включить в коллекцию эти экспонаты.

Размышляя о создании таблицы для хранения информации, вы понимаете, что необходимы как минимум столбцы `name`, `date` и `origin`, соответствующие тем видам данных, которые Джуниор должен записывать:

```
CREATE TABLE insect
(
    name    VARCHAR(30) NOT NULL,   # тип насекомого
    date    DATE NOT NULL,          # когда найдено
    origin  VARCHAR(30) NOT NULL    # где найдено
);
```

Однако этих столбцов может быть недостаточно для удобной работы с таблицей. Обратите внимание на то, что записи не уникальны: оба муравья (ant) были найдены в одно и то же время в одном месте. Если поместить информацию в таблицу `insect`, имеющую описанную выше структуру, то ни на одну запись о муравье нельзя будет сослаться индивидуально, так как их невозможно различить. Если ввести уникальные идентификаторы, можно будет отличать записи друг от друга и ссылаться на каждую из них по соответствующему значению. Удобно использовать столбец `AUTO_INCREMENT`, изменив структуру таблицы `insect` следующим образом:

```
CREATE TABLE insect
(
    id      INT UNSIGNED NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (id),
    name    VARCHAR(30) NOT NULL,   # тип насекомых
    date    DATE NOT NULL,          # когда найдено
    origin  VARCHAR(30) NOT NULL    # где найдено
);
```

Создадим таблицу, используя это второе определение. В рецепте 11.3 мы поговорим о том, почему столбец `id` объявлен именно так.

11.2. Генерирование значений последовательности

Задача

Теперь, когда у вас есть столбец `AUTO_INCREMENT`, хотелось бы использовать его для формирования нового значения последовательности.

Решение

Вставьте в столбец NULL или просто пропустите в предложении INSERT. И в том и в другом случае MySQL создаст для вас новый номер последовательности.

Обсуждение

Одним из полезных свойств столбца AUTO_INCREMENT является то, что вы не должны присваивать ему значения – это сделает MySQL. Есть два способа формирования новых значений AUTO_INCREMENT, о которых будет рассказано на примере столбца id таблицы insect. Во-первых, можно явно установить столбец id в NULL.[1] Следующее предложение вставляет четыре первых экспоната Джуниора в таблицу insect именно так:

```
mysql> INSERT INTO insect (id,name,date,origin) VALUES
    -> (NULL,'housefly','2001-09-10','kitchen'),
    -> (NULL,'millipede','2001-09-10','driveway'),
    -> (NULL,'grasshopper','2001-09-10','front yard'),
    -> (NULL,'stink bug','2001-09-10','front yard');
```

Во-вторых, можно полностью убрать столбец id из предложения INSERT. В MySQL можно создавать записи, не указывая явно значение каждого столбца. MySQL автоматически присваивает неуказанным столбцам значения по умолчанию, а умолчание для столбца AUTO_INCREMENT – это следующий номер последовательности. То есть вы можете вставлять записи в таблицу insect, не упоминая о столбце id. Это предложение добавляет четыре следующих экспоната Джуниора в таблицу insect вторым способом:

```
mysql> INSERT INTO insect (name,date,origin) VALUES
    -> ('cabbage butterfly','2001-09-10','garden'),
    -> ('ant','2001-09-10','back yard'),
    -> ('ant','2001-09-10','back yard'),
    -> ('millbug','2001-09-10','under rock');
```

Какой бы способ вы ни выбрали, MySQL определяет следующий номер последовательности для каждой записи и присваивает его столбцу id:

```
mysql> SELECT * FROM insect ORDER BY id;
+----+--------------------+------------+------------+
| id | name               | date       | origin     |
+----+--------------------+------------+------------+
|  1 | housefly           | 2001-09-10 | kitchen    |
|  2 | millipede          | 2001-09-10 | driveway   |
|  3 | grasshopper        | 2001-09-10 | front yard |
|  4 | stink bug          | 2001-09-10 | front yard |
|  5 | cabbage butterfly  | 2001-09-10 | garden     |
|  6 | ant                | 2001-09-10 | back yard  |
```

[1] В настоящее время установка столбца AUTO_INCREMENT в ноль имеет тот же эффект, что и установка в NULL. Но поскольку нет гарантии того, что так будет всегда, рекомендуется использовать NULL.

```
|  7 | ant               | 2001-09-10 | back yard  |
|  8 | millbug           | 2001-09-10 | under rock |
+----+-------------------+------------+------------+
```

Когда Джуниор найдет следующих насекомых, вы сможете добавить в таблицу новые записи, которым будут присвоены следующие значения последовательности (9, 10, ...).

Принцип, лежащий в основе формирования столбца AUTO_INCREMENT, достаточно прост: каждый раз, когда вы создаете новую строку, MySQL генерирует новый номер в последовательности и присваивает его строке. Но необходимо иметь в виду некоторые тонкости, а также особенности обработки последовательностей AUTO_INCREMENT для разных типов таблиц, тогда вы сможете более эффективно работать с последовательностями и избежать неприятных сюрпризов. Например, вот что может произойти, если вы явно устанавливаете столбец id в значение не-NULL:

- Если такое значение уже присутствует в таблице, выдается ошибка:

  ```
  mysql> INSERT INTO insect (id,name,date,origin) VALUES
      -> (3,'cricket','2001-09-11','basement');
  ERROR 1062 at line 1: Duplicate entry '3' for key 1
  ```

  Она возникает, так как при создании столбца AUTO_INCREMENT вы должны создать для него индекс PRIMARY KEY или UNIQUE, поэтому он не может содержать повторяющиеся значения.

- Если значение не присутствует в таблице, MySQL вставляет запись с этим значением. Кроме того, если оно больше текущего значения счетчика последовательности, то счетчику присваивается значение, на единицу превышающее вставленное. Сейчас таблица insect содержит значения последовательности от 1 до 8. Если вы вставляете новую строку, в которой столбец id установлен в 20, это значение становится новым максимумом. При последующих вставках, автоматически формирующих значения id, значения будут начинаться с 21. В результате значения с 9 по 19 останутся неиспользованными, и в последовательности возникнет разрыв.

Теперь давайте подробнее поговорим о том, как определить столбцы AUTO_INCREMENT и как они себя ведут.

11.3. Выбор типа для столбца последовательности

Задача

Вы хотите побольше узнать о том, как определить столбец последовательности.

Решение

Используйте приведенные в рецепте инструкции.

Обсуждение

При создании столбца AUTO_INCREMENT необходимо следовать некоторым правилам. В качестве примера будем рассматривать объявление столбца id таблицы insect:

```
id INT UNSIGNED NOT NULL AUTO_INCREMENT,
PRIMARY KEY (id)
```

Ключевое слово AUTO_INCREMENT информирует MySQL о том, что необходимо генерировать последовательные порядковые значения для значений столбца. Кроме того, указывается следующая информация:

- INT – это базовый тип столбца. Необязательно использовать именно INT, но столбец должен относиться к одному из целых типов: TINYINT, SMALLINT, MEDIUMINT, INT или BIGINT. Важно помнить о том, что AUTO_INCREMENT – это атрибут столбца, применимый *только* к целым типам. Ранние версии MySQL позволяют создавать столбец AUTO_INCREMENT и для нецелых типов, таких как CHAR, но ничего хорошего из этого не получится. (Даже если кажется, что номера последовательности генерируются нормально, рано или поздно произойдет сбой. Типичная ошибка – «duplicate key» (повторный ключ) – возникает после вставки нескольких записей, когда вы уверены в том, что столбец еще может хранить новые номера.) Не создавайте себе проблем и всегда используйте для столбца AUTO_INCREMENT целый тип.
- Столбец объявляется как UNSIGNED. Нет смысла разрешать отрицательные значения, так как последовательности AUTO_INCREMENT всегда состоят только из положительных целых (обычно начиная с 1). Более того, если *не* объявить столбец как UNSIGNED, то диапазон значений вашей последовательности уменьшится наполовину. Например, тип TINYINT имеет диапазон от -128 до 127. Последовательности включают только положительные значения, так что значения последовательности TINYINT будут находится в промежутке от 1 до 127. Диапазон же столбца TINYINT без знака – от 0 до 255, и наибольшее значение последовательности сдвигается к 255. Максимальное значение последовательности определяется конкретным используемым целым типом, поэтому следует выбирать тип, достаточно емкий для того, чтобы вместить необходимое максимальное значение. Максимальное значение без знака, допускаемое каждым целым типом, приведено в табл. 11.1:

Таблица 11.1. Максимальные значения целых типов

Тип столбца	Максимальное значение без знака
TINYINT	255
SMALLINT	65 535
MEDIUMINT	16 777 215
INT	4 294 967 295
BIGINT	18 446 744 073 709 551 615

Некоторые предпочитают не использовать UNSIGNED, чтобы столбец последовательности мог содержать отрицательные числа (например, используя -1, чтобы обозначить случай «нет идентификатора»). MySQL не предоставляет никаких гарантий относительно обработки отрицательных чисел, так что, используя их в столбце AUTO_INCREMENT, вы играете с огнем. Например, если вы реинициализируете столбец, то обнаружите, что все ваши отрицательные значение преобразованы в обычные (положительные) номера последовательности.

- Столбец AUTO_INCREMENT не может содержать значения NULL, поэтому id объявляется как NOT NULL. (Да, вы можете указать NULL в качестве значения столбца при вставке новой записи, но для AUTO_INCREMENT это будет означать «генерировать следующее значение последовательности».) Текущие версии MySQL автоматически определяют столбцы AUTO_INCREMENT как NOT NULL, если вы забудете это сделать. Однако лучше явно указать NOT NULL в предложении CREATE TABLE, если есть вероятность того, что однажды придется использовать его со старой версией MySQL.
- Столбец объявляется как PRIMARY KEY, чтобы обеспечить уникальность всех его значений. У таблицы может быть только один первичный ключ, так что если в таблице какой-то другой столбец уже определен как PRIMARY KEY, вы можете объявить столбец AUTO_INCREMENT как уникальный (UNIQUE):

```
id INT UNSIGNED NOT NULL AUTO_INCREMENT,
UNIQUE (id)
```

Если столбец AUTO_INCREMENT – это единственный столбец в индексе PRIMARY KEY или UNIQUE, вы можете объявить это в определении столбца, а не в отдельной инструкции. Например, эквиваленты такие определения:

```
id INT UNSIGNED NOT NULL AUTO_INCREMENT,
PRIMARY KEY (id)

id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY
Как и такие:
id INT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE

id INT UNSIGNED NOT NULL AUTO_INCREMENT,
UNIQUE (id)
```

Использование отдельной инструкции для указания индекса подчеркивает то, что, строго говоря, это не часть определения столбца. (Если вы читали главу 8, то должны были заметить, что изменение индексов столбцов рассматривается отдельно от изменения определения самого столбца.)

При создании таблицы, содержащей столбец AUTO_INCREMENT, необходимо учитывать тип таблицы (MyISAM, InnoDB и т. д.). Тип таблицы определяет такие свойства, как повторное использование значений, удаляемых из вершины последовательности, и возможность установки начального значения последовательности. В общем, MyISAM – это наиболее удобный тип для таблиц, содержащих столбец AUTO_INCREMENT, так как он поддерживает наибольший объем возможностей по управлению последовательностями. Дочитав главу до конца, вы согласитесь со мной.

11.4. Удаление записей и формирование последовательности

Задача

Вы хотите знать, что происходит с последовательностью, когда вы удаляете записи из таблицы, содержащей столбец AUTO_INCREMENT.

Решение

Все зависит от типа таблицы и от того, какие записи удаляются.

Обсуждение

Пока что мы говорили только о генерировании значений последовательности для столбца AUTO_INCREMENT при добавлении записей в таблицу. Но не стоит надеяться на то, что записи никогда не будут удаляться. Что же тогда произойдет с последовательностью?

Снова обратимся к проекту Джуниора по сбору коллекции насекомых, для которого у нас пока есть такая таблица insect:

```
mysql> SELECT * FROM insect ORDER BY id;
+----+-------------------+------------+------------+
| id | name              | date       | origin     |
+----+-------------------+------------+------------+
|  1 | housefly          | 2001-09-10 | kitchen    |
|  2 | millipede         | 2001-09-10 | driveway   |
|  3 | grasshopper       | 2001-09-10 | front yard |
|  4 | stink bug         | 2001-09-10 | front yard |
|  5 | cabbage butterfly | 2001-09-10 | garden     |
|  6 | ant               | 2001-09-10 | back yard  |
|  7 | ant               | 2001-09-10 | back yard  |
|  8 | millbug           | 2001-09-10 | under rock |
+----+-------------------+------------+------------+
```

Теперь ее надо изменить, потому что после того, как Джуниор наконец донес до дома письменные указания по выполнению работы, вы обнаружили там два факта, связанных с содержимым таблицы insect:

- Экспонатами могут быть только насекомые, а не те, кто на них похож, как многоножки (millipede) и мокрицы (millbug).
- Идея проекта в том, чтобы собрать как можно больше разных экспонатов, а не просто как можно больше насекомых. То есть допустима только одна запись для муравья (ant).

Правила требуют удаления нескольких строк из таблицы insect, а именно, записей со значениями id, равными 2 (millipede), 8 (millbug) и 7 (дубликат для ant). Итак, несмотря на понятное разочарование Джуниора в связи с уменьшением его коллекции, вы поручаете ему удалить записи, используя предложение DELETE:

```
mysql> DELETE FROM insect WHERE id IN (2,8,7);
```

Одна из причин, по которой полезны уникальные значения идентификаторов, – они позволяют однозначно идентифицировать каждую запись. Записи о муравьях идентичны во всем, кроме идентификаторов. Если бы в таблице `insect` не было такого столбца, было бы сложнее удалить одну запись.

После удаления неподходящих записей результирующая таблица будет выглядеть так:

```
mysql> SELECT * FROM insect ORDER BY id;
+----+-------------------+------------+------------+
| id | name              | date       | origin     |
+----+-------------------+------------+------------+
|  1 | housefly          | 2001-09-10 | kitchen    |
|  3 | grasshopper       | 2001-09-10 | front yard |
|  4 | stink bug         | 2001-09-10 | front yard |
|  5 | cabbage butterfly | 2001-09-10 | garden     |
|  6 | ant               | 2001-09-10 | back yard  |
+----+-------------------+------------+------------+
```

Теперь последовательность в столбце `id` имеет пропуск (нет строки 2), а значения 7 и 8 на вершине последовательности отсутствуют. Как эти удаления повлияют на последующие операции вставки? Какой номер последовательности получит следующая строка?

Удаление строки 2 привело к созданию пропуска в середине последовательности. Это никак не влияет на последующие операции вставки, так как MySQL не предпринимает попыток заполнения «дыр». Но удаление строк 7 и 8 удаляет значения верхушки последовательности, и результат такой операции определятся типом таблицы:

- В таблицах ISAM и BDB следующий номер последовательности – это всегда наименьшее положительное целое, не присутствующее в столбце. Если вы удаляете строки, содержащие максимальные значения последовательности, эти значения будут использованы повторно. (То есть после удаления записей со значениями 7 и 8 следующей вставленной записи будет присвоено значение 7.)
- В таблицах MyISAM и InnoDB значения повторно не используются. Следующий номер последовательности – это наименьшее положительное целое, которое ранее не использовалось. (Для последовательности, остановившейся на значении 8, следующая запись получит значение 9, даже если предварительно будут удалены записи 7 и 8.) Если вам нужны строго монотонные последовательности, используйте один из этих типов таблиц.

Таблицы ISAM – это единственный тип таблиц, доступных в MySQL до версии 3.23, так что до этой версии вы можете только повторно использовать значения, удаленные из верхушки последовательности. Таблицы MyISAM появляются в версии MySQL 3.23 (и с этого же момента MyISAM становится типом таблицы по умолчанию). Таблицы BDB и InnoDB доступны, начиная с версии MySQL 3.23.17 и 3.23.29 соответственно.

Если вы используете таблицу, тип которой обеспечивает не то поведение в отношении повторного использования значений последовательности, которое вам необходимо, используйте предложение ALTER TABLE для изменения типа таблицы на более подходящий. Например, если вы хотите изменить тип таблицы с ISAM на MyISAM (чтобы не допустить повторного использования значений после удаления записей), сделайте следующее:

```
ALTER TABLE имя_таблицы TYPE = MYISAM;
```

Если вы не знаете тип вашей таблицы, используйте предложение SHOW TABLE STATUS:

```
mysql> SHOW TABLE STATUS LIKE 'insect'\G;
*************************** 1. row ***************************
           Name: insect
           Type: MyISAM
     Row_format: Dynamic
           Rows: 7
 Avg_row_length: 30
    Data_length: 216
Max_data_length: 4294967295
   Index_length: 2048
      Data_free: 0
 Auto_increment: 8
    Create_time: 2002-01-25 16:55:32
    Update_time: 2002-01-25 16:55:32
     Check_time: NULL
 Create_options:
        Comment:
```

Вывод показывает, что таблица insect относится к типу MyISAM. (Можно было использовать SHOW CREATE TABLE.)

В этой главе будем считать, что если таблица создается без явного указания типа, то это таблица MyISAM.

Особым случаем удаления записей является очистка всей таблицы при помощи предложения DELETE без инструкции WHERE:

```
DELETE FROM имя_таблицы;
```

В этом случае счетчик последовательности может быть заново установлен в 1, даже для тех типов таблиц, которые обычно не используют значения повторно (MyISAM и InnoDB). Для таких типов, если вы хотите удалить все записи, сохранив текущее значение последовательности, укажите MySQL на необходимость выполнения построчного удаления, добавив инструкцию WHERE с заведомо истинным условием:

```
DELETE FROM имя_таблицы WHERE 1 > 0;
```

11.5. Извлечение значений последовательности

Задача

После создания записи, включающей новый номер последовательности, вы хотите узнать, что это был за номер.

Решение

В предложении SQL можно использовать функцию LAST_INSERT_ID(). Если вы пишете программу, то ваш API для MySQL может предлагать какой-то способ получения значения без непосредственного использования LAST_INSERT_ID().

Обсуждение

Во многих приложениях требуется определить значение AUTO_INCREMENT только что созданной записи. Например, если вы честолюбивы и беретесь за создание веб-интерфейса для ввода записей в таблицу insect Джуниора, то можете сделать так, чтобы приложение отображало каждую новую запись красиво отформатированной на новой странице сразу же после того, как вы нажимаете кнопку Submit. Для этого необходимо знать новое значение id, чтобы извлечь соответствующую запись. Другая распространенная ситуация, в которой требуется значение AUTO_INCREMENT, возникает, когда используется несколько таблиц: после вставки записи в главную таблицу обычно необходимо извлечь ее идентификатор, чтобы создать в других таблицах записи, ссылающиеся на главную. (В рецепте 11.15 показано, как связать несколько таблиц посредством номеров последовательности.)

Когда генерируется новое значение AUTO_INCREMENT, вы можете получить значение от сервера, создав запрос, вызывающий функцию LAST_INSERT_ID(). Кроме того, многие API для MySQL предоставляют на стороне клиента механизм для обращения к значению без запуска дополнительного запроса. В этом разделе описаны оба способа и рассмотрены их отличия.

Получение значений AUTO_INCREMENT с помощью функции LAST_INSERT_ID()

Очевидный (но неверный) способ определения значения AUTO_INCREMENT новой записи использует тот факт, что генерируемое MySQL значение становится максимальным номером последовательности в столбце. То есть можно попробовать использовать для его извлечения функцию MAX():

```
SELECT MAX(id) FROM insect;
```

Но такой способ ненадежен, так как он не принимает в расчет многопоточность сервера MySQL. Запрос SELECT действительно возвращает максимальное значение id в таблице, но это значение вполне может быть сформировано *не вами*. Предположим, что вы вставляете запись, которая получает значение id, равное 9. Если другое клиентское приложение вставит запись, прежде

чем вы запустите запрос SELECT, значением MAX(id) будет 10, а не 9. Для решения проблемы можно сгруппировать предложения INSERT и SELECT в транзакцию или заблокировать таблицу, но MySQL предоставляет и более простой способ получения правильного значения – функцию LAST_INSERT_ID(). Она возвращает последнее из значений AUTO_INCREMENT, сгенерированное вами в рамках соединения с сервером. Например, можно вставить запись в таблицу insect, а затем извлечь ее значение id так:

```
mysql> INSERT INTO insect (name,date,origin)
    -> VALUES('cricket','2001-09-11','basement');
mysql> SELECT LAST_INSERT_ID();
+------------------+
| last_insert_id() |
+------------------+
|                9 |
+------------------+
```

Или можно использовать новое значение для извлечения всей записи, даже не зная ее идентификатора:

```
mysql> INSERT INTO insect (name,date,origin)
    -> VALUES('moth','2001-09-14','windowsill');
mysql> SELECT * FROM insect WHERE id = LAST_INSERT_ID();
+----+------+------------+------------+
| id | name | date       | origin     |
+----+------+------------+------------+
| 10 | moth | 2001-09-14 | windowsill |
+----+------+------------+------------+
```

Могут ли другие клиенты изменить значение, возвращаемое функцией LAST_INSERT_ID()?

Такой вопрос может возникнуть, если вас беспокоит возможность получения некорректного значения от LAST_INSERT_ID() в том случае, когда одновременно с вами другие клиентские приложения генерируют значения AUTO_INCREMENT. Но волноваться не о чем. Значение, возвращаемое функцией LAST_INSERT_ID(), определяется сервером в рамках соединения. Это свойство чрезвычайно важно, так как оно не дает клиентским приложениям мешать друг другу. Когда вы формируете значение AUTO_INCREMENT, функция LAST_INSERT_ID() возвращает именно его, даже если другие клиенты тем временем вносят новые записи в ту же таблицу.

Использование API для получения значений AUTO_INCREMENT

LAST_INSERT_ID() – это функция SQL, так что вы можете использовать ее в любом клиентским приложении, умеющем создавать предложения SQL. Но вам приходится выполнять специальный запрос для получения ее значения. Если вы пишете собственную программу, то можете пойти другим путем. Многие

интерфейсы MySQL включают специальное расширение, которое возвращает значение AUTO_INCREMENT без запуска дополнительного запроса. В частности, такой способностью обладают все наши четыре API.

Perl

Используйте атрибут mysql_insertid для получения значения AUTO_INCREMENT, сгенерированного запросом. Доступ к этому атрибуту можно получить или через дескриптор (handle) базы данных, или через дескриптор предложения, в зависимости от способа запуска запроса. Посмотрим, как выполнить операцию при помощи дескриптора базы данных:

```
$dbh->do ("INSERT INTO insect (name,date,origin)
           VALUES('moth','2001-09-14','windowsill')");
my $seq = $dbh->{mysql_insertid};
```

Если вы используете для запуска запроса prepare() и execute(), обращайтесь к mysql_insertid как к атрибуту дескриптора предложения:

```
my $sth = $dbh->prepare ("INSERT INTO insect (name,date,origin)
                          VALUES('moth','2001-09-14','windowsill')");
$sth->execute ();
my $seq = $sth->{mysql_insertid};
```

Если окажется, что значение атрибута mysql_insertid всегда равно нулю, вероятно, вы работаете со старой версией *DBD::mysql*, которая его не поддерживает. Тогда попробуйте применить атрибут insertid. (В этом случае insertid доступен только как атрибут дескриптора базы данных.)

PHP

После запуска запроса, формирующего значение AUTO_INCREMENT, извлеките значение, вызвав mysql_insert_id():

```
mysql_query ("INSERT INTO insect (name,date,origin)
           VALUES('moth','2001-09-14','windowsill')", $conn_id);
$seq = mysql_insert_id ($conn_id);
```

Аргументом функции является идентификатор соединения. Если аргумент не указан, mysql_insert_id() использует последнее открытое соединение.

Python

Драйвер MySQLdb для API DB предоставляет метод курсора insert_id() для получения значений последовательности. Используйте его с объектом курсора, через который выполняется запрос, формирующий значение AUTO_INCREMENT:

```
cursor = conn.cursor ()
cursor.execute ("""
        INSERT INTO insect (name,date,origin)
        VALUES('moth','2001-09-14','windowsill')
```

```
    """)
seq = cursor.insert_id ()
```

Java

Драйвер JDBC MySQL Connector/J предлагает метод `getLastInsertID()` для получения значений `AUTO_INCREMENT`. Метод может использоваться с объектами как `Statement`, так и `PreparedStatement`. Рассмотрим на примере `Statement`:

```
Statement s = conn.createStatement ();
s.executeUpdate (
            "INSERT INTO insect (name,date,origin)"
            + " VALUES('moth','2001-09-14','windowsill')");
long seq = ((com.mysql.jdbc.Statement) s).getLastInsertID ();
s.close ();
```

Заметим, что поскольку `getLastInsertID()` зависит от драйвера, для получения доступа к методу необходимо преобразовать объект `Statement` к типу `com.mysql.jdbc.Statement`. Если используется объект `PreparedStatement`, приведите его к типу `com.mysql.jdbc.PreparedStatement`:

```
PreparedStatement s = conn.prepareStatement (
                "INSERT INTO insect (name,date,origin)"
                + " VALUES('moth','2001-09-14','windowsill')");
s.executeUpdate ();
long seq = ((com.mysql.jdbc.PreparedStatement) s).getLastInsertID ();
s.close ();
```

Сравнение серверного и клиентского способов извлечения значений последовательности

Как уже говорилось, значение `LAST_INSERT_ID()` поддерживается в рамках соединения на серверной стороне соединения MySQL. А API-способы, обеспечивающие прямой доступ к значениям `AUTO_INCREMENT`, реализованы на клиентской стороне. Серверный и клиентский приемы извлечения значений последовательности имеют как сходства, так и различия.

Оба способа (и клиентский, и серверный) обладают общим свойством: вы должны обращаться за значением `AUTO_INCREMENT` в пределах того же соединения MySQL, в котором это значение было сгенерировано. Если вы формируете значение `AUTO_INCREMENT`, затем отключаетесь от сервера и вновь устанавливаете соединение с тем, чтобы получить значение, то получите ноль. Но продолжительность жизни значений `AUTO_INCREMENT` на стороне сервера может быть гораздо больше:

- После запуска запроса, формирующего значение `AUTO_INCREMENT`, значение остается доступным для функциии `LAST_INSERT_ID()`, даже если будут запускаться другие предложения, при условии, что ни одно из них не генерирует значение `AUTO_INCREMENT`.
- Значение последовательности, доступное на клиентской стороне соединения, обычно устанавливается *для всех* запросов, а не только для генериру-

ющих значения AUTO_INCREMENT. Если запускается предложение INSERT, формирующее новое значение, затем выполняется какой-то другой запрос, то, обратившись за значением последовательности на клиентской стороне, вы, вероятно, получите ноль. Конкретное положение дел зависит от используемого API, но для надежности я бы рекомендовал руководствоваться таким общим правилом: если запрос формирует значение последовательности, которое вы не планируете сразу же использовать, сохраните его в переменной, на которую вы сможете затем сослаться. В противном случае при обращении за значением последовательности может оказаться, что оно уже ликвидировано.

11.6. Стоит ли повторно упорядочивать столбец

Задача

В столбце последовательности есть пропуски, и вы думаете о том, стоит ли повторно упорядочить его.

Решение

Не беспокойтесь об этом. Или по крайней мере не делайте этого без достаточных на то оснований, которых очень немного.

Обсуждение

Если вы вставляете записи в таблицу, содержащую столбец AUTO_INCREMENT, и никогда их не удаляете, то значения столбца образуют непрерывную последовательность. Но если вы удаляете записи, в последовательности начинают появляться пустоты. Например, таблица Джуниора insect сейчас выглядит примерно так, с «дырами» в последовательности (предполагается, что были вставлены записи про сверчка (cricket) и мотылька (moth), упомянутые в предыдущем разделе про извлечение значений последовательности):

```
mysql> SELECT * FROM insect ORDER BY id;
+----+-------------------+------------+------------+
| id | name              | date       | origin     |
+----+-------------------+------------+------------+
|  1 | housefly          | 2001-09-10 | kitchen    |
|  3 | grasshopper       | 2001-09-10 | front yard |
|  4 | stink bug         | 2001-09-10 | front yard |
|  5 | cabbage butterfly | 2001-09-10 | garden     |
|  6 | ant               | 2001-09-10 | back yard  |
|  9 | cricket           | 2001-09-11 | basement   |
| 10 | moth              | 2001-09-14 | windowsill |
+----+-------------------+------------+------------+
```

MySQL не будет пытаться заполнить пропуски неиспользованными значениями при вставке новых записей. Те, кому не нравится такое поведение, обычно периодически переупорядочивают столбцы AUTO_INCREMENT для удаления

пустот. В следующих разделах показано, как это сделать. Кроме того, рассказано о том, как добавить столбец последовательности в таблицу, в которой его еще нет, инициировать повторное использование удаленных значений вершины последовательности и указать начальное значение последовательности при создании или повторном упорядочивании таблицы.

Причины избегать повторного упорядочивания

Прежде чем принять решение о повторном упорядочивании столбца AUTO_INCREMENT, подумайте о том, действительно ли вы этого хотите и есть ли такая необходимость. В большинстве случаев это совсем необязательно. На самом деле, повторная нумерация последовательности может создать серьезные проблемы. Например, *нельзя* переупорядочивать столбец значений последовательности, на который ссылается другая таблица. Изменение нумерации разрушает связь значений со значениями в другой таблице, делая невозможным корректное сопоставление записей двух таблиц.

Причины, по которым стремятся к повторному упорядочиванию:

Эстетика. Иногда хочется перенумеровать столбец из эстетических соображения. Неразрывные последовательности выглядят более красиво, чем последовательности с дырами. Если ваша причина именно такова, я вряд ли смогу вас переубедить. Тем не менее, это не самая убедительная причина.

Производительность. Стремление к повторному упорядочиванию может объясняться мнением о том, что удаление пробелов делает столбец последовательности более компактным и позволяет запросам MySQL выполняться быстрее. Но это неверно. Наличие или отсутствие пробелов не беспокоит MySQL, и перенумерация столбца AUTO_INCREMENT не увеличивает производительность. Можно даже сказать, что повторное упорядочивание отрицательно влияет на производительность в том смысле, что таблица остается заблокированной на время выполнения операции, которая может быть достаточно долгой для больших таблиц. Другие клиенты могут в это время читать данные из таблицы, и если они захотят вставить новые строки, то придется подождать завершения операции.

Нехватка номеров. Верхняя граница значений столбца последовательности определяется типом данных столбца. Если последовательность AUTO_INCREMENT приближается к верхней границе, перенумерация освобождает значения верхушки последовательности. Это разумная причина повторного упорядочивания столбца, но все же во многих случаях и в этом нет необходимости. Можно расширить диапазон значений столбца для увеличения его верхней границы без изменения хранимых значений.

11.7. Расширение диапазона последовательности

Задача

Вы не хотите повторно упорядочивать столбец, но вам не хватает номеров последовательности для новых строк.

Решение

Посмотрите, можно ли изменить тип столбца на UNSIGNED или на более широкий целый тип.

Обсуждение

Повторное упорядочивание столбца AUTO_INCREMENT изменяет содержимое практически каждой строки таблицы. Часто этого удается избежать, расширив диапазон столбца, то есть изменив структуру таблицы, а не ее содержимое:

- Если столбец имеет тип со знаком, измените его на UNSIGNED, увеличив тем самым диапазон разрешенных значений вдвое. Предположим, что у вас есть столбец id, который в настоящий момент определен так:

  ```
  id MEDIUMINT NOT NULL AUTO_INCREMENT
  ```

  Верхняя граница столбца MEDIUMINT со знаком составляет 8 388 607. Ее можно увеличить до 16 777 215, убрав у столбца знак при помощи предложения ALTER TABLE:

  ```
  ALTER TABLE имя_таблицы MODIFY id MEDIUMINT UNSIGNED NOT NULL AUTO_ INCREMENT;
  ```

- Если столбец уже имеет тип UNSIGNED, но относится еще не к самому широкому из целых типов (BIGINT), преобразуем его, увеличив тем самым диапазон. Для этого также можно использовать ALTER TABLE. Например, тип столбца id из предыдущего примера можно прообразовать из MEDIUMINT в BIGINT так:

  ```
  ALTER TABLE имя_таблицы MODIFY id BIGINT UNSIGNED NOT NULL AUTO_INCREMENT;
  ```

В рецепте 11.3 приведена таблица, содержащая диапазоны всех целых типов, которую вы можете использовать при выборе типа столбца.

11.8. Перенумерация существующей последовательности

Задача

Несмотря на мои рекомендации вы все-таки решили повторно упорядочить столбец последовательности.

Решение

Удалите столбец из таблицы. Затем вставьте снова. MySQL сделает последовательность номеров непрерывной.

Обсуждение

Если вы считаете, что повторного упорядочивания столбца AUTO_INCREMENT не избежать, удалите столбец из таблицы и заново вставьте его. Следующий пример показывает, как перенумеровать значения id таблицы insect:

```
mysql> ALTER TABLE insect DROP id;
mysql> ALTER TABLE insect
    -> ADD id INT UNSIGNED NOT NULL AUTO_INCREMENT FIRST,
    -> ADD PRIMARY KEY (id);
```

Первое предложение ALTER TABLE удаляет столбец id (в результате чего удаляется и PRIMARY KEY, так как столбца, на который он ссылается, больше нет). Второе предложение восстанавливает столбец в таблице и делает его индексом PRIMARY KEY. (Ключевое слово FIRST помещает столбец на первое место в таблице, где он и располагался изначально. Обычно же ADD помещает столбцы в конец таблицы.) При добавлении в таблицу столбца AUTO_INCREMENT MySQL автоматически нумерует все строки последовательно, так что в итоге содержимое таблицы insect выглядит так:

```
mysql> SELECT * FROM insect ORDER BY id;
+----+--------------------+------------+------------+
| id | name               | date       | origin     |
+----+--------------------+------------+------------+
|  1 | housefly           | 2001-09-10 | kitchen    |
|  2 | grasshopper        | 2001-09-10 | front yard |
|  3 | stink bug          | 2001-09-10 | front yard |
|  4 | cabbage butterfly | 2001-09-10 | garden     |
|  5 | ant                | 2001-09-10 | back yard  |
|  6 | cricket            | 2001-09-11 | basement   |
|  7 | moth               | 2001-09-14 | windowsill |
+----+--------------------+------------+------------+
```

Проблема повторного упорядочивания столбца при помощи двух предложений ALTER TABLE в том, что в промежутке между операциями таблица остается без столбца. Это может создать трудности для других клиентов, пытающихся обратиться к таблице в течение данного промежутка времени. Чтобы не допустить подобной ситуации, выполним обе операции в одном предложении ALTER TABLE:

```
mysql> ALTER TABLE insect
    -> DROP id,
    -> ADD id INT UNSIGNED NOT NULL AUTO_INCREMENT FIRST,
    -> AUTO_INCREMENT = 1;
```

MySQL разрешает выполнять в одном предложении ALTER TABLE несколько действий (что работает не во всех СУБД). Однако обратите внимание на то, что это многооперационное предложение представляет собой не просто объединение двух простых предложений ALTER TABLE, а имеет два отличия:

- Нет необходимости в восстановлении первичного ключа, так как MySQL не удаляет его до тех пор, пока, после выполнения всех действий, указанных в предложении ALTER TABLE, не удален индексированный столбец.
- Инструкция AUTO_INCREMENT указывает на то, что нумерация должна начаться со значения 1. На самом деле это необходимо только до версии 3.23.39 из-за возникавшей ошибки MySQL: счетчик последовательности не возвращался в начальное положение при удалении и добавлении столбца в

одном предложении ALTER TABLE. (Например, если у вас была таблица со значениями 8, 12 и 14, то без инструкции AUTO_INCREMENT новая последовательность была бы пронумерована как 15, 16, 17.)

11.9. Повторное использование последних значений последовательности

Задача

Вы удалили строки с максимальными номерами последовательности. Можно ли избежать повторного упорядочивания столбца, но при этом заново использовать удаленные значения?

Решение

Да, используйте предложение ALTER TABLE для изменения значения счетчика последовательности. MySQL будет формировать новые номера последовательности, начиная со значения, следующего за текущим максимальным значением, имеющимся в таблице.

Обсуждение

Если вы удалили записи только из верхушки последовательности, то оставшиеся все еще образуют непрерывную последовательность (например, если у вас были записи с номерами от 1 до 100, и вы удалили записи с 91 по 100, то оставшиеся представляют собой монотонную последовательность без разрывов от 1 до 90.) В этом случае нет необходимости в повторной нумерации столбца. Просто сообщите MySQL о том, что нужно возобновить нумерацию со значения, на единицу большего, чем текущий наибольший номер последовательности. Таблицы ISAM и BDB ведут себя так по умолчанию, так что удаленные значения будут использованы повторно без каких-то дополнительных действий с вашей стороны. Для таблиц MyISAM и InnoDB выполните такое предложение:

```
ALTER TABLE имя_таблицы AUTO_INCREMENT = 1
```

Тогда MySQL уменьшит счетчик последовательности настолько, насколько это возможно, для создания новых записей.

Вы можете использовать ALTER TABLE для установки нового счетчика последовательности и в тех случаях, когда столбец имеет пропуски в середине, но повторно использованы будут только значения, удаленные из верхушки последовательности. Пропуски не будут заполнены. Предположим, что у вас есть таблица со значениями последовательности от 1 до 10, из которой удаляются записи со значениями 3, 4, 5, 9 и 10. Максимальное из оставшихся значений – это 8, так что если вы выполните ALTER TABLE для переназначения счетчика последовательности, то следующая запись получит в качестве значения последовательнъости 9, но не 3. Изменение нумерации последовательности с заполнением «дыр» описано в рецепте 11.8.

11.10. Управление изменением нумерации строк

Задача

Вы упорядочили столбец повторно, но MySQL нумерует строки не в том порядке, как вам хотелось бы.

Решение

Отберите строки в другую таблицу, используя инструкцию `ORDER BY` для размещения их в нужном порядке, и позвольте MySQL пронумеровать их. Строки будут пронумерованы в порядке сортировки.

Обсуждение

Когда вы повторно упорядочиваете столбец `AUTO_INCREMENT`, MySQL может выбирать строки таблицы в любом порядке, который совсем не обязательно совпадет с ожидаемым вами. Это не имеет значения, если важно только наличие у каждой строки уникального идентификатора. Но бывают приложения, в которых необходимо, чтобы строки были пронумерованы в каком-то определенном порядке. Например, вы можете захотеть, чтобы последовательность соответствовала порядку создания строк, указанному в столбце `TIMESTAMP`. Для присваивания номеров в определенном порядке выполните такую процедуру:

1. Создайте пустой клон таблицы.
2. Скопируйте туда строки исходной таблицы, используя `INSERT INTO ... SELECT`. Копируйте все столбцы, кроме столбца последовательности, применяя инструкцию `ORDER BY` для указания порядка копирования строк (и, следовательно, присваивания номеров последовательности).
3. Удалите исходную таблицу и переименуйте клон, присвоив ему имя удаленной таблицы.
4. Если таблица большая и имеет много индексов, эффективнее сначала создать новую таблицу только с одним индексом для столбца `AUTO_INCREMENT`, затем скопировать исходную таблицу в новую, а потом уже добавить недостающие индексы.

А можно поступить и по-другому:

1. Создать новую таблицу, содержащую все столбцы исходной, кроме `AUTO_INCREMENT`.
2. Скопировать столбцы не-`AUTO_INCREMENT` из исходной таблицы в новую с помощью `INSERT INTO ... SELECT`.
3. Удалить строки из исходной таблицы и (если необходимо) установить счетчик последовательности в 1.
4. Скопировать строки из новой таблицы обратно в исходную, используя инструкцию `ORDER BY` для сортировки строк в том порядке, в котором им должны быть присвоены номера последовательности.

Информация о создании клона таблицы приведена в рецепте 3.25. Если вы выполняете одну из описанных выше процедур в программе, которая не имеет достаточных сведений о структуре таблицы, используйте метаданные таблицы для получения списка имен столбцов и определения того, какой из столбцов имеет атрибут AUTO_INCREMENT (см. рецепт 9.5).

11.11. Как начать последовательность с определенного значения

Задача

Последовательности начинаются с 1, но вам хотелось бы использовать другое начальное значение.

Решение

Добавьте инструкцию AUTO_INCREMENT в предложение CREATE TABLE при создании таблицы. Если таблица уже создана, выполните предложение ALTER TABLE для установки начального значения.

Обсуждение

По умолчанию последовательности AUTO_INCREMENT начинаются с 1:

```
mysql> CREATE TABLE t
    -> (id INT UNSIGNED NOT NULL AUTO_INCREMENT, PRIMARY KEY (id));
mysql> INSERT INTO t (id) VALUES(NULL);
mysql> INSERT INTO t (id) VALUES(NULL);
mysql> INSERT INTO t (id) VALUES(NULL);
mysql> SELECT id FROM t ORDER BY id;
+----+
| id |
+----+
|  1 |
|  2 |
|  3 |
+----+
```

В таблице MyISAM можно начинать последовательность со значения *n*, добавив инструкцию AUTO_INCREMENT = *n* в конец предложения CREATE TABLE:

```
mysql> CREATE TABLE t
    -> (id INT UNSIGNED NOT NULL AUTO_INCREMENT, PRIMARY KEY (id))
    -> AUTO_INCREMENT = 100;
mysql> INSERT INTO t (id) VALUES(NULL);
mysql> INSERT INTO t (id) VALUES(NULL);
mysql> INSERT INTO t (id) VALUES(NULL);
mysql> SELECT id FROM t ORDER BY id;
+-----+
| id  |
+-----+
| 100 |
```

```
| 101 |
| 102 |
+-----+
```

Есть и другой способ: вы можете создать таблицу, а затем установить начальное значение последовательности при помощи ALTER TABLE:

```
mysql> CREATE TABLE t
    -> (id INT UNSIGNED NOT NULL AUTO_INCREMENT, PRIMARY KEY (id));
mysql> ALTER TABLE t AUTO_INCREMENT = 100;
mysql> INSERT INTO t (id) VALUES(NULL);
mysql> INSERT INTO t (id) VALUES(NULL);
mysql> INSERT INTO t (id) VALUES(NULL);
mysql> SELECT id FROM t ORDER BY id;
+-----+
| id  |
+-----+
| 100 |
| 101 |
| 102 |
+-----+
```

Чтобы установить начало последовательности в *n* для таблиц, тип которых отличается от MyISAM, требуется небольшая хитрость: нужно вставить «фальшивую» запись со значением последовательности *n* –1, а затем удалить ее после вставки одной или более «настоящих» записей. Следующий пример показывает, как начать последовательность со 100 в таблице InnoDB:

```
mysql> CREATE TABLE t
    -> (id INT UNSIGNED NOT NULL AUTO_INCREMENT, PRIMARY KEY (id))
    -> TYPE = InnoDB;
mysql> INSERT INTO t (id) VALUES(99);
mysql> INSERT INTO t (id) VALUES(NULL);
mysql> INSERT INTO t (id) VALUES(NULL);
mysql> INSERT INTO t (id) VALUES(NULL);
mysql> DELETE FROM t WHERE id = 99;
mysql> SELECT * FROM t ORDER BY id;
+-----+
| id  |
+-----+
| 100 |
| 101 |
| 102 |
+-----+
```

Помните, что если вы удаляете содержимое таблицы при помощи предложения DELETE без инструкции WHERE, начальное значение последовательности может быть установлено в 1 даже для тех типов таблиц, которые обычно не используют значения последовательности повторно (см. рецепт 11.4). В этом случае, если вы не хотите, чтобы последовательность начиналась с 1, следует явно установить начальное значение последовательности после очистки таблицы.

11.12. Добавление последовательности в существующую таблицу

Задача

Вы забыли включить столбец AUTO_INCREMENT в таблицу при создании. Теперь уже поздно?

Решение

Нет, просто добавьте его с помощью предложения ALTER TABLE. MySQL автоматически создаст столбец и пронумерует строки.

Обсуждение

Чтобы добавить последовательность в таблицу, где ее раньше не было, выполните предложение ALTER TABLE для создания столбца AUTO_INCREMENT. Предположим, что у вас есть таблица t, включающая столбцы name и age и не содержащая столбца последовательности:

```
+----------+------+
| name     | age  |
+----------+------+
| boris    |   47 |
| clarence |   62 |
| abner    |   53 |
+----------+------+
```

Вы можете добавить в таблицу столбец последовательности id следующим образом:

```
mysql> ALTER TABLE t
    -> ADD id INT NOT NULL AUTO_INCREMENT,
    -> ADD PRIMARY KEY (id);
mysql> SELECT * FROM t ORDER BY id;
+----------+------+----+
| name     | age  | id |
+----------+------+----+
| boris    |   47 |  1 |
| clarence |   62 |  2 |
| abner    |   53 |  3 |
+----------+------+----+
```

MySQL автоматически пронумерует строки, вам не нужно присваивать значения самостоятельно – очень удобно.

По умолчанию ALTER TABLE добавляет новые столбцы в конец таблицы. Чтобы поместить столбец в какое-то определенное место, используйте ключевое слово FIRST или AFTER в конце инструкции ADD. Следующие предложения ALTER TABLE аналогичны приведенному выше, но помещают столбец id первым в таблице или после столбца name соответственно:

```
ALTER TABLE t
ADD id INT NOT NULL AUTO_INCREMENT FIRST,
ADD PRIMARY KEY (id);

ALTER TABLE t
ADD id INT NOT NULL AUTO_INCREMENT AFTER name,
ADD PRIMARY KEY (id);
```

В таблицах MyISAM можно указать начальное значение для нового столбца последовательности, добавив инструкцию AUTO_INCREMENT = *n* в предложение ALTER TABLE:

```
mysql> ALTER TABLE t
    -> ADD id INT NOT NULL AUTO_INCREMENT FIRST,
    -> ADD PRIMARY KEY (id),
    -> AUTO_INCREMENT = 100;
mysql> SELECT * FROM t ORDER BY id;
+-----+----------+------+
| id  | name     | age  |
+-----+----------+------+
| 100 | boris    |   47 |
| 101 | clarence |   62 |
| 102 | abner    |   53 |
+-----+----------+------+
```

11.13. Создание последовательностей с помощью столбца AUTO_INCREMENT

Задача

Вас интересует более тонкая работа с последовательностями, чем просто последовательная нумерация строк. Необходимо связать различные последовательности со значениями других столбцов таблицы.

Решение

Свяжите столбец AUTO_INCREMENT с нужными столбцами, сделав все их составляющими одного индекса.

Обсуждение

Если столбец AUTO_INCREMENT является единственным столбцом индекса PRIMARY KEY или UNIQUE, он формирует одну последовательность 1, 2, 3,..., в которой последовательные значения увеличиваются на единицу при каждом добавлении записи вне зависимости от ее содержимого. Начиная с версии MySQL 3.23.5 для таблиц MyISAM можно создавать индекс, объединяющий столбец AUTO_INCREMENT с другими столбцами таблицы для генерирования нескольких последовательностей внутри одной таблицы.

Вот как это происходит: предположим, что Джуниор настолько увлекся проектом, что продолжает заниматься коллекцией жуков даже после выполне-

ния школьного задания. Только теперь, когда он не связан школьными правилами, он включает в свою коллекцию насекомообразных, вроде многоножки, и собирает сколь угодно много экземпляров каждого экспоната. За последние дни Джуниору удалось собрать несколько новых образцов:

Name (Название)	Date (Дата)	Origin (Источник)
ant	2001-10-07	kitchen
millipede	2001-10-07	basement (подвал)
beetle (жук)	2001-10-07	basement
ant	2001-10-07	front yard
ant	2001-10-07	front yard
honeybee (пчела)	2001-10-08	back yard
cricket	2001-10-08	garage (гараж)
beetle	2001-10-08	front yard
termite (термит)	2001-10-09	kitchen woodwork (дверь кухни)
cricket	2001-10-11	basement
termite	2001-10-11	bathroom woodwork (дверь ванной)
honeybee	2001-10-11	garden
cricket	2001-10-11	garden
ant	2001-10-11	garden

Сделав заметки, он собирается внести их в базу данных, но при этом хочет, чтобы каждый вид жучков был пронумерован отдельно (ant 1, ant 2, ..., beetle 1, beetle 2, ..., cricket 1, cricket 2 и т. д.). Вы просматриваете данные (отмечая с некоторой тревогой, что Джуниор обнаружил в доме термитов и что нужно не забыть вызвать специалиста по их уничтожению), затем создаете для Джуниора такую таблицу `bug`:

```
CREATE TABLE bug
(
    id      INT UNSIGNED NOT NULL AUTO_INCREMENT,
    name    VARCHAR(30) NOT NULL,   # тип жучка
    PRIMARY KEY (name, id),
    date    DATE NOT NULL,          # когда пойман
    origin  VARCHAR(30) NOT NULL    # где пойман
);
```

Она очень похожа на таблицу `insect`, но имеет одно важное отличие: индекс `PRIMARY KEY` содержит два столбца, а не один. Поэтому столбец `id` будет вести себя не так, как в таблице `insect`. Если в таблицу `bug` вводится новый набор экспонатов в том порядке, в котором их записал Джуниор, то в результате таблица будет выглядеть так:

```
mysql> SELECT * FROM bug;
```

```
+----+-----------+------------+-------------------+
| id | name      | date       | origin            |
+----+-----------+------------+-------------------+
|  1 | ant       | 2001-10-07 | kitchen           |
|  1 | millipede | 2001-10-07 | basement          |
|  1 | beetle    | 2001-10-07 | basement          |
|  2 | ant       | 2001-10-07 | front yard        |
|  3 | ant       | 2001-10-07 | front yard        |
|  1 | honeybee  | 2001-10-08 | back yard         |
|  1 | cricket   | 2001-10-08 | garage            |
|  2 | beetle    | 2001-10-08 | front yard        |
|  1 | termite   | 2001-10-09 | kitchen woodwork  |
|  2 | cricket   | 2001-10-10 | basement          |
|  2 | termite   | 2001-10-11 | bathroom woodwork |
|  2 | honeybee  | 2001-10-11 | garden            |
|  3 | cricket   | 2001-10-11 | garden            |
|  4 | ant       | 2001-10-11 | garden            |
+----+-----------+------------+-------------------+
```

На первый взгляд может показаться, что значения `id` присваивались произвольно, но это не так. Отсортируйте таблицу по `name` и `id`, и станет яснее, как MySQL присваивает значения. А именно, MySQL создает отдельную последовательность `id` для всех различных значений `name`:

```
mysql> SELECT * FROM bug ORDER BY name, id;
+----+-----------+------------+-------------------+
| id | name      | date       | origin            |
+----+-----------+------------+-------------------+
|  1 | ant       | 2001-10-07 | kitchen           |
|  2 | ant       | 2001-10-07 | front yard        |
|  3 | ant       | 2001-10-07 | front yard        |
|  4 | ant       | 2001-10-11 | garden            |
|  1 | beetle    | 2001-10-07 | basement          |
|  2 | beetle    | 2001-10-08 | front yard        |
|  1 | cricket   | 2001-10-08 | garage            |
|  2 | cricket   | 2001-10-10 | basement          |
|  3 | cricket   | 2001-10-11 | garden            |
|  1 | honeybee  | 2001-10-08 | back yard         |
|  2 | honeybee  | 2001-10-11 | garden            |
|  1 | millipede | 2001-10-07 | basement          |
|  1 | termite   | 2001-10-09 | kitchen woodwork  |
|  2 | termite   | 2001-10-11 | bathroom woodwork |
+----+-----------+------------+-------------------+
```

При создании многостолбцового индекса, включающего столбец типа `AUTO_INCREMENT`, необходимо учитывать следующее:

- Порядок, в котором индексированные столбцы определены в предложении `CREATE TABLE`, не имеет значения. Важен лишь порядок, в котором имена столбцов указаны в определении индекса. Столбец `AUTO_INCREMENT` должен быть указан последним, в противном случае механизм множественных последовательностей не будет работать.

- Индекс PRIMARY KEY не может содержать значения NULL, а UNIQUE — может. Если какой-то из столбцов не-AUTO_INCREMENT допускает значения NULL, необходимо создать индекс UNIQUE, а не PRIMARY KEY.

В таблице bug индекс AUTO_INCREMENT содержит два столбца. Но количество столбцов можно увеличить, при этом принцип останется тем же: для *n*-столбцового индекса, последним в котором является столбец AUTO_INCREMENT, MySQL генерирует независимую последовательность для каждой уникальной комбинации значений столбцов не-AUTO_INCREMENT. Предположим, что вы собираете информацию для исследования по субъектам мужского и женского пола, проверяемым в экспериментальных и контрольных условиях. Чтобы присвоить разные наборы номеров последовательности каждому из полов в каждых условиях, создадим трехстолбцовый индекс AUTO_INCREMENT:

```
CREATE TABLE subj_list
(
    name        CHAR(40),   # имя испытуемого
    condition   ENUM('control','experimental') NOT NULL,
    sex         ENUM('M','F') NOT NULL,
    id          INT UNSIGNED NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (condition, sex, id)
);
```

MySQL-механизм многостолбцовых последовательностей удобнее в использовании, чем логически эквивалентные одностолбцовые значения. Вспомните, как в рецепте 6.12 мы использовали таблицу housewares, содержащую строки с идентификаторами изделий, состоящими из трех частей: трехсимвольной аббревиатуры категории, пятизначного серийного номера и двухсимвольного кода страны-изготовителя:

```
+------------+------------------+
| id         | description      |
+------------+------------------+
| DIN40672US | dining table     |
| KIT00372UK | garbage disposal |
| KIT01729JP | microwave oven   |
| BED00038SG | bedside lamp     |
| BTH00485US | shower stall     |
| BTH00415JP | lavatory         |
+------------+------------------+
```

Там таблица использовалась для иллюстрации разбиения значений id на составляющие и их отдельной сортировки при помощи LEFT(), MID() и RIGHT(). Инструкции ORDER BY получались ужасно некрасивыми, кроме того, один вопрос я даже не стал тогда поднимать: как генерировать серийные номера в середине значений?

Иногда подобный многокомпонентный столбец можно заменить отдельными столбцами, связанными индексом AUTO_INCREMENT. Например, в качестве второго способа обработки значений id предметов домашнего обихода можно предложить их представление в виде столбцов category, serial и country и их объединение в индекс PRIMARY KEY, где серийный номер был бы столбцом

AUTO_INCREMENT. Тогда серийные номера независимо увеличивались бы для каждой уникальной комбинации категории и страны. Для создания таблицы с нуля следовало бы написать такое предложение CREATE TABLE:

```
CREATE TABLE housewares
(
    category    VARCHAR(3) NOT NULL,
    serial      INT UNSIGNED NOT NULL AUTO_INCREMENT,
    country     VARCHAR(2) NOT NULL,
    description VARCHAR(255),
    PRIMARY KEY (category, country, serial)
);
```

Если же считать, что исходная таблица housewares уже создана в том виде, в котором она использовалась в главе 6, то можно изменить ее структуру прямо «на месте»:

```
mysql> ALTER TABLE housewares
    -> ADD category VARCHAR(3) NOT NULL FIRST,
    -> ADD serial INT UNSIGNED NOT NULL AUTO_INCREMENT AFTER category,
    -> ADD country VARCHAR(2) NOT NULL AFTER serial,
    -> ADD PRIMARY KEY (category, country, serial);
mysql> UPDATE housewares SET category = LEFT(id,3);
mysql> UPDATE housewares SET serial = MID(id,4,5);
mysql> UPDATE housewares SET country = RIGHT(id,2);
mysql> ALTER TABLE housewares DROP id;
mysql> SELECT * FROM housewares;
+----------+--------+---------+------------------+
| category | serial | country | description      |
+----------+--------+---------+------------------+
| DIN      |  40672 | US      | dining table     |
| KIT      |    372 | UK      | garbage disposal |
| KIT      |   1729 | JP      | microwave oven   |
| BED      |     38 | SG      | bedside lamp     |
| BTH      |    485 | US      | shower stall     |
| BTH      |    415 | JP      | lavatory         |
+----------+--------+---------+------------------+
```

Когда значения id разбиты на составляющие, становится проще задавать операции сортировки, так как можно указывать отдельные столбцы напрямую, а не выделять подстроки исходных значений столбца id. Сортировку можно сделать еще эффективнее, добавив дополнительные индексы для столбцов serial и country. Но один вопрос пока не решен: как вывести идентификатор каждого продукта в виде одной строки, а не трех отдельных значений? Используем CONCAT():

```
mysql> SELECT category, serial, country,
    -> CONCAT(category,LPAD(serial,5,'0'),country) AS id
    -> FROM housewares ORDER BY category, country, serial;
+----------+--------+---------+------------+
| category | serial | country | id         |
+----------+--------+---------+------------+
| BED      |     38 | SG      | BED00038SG |
```

```
| BTH      |    415 | JP      | BTH00415JP |
| BTH      |    485 | US      | BTH00485US |
| DIN      |  40672 | US      | DIN40672US |
| KIT      |   1729 | JP      | KIT01729JP |
| KIT      |    372 | UK      | KIT00372UK |
+----------+--------+---------+------------+
```

Можно даже избавиться от необходимости применения `LPAD()`, объявив `serial` как столбец, использующий заполнители-нули для обеспечения вывода пятизначных значений:

```
mysql> ALTER TABLE housewares
    -> MODIFY serial INT(5) UNSIGNED ZEROFILL NOT NULL AUTO_INCREMENT;
```

Тогда MySQL будет добавлять начальные нули автоматически, и выражение `CONCAT()` станет проще:

```
mysql> SELECT category, serial, country,
    -> CONCAT(category,serial,country) AS id
    -> FROM housewares ORDER BY category, country, serial;
+----------+--------+---------+------------+
| category | serial | country | id         |
+----------+--------+---------+------------+
| BED      |  00038 | SG      | BED00038SG |
| BTH      |  00415 | JP      | BTH00415JP |
| BTH      |  00485 | US      | BTH00485US |
| DIN      |  40672 | US      | DIN40672US |
| KIT      |  01729 | JP      | KIT01729JP |
| KIT      |  00372 | UK      | KIT00372UK |
+----------+--------+---------+------------+
```

Пример иллюстрирует важную идею: вы можете воспринимать значения каким-то определенным образом (например, представлять значения `id` в виде единой строки), но это совсем не значит, что в базе данных их нужно хранить именно так. Если работа с другим представлением (отдельные столбцы) может оказаться эффективнее или проще, используйте его, даже если придется переформатировать столбцы для отображения, чтобы вывести значения в том виде, которого ожидают пользователи.

11.14. Управление несколькими столбцами AUTO_INCREMENT одновременно

Задача

Вы работаете с двумя или более таблицами, содержащими столбцы `AUTO_INCREMENT`, и вам трудно отслеживать значения последовательности, сформированные для каждой таблицы.

Решение

Сохраните значения в переменных SQL на будущее. Если вы используете запросы внутри программы, сохраните значения последовательности в пере-

менных программы. А может быть, вам удастся создавать запросы, используя разные объекты соединения или предложения, чтобы не смешивать их.

Обсуждение

Как говорилось в рецепте 11.5, функция-индикатор значения последовательности на стороне сервера LAST_INSERT_ID() устанавливается при каждом генерировании запросом значения AUTO_INCREMENT, в то время как клиентские индикаторы могут восстанавливаться после каждого запроса. Предположим, что вы хотите запустить предложение, генерирующее значение AUTO_INCREMENT, но использовать это значение только после того, как запущено второе предложение, тоже генерирующее значение AUTO_INCREMENT. Тогда первое значение уже не будет доступно ни как значение LAST_INSERT_ID(), ни как какое-то клиентское значение. Чтобы вновь получить доступ к исходному значению, необходимо сохранить его перед выполнением второго предложения. Есть несколько способов сделать это:

- На уровне SQL можно сохранить значение в переменной SQL после запуска запроса, генерирующего значение AUTO_INCREMENT:

```
INSERT INTO имя_таблицы (id,...) VALUES(NULL,...);
SET @saved_id = LAST_INSERT_ID();
```

  Тогда можно выполнять другие предложения, не заботясь об их воздействии на LAST_INSERT_ID(). Чтобы использовать в последующем запросе исходное значение AUTO_INCREMENT, обратитесь к переменной @saved_id.

- На уровне API можно хранить значение AUTO_INCREMENT в переменной базового языка. Для этого можно сохранить значение, возвращаемое LAST_INSERT_ID() или любым доступным специальным расширением API.

- Третий способ можно использовать в тех API, которые обеспечивают поддержку независимых клиентских значений AUTO_INCREMENT. Например, в Python при использовании объекта курсора для выполнения запроса для доступа к значению AUTO_INCREMENT, генерируемому запросом, вызовите метод курсора insert_id(). Если вы создаете другие запросы, используя тот же курсор, то значение будет потеряно. Однако если использовать другой объект курсора для выполнения дополнительных запросов, то исходное значение insert_id останется неизменным:

```
cursor1 = conn.cursor ()
cursor2 = conn.cursor ()
gen_seq_val (cursor1)       # создать запрос, формирующий
                            # номер последовательности
gen_seq_val (cursor2)       # создать еще запрос, используя другой курсор
seq1 = cursor1.insert_id ()
seq2 = cursor2.insert_id ()
print "seq1:", seq1, "seq2:", seq2  # эти значения будут различными
cursor1.close ()
cursor2.close ()
```

  В Perl тот же эффект можно получить, используя два дескриптора предложений; атрибут mysql_insertid каждого из них не изменяется при вы-

полнении запросов для другого. В Java используйте разные объекты `Statement` или `PreparedStatement`.

Третий способ не работает в PHP, так как в нем нет клиентского объекта или структуры, которые поддерживали бы значение `AUTO_INCREMENT` на уровне запросов. Клиентское значение `AUTO_INCREMENT` возвращается функцией `mysql_insert_id()`, то есть оно связано с соединением, а не с предложением. Да, я знаю, о чем вы подумали: можно было бы открыть второе соединение с сервером и запустить первый и второй запросы в рамках разных соединений. Вы правы, это сработало бы, но в данном случае цель не оправдывает приложенных усилий. Затраты на открытие второго соединения гораздо выше, чем на простое сохранение значения `mysql_insert_id()` в переменной PHP перед запуском второго запроса. Более того, открыть второе соединение не так просто, как кажется. Если выполнить второй вызов `mysql_connect()` или `mysql_pconnect()` с теми же параметрами соединения, что и первоначальный вызов, то PHP вернет тот же идентификатор соединения, что и в первый раз! Для того чтобы получить действительно другой идентификатор соединения, вам нужно будет подключиться к серверу под другим именем пользователя. (Рискуя замутить воду, все же отмечу, что начиная с PHP 4.2.0 `mysql_connect()` поддерживает опцию явного указания на необходимость открытия нового соединения. Вы можете использовать эту возможность для хранения разных клиентских значений `AUTO_INCREMENT`.)

11.15. Использование значений AUTO_INCREMENT для связывания таблиц

Задача

Вы используете значения последовательности одной таблицы как ключи для второй таблицы, чтобы соответствующим образом связать записи двух таблиц. Но что-то не получается.

Решение

Вероятно, вы вставляете записи не в том порядке или же не удается отслеживать значения последовательности. Измените порядок вставки строк или храните значения последовательности так, чтобы на них можно было сослаться в случае необходимости.

Обсуждение

Будьте внимательны при использовании значений `AUTO_INCREMENT` для идентификации строк главной таблицы, если вы храните эти же значения в записях подчиненной таблицы для сопоставления соответствующей записи главной таблицы. Такие ситуации весьма распространены. Предположим, что у вас есть таблица `invoice` с информацией о счетах по заказам пользователей и таблица `inv_item`, перечисляющая позиции для каждого счета. Здесь `invoice` является главной таблицей, а `inv_item` – подчиненной. Для однозначной

идентификации каждого заказа таблица invoice могла бы содержать столбец AUTO_INCREMENT с именем inv_id. Также можно было бы хранить соответствующий номер счета в каждой записи таблицы inv_item, чтобы определить, к какому счету она относится. Таблицы могут выглядеть как-то так:

```
CREATE TABLE invoice
(
    inv_id  INT UNSIGNED NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (inv_id),
    date    DATE NOT NULL
    # ... здесь могут быть другие столбцы
    # ... (идентификатор клиента, адрес доставки и т. д.)
);
CREATE TABLE inv_item
(
    inv_id      INT UNSIGNED NOT NULL,  # идентификатор счета (из таблицы invoice)
    INDEX (inv_id),
    qty         INT,                    # количество
    description VARCHAR(40)             # описание
);
```

При подобном связывании таблиц обычно сначала вставляется запись в главную (master) таблицу (для формирования значения AUTO_INCREMENT, идентифицирующего запись), затем вставляется запись подчиненной (detail) таблицы, получающая идентификатор записи главной таблицы от функции LAST_INSERT_ID(). Например, если клиент покупает молоток (hammer), три коробки гвоздей (nails) и (предвидя разбитые пальцы) дюжину бинтов (bandage), то относящиеся к заказу записи могут быть вставлены в две таблицы следующим образом:

```
INSERT INTO invoice (inv_id,date)
    VALUES(NULL,CURDATE());
INSERT INTO inv_item (inv_id,qty,description)
    VALUES(LAST_INSERT_ID(),1,'hammer');
INSERT INTO inv_item (inv_id,qty,description)
    VALUES(LAST_INSERT_ID(),3,'nails, box');
INSERT INTO inv_item (inv_id,qty,description)
    VALUES(LAST_INSERT_ID(),12,'bandage');
```

Первое предложение INSERT добавляет запись в главную таблицу invoice и генерирует новое значение AUTO_INCREMENT для ее столбца inv_id. Каждое из последующих предложений INSERT добавляет запись в подчиненную таблицу inv_item, используя LAST_INSERT_ID() для получения номера счета. Так записи подчиненной таблицы связываются с соответствующей главной записью.

Но что делать, если вам нужно обработать несколько счетов? Есть правильный и неправильный способы ввода информации. Правильный способ состоит в том, чтобы вставить все сведения о первом счете, затем перейти к следующему. Неправильный – добавить все главные записи в таблицу invoice, затем добавить все подчиненные записи в таблицу inv_item. Если выбрать второй способ, то *все* вставленные подчиненные записи таблицы inv_item будут содержать значение AUTO_INCREMENT последней записи из вставленных в таблицу

invoice. То есть будет казаться, что все входит в один счет, и записи двух таблиц не будут корректно сопоставлены друг другу.

Если подчиненная таблица содержит собственный столбец AUTO_INCREMENT, то добавлять записи в таблицы нужно еще осторожнее. Предположим, что требуется последовательно пронумеровать строки таблицы inv_item для каждого заказа. Для этого создаем многостолбцовый индекс AUTO_INCREMENT, генерирующий отдельную последовательность для составляющих каждого счета (о таких индексах рассказано в рецепте 11.13). Создаем таблицу inv_item, используя PRIMARY KEY, объединяющий столбец inv_id со столбцом AUTO_INCREMENT, который называется seq:

```
CREATE TABLE inv_item
(
    inv_id      INT UNSIGNED NOT NULL,  # идентификатор счета (из таблицы invoice)
    seq         INT UNSIGNED NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (inv_id, seq),
    qty         INT,                    # количество
    description VARCHAR(40)             # описание
);
```

Столбец inv_id позволяет сопоставить каждую строку inv_item соответствующей записи таблицы invoice, как и в исходной таблице. Дополнительно индекс обеспечивает последовательную нумерацию, начиная с 1, значений seq для элементов каждого счета. Однако теперь, когда обе таблицы содержат столбец AUTO_INCREMENT, вводить информацию о счете так же, как раньше, уже нельзя. Чтобы понять, почему, попробуем сделать следующее:

```
INSERT INTO invoice (inv_id,date)
    VALUES(NULL,CURDATE());
INSERT INTO inv_item (inv_id,qty,description)
    VALUES(LAST_INSERT_ID(),1,'hammer');
INSERT INTO inv_item (inv_id,qty,description)
    VALUES(LAST_INSERT_ID(),3,'nails, box');
INSERT INTO inv_item (inv_id,qty,description)
    VALUES(LAST_INSERT_ID(),12,'bandage');
```

Это те же самые запросы, но ведут они себя несколько иначе из-за изменений в структуре таблицы inv_item. Предложение INSERT для таблицы invoice выполняется корректно. Выполняется и первое предложение INSERT для таблицы inv_item; LAST_INSERT_ID() возвращает значение inv_id главной записи из таблицы invoice. Но это предложение INSERT генерирует и собственное значение AUTO_INCREMENT (для столбца seq), которое изменяет значение LAST_INSERT_ID(), и значение inv_id главной записи «пропадает». В результате последующие вставки в таблицу inv_item сохраняют значение seq предыдущей записи в столбце inv_id. Поэтому вторая и третья записи содержат некорректные значения inv_id.

Есть несколько путей решения проблемы. Один из них связан с использованием другого синтаксиса предложения INSERT для добавления подчиненных записей, другие предлагают сохранять значение AUTO_INCREMENT главной записи в переменной для дальнейшего использования:

Одновременное добавление нескольких подчиненных записей. Одним из решений является добавление подчиненных записей с использованием синтаксиса `INSERT` MySQL, который позволяет добавить несколько строк в одном предложении. Тогда можно будет применить значение `LAST_INSERT_ID()` главной записи ко всем подчиненным:

```
INSERT INTO invoice (inv_id,date)
    VALUES(NULL,CURDATE());
INSERT INTO inv_item (inv_id,qty,description) VALUES
    (LAST_INSERT_ID(),1,'hammer'),
    (LAST_INSERT_ID(),3,'nails, box'),
    (LAST_INSERT_ID(),12,'bandage');
```

Использование переменной SQL. Второй способ заключается в сохранении значения `AUTO_INCREMENT` главной записи в переменной SQL для использования при последующей вставке подчиненных записей:

```
INSERT INTO invoice (inv_id,date)
    VALUES(NULL,CURDATE());
SET @inv_id = LAST_INSERT_ID();
INSERT INTO inv_item (inv_id,qty,description)
    VALUES(@inv_id,1,'hammer');
INSERT INTO inv_item (inv_id,qty,description)
    VALUES(@inv_id,3,'nails, box');
INSERT INTO inv_item (inv_id,qty,description)
    VALUES(@inv_id,12,'bandage');
```

Использование переменной API. Третий способ похож на второй, но работает только в рамках API. Вставьте главную запись, затем сохраните значение `AUTO_INCREMENT` в переменной API для использования при последующей вставке подчиненных записей. Например, в Perl вы можете получить доступ к `AUTO_INCREMENT` через атрибут `mysql_insertid`, поэтому процедура ввода счета будет такой:

```
$dbh->do ("INSERT INTO invoice (inv_id,date) VALUES(NULL,CURDATE())");
$inv_id = $dbh->{mysql_insertid};
$sth = $dbh->prepare ("INSERT INTO inv_item (inv_id,qty,description)
                    VALUES(?,?,?)");
$sth->execute ($inv_id, 1, "hammer");
$sth->execute ($inv_id, 3, "nails, box");
$sth->execute ($inv_id, 12, "bandage");
```

11.16. Генераторы однострочных последовательностей

Задача

Вас интересует только подсчет количества событий, так что нет смысла создавать запись для каждого отсчета.

Решение

Примените другой механизм формирования последовательности, который использует всего одну строку.

Обсуждение

Столбцы AUTO_INCREMENT полезны для генерирования последовательностей для множества отдельных записей. Но в некоторых приложениях требуется лишь счетчик количества произошедших событий, и не стоит создавать отдельную запись для каждого события. В качестве примеров можно привести счетчики посещаемости веб-страницы и нажатий на баннер, учет проданных изделий или подсчет результатов голосования. В таких приложениях вам нужна всего одна запись для хранения изменяющегося с течением времени счетчика. MySQL поддерживает соответствующий механизм, позволяющий представлять счетчики значениями типа AUTO_INCREMENT, так что вы можете не только увеличивать счетчик, но и без труда извлекать обновленное значение.

Чтобы подсчитывать события одного типа, можно использовать тривиальную таблицу, состоящую из одной строки и одного столбца. Например, если вы продаете книгу под названием «Red Horse Hill», то можете создать и инициализировать таблицу для записи результатов продаж следующим образом:

```
CREATE TABLE red_horse_hill (copies INT UNSIGNED);
INSERT INTO red_horse_hill (copies) VALUES(0);
```

Однако если вы продаете разные книги, этот способ уже не так хорош. Вы, конечно, не захотите создавать отдельную однострочную таблицу для подсчета продаж каждой книги. Все можно сделать в одной таблице, если включить в нее столбец, содержащий уникальный идентификатор для каждой книги. Таблица booksales выполняет нашу задачу, используя столбец title для названия книги в дополнение к столбцу copies, в котором ведется подсчет проданных экземпляров:

```
CREATE TABLE booksales
(
    title   VARCHAR(60) NOT NULL,   # название книги
    copies  INT UNSIGNED NOT NULL,  # продано экземпляров
    PRIMARY KEY (title)
);
```

Инициализируем таблицу, добавляя строку для каждой книги:

```
mysql> INSERT INTO booksales (title) VALUES
    -> ('Red Horse Hill'),
    -> ('Sparkplug of the Hornets'),
    -> ('Bulldozer'),
    -> ('The Long Trains Roll'),
    -> ('Who Rides in the Dark?');
mysql> SELECT * FROM booksales;
+-------------------------+--------+
| title                   | copies |
```

```
+-------------------------+--------+
| The Long Trains Roll    |      0 |
| Bulldozer               |      0 |
| Sparkplug of the Hornets |      0 |
| Red Horse Hill          |      0 |
| Who Rides in the Dark?  |      0 |
+-------------------------+--------+
```

Таблица создана. Как ее использовать? Можно увеличивать столбец `copies` для указанной книги, создавая простое предложение `UPDATE` с названием книги:

```
UPDATE booksales SET copies = copies+1 WHERE title = 'Bulldozer';
```

Для извлечения счетчика (чтобы можно было, например, вывести для клиента сообщение типа «вы купили *n*-ый экземпляр книги») создайте запрос `SELECT` для книги с этим названием:

```
SELECT copies FROM booksales WHERE title = 'Bulldozer';
```

К сожалению, такой способ на самом деле не очень хорошо работает. Предположим, что в промежуток времени между предложениями `UPDATE` и `SELECT` кто-то другой покупает экземпляр книги, тем самым увеличивая значение `copies`. Тогда предложение `SELECT` фактически выведет не то значение счетчика продаж, которое получено путем прибавления вашего экземпляра, а самое последнее значение счетчика. Иначе говоря, другие клиенты могут изменить значение прежде, чем вы успеете его извлечь. Проблема похожа на обсуждавшуюся ранее и возникающую при извлечении последнего значения `AUTO_INCREMENT` из столбца при помощи вызова `MAX(`*`имя_столбца`*`)`, а не `LAST_INSERT_ID()`.

Есть варианты ухода от этой проблемы (группировка предложений в транзакцию или блокировка таблицы), но MySQL предлагает и свое решение, основанное на применении `LAST_INSERT_ID()`. Если вызвать функцию `LAST_INSERT_ID()` с аргументом-выражением, MySQL воспримет его как значение `AUTO_INCREMENT`.[1] Для того чтобы использовать такую возможность увеличения счетчиков в таблице `booksales`, надо немного изменить предложение `UPDATE`:

```
UPDATE booksales SET copies = LAST_INSERT_ID(copies+1)
WHERE title = 'Bulldozer';
```

Затем можно вызвать `LAST_INSERT_ID()` без аргументов для извлечения значения:

```
SELECT LAST_INSERT_ID();
```

Обновляя таким образом столбец `copies`, вы всегда можете получить присвоенное значение, даже если какое-то другое клиентское приложение его тем временем изменило. Если вы запускаете предложение `UPDATE` из API, обеспечивающих непосредственное извлечение последнего значения `AUTO_INCREMENT`, то можно даже не создавать запрос `SELECT`. Например, в Python вы можете обновить счетчик и получить новое значение, используя метод `insert_id()`:

[1] Механизм `LAST_INSERT_ID(`*`выражение`*`)` появился в MySQL версии 3.22.9.

```
cursor = conn.cursor ()
cursor.execute ("""
        UPDATE booksales SET copies = LAST_INSERT_ID(copies+1)
        WHERE title = 'Bulldozer'
    """)
count = cursor.insert_id ()
```

В Java эта операция выглядит так:

```
Statement s = conn.createStatement ();
s.executeUpdate (
    "UPDATE booksales SET copies = LAST_INSERT_ID(copies+1)"
    + " WHERE title = 'Bulldozer'");
long count = ((com.mysql.jdbc.Statement) s).getLastInsertID ();
s.close ();
```

Последовательность, сформированная с использованием LAST_INSERT_ID(), обладает свойствами, отличными от настоящих последовательностей AUTO_INCREMENT:

- Значения AUTO_INCREMENT каждый раз увеличиваются на единицу, в то время как значения счетчика, генерируемые LAST_INSERT_ID(*выражение*) могут увеличиваться на любое значение. Например, чтобы сформировать последовательность 10, 20, 30,..., каждый раз увеличивайте счетчик на 10. Необязательно даже каждый раз увеличивать счетчик на одно и то же значение. Если вы продали двенадцать экземпляров книги, а не один, измените значение счетчика ее продаж так:

  ```
  UPDATE booksales SET copies = LAST_INSERT_ID(copies+12)
  WHERE title = 'Bulldozer';
  ```

- Начать последовательность можно с любого целого, в том числе с отрицательного значения. Можно формировать убывающие последовательности, используя отрицательное приращение (для столбца, применяемого для формирования последовательности, которая включает отрицательные значения, следует убрать UNSIGNED из определения столбца).
- Чтобы вернуть счетчик в исходное значение, просто назначьте ему нужное значение. Предположим, что вы хотите сообщить покупателям книг о продажах текущего месяца, а не об общем объеме продаж (например, выводя сообщение типа «вы являетесь *n*-ым покупателем месяца»). Для обнуления счетчика в начале каждого месяца выполните такой запрос:

  ```
  UPDATE booksales SET copies = 0;
  ```

- Одно не столь приятное свойство заключается в том, что значение, формируемое вызовом LAST_INSERT_ID(*выражение*), не всегда доступно для извлечения клиентскими приложениями. Его можно получить после запросов UPDATE и INSERT, но не для предложений SET. Если генерировать значение следующим образом (в Perl), то клиентское значение, возвращенное mysql_insertid, будет равно 0, а не 48:

  ```
  $dbh->do ("SET \@x = LAST_INSERT_ID(48)");
  $seq = $dbh->{mysql_insertid};
  ```

Чтобы в этом случае получить значение, необходимо запросить его у сервера:

```
$seq = $dbh->selectrow_array ("SELECT LAST_INSERT_ID()");
```

Мы вернемся к механизму формирования однострочной последовательности в рецепте 18.12, где он будет использоваться в качестве основы для реализации счетчиков посещаемости веб-страницы.

11.17. Формирование повторяющихся последовательностей

Задача

Вам нужно создать последовательность, которая содержит циклы.

Решение

Сформируйте последовательность и получите циклические элементы, используя операторы деления и деления по модулю.

Обсуждение

В некоторых задачах формирования последовательностей требуются значения, повторяющиеся циклически. Предположим, что вы выпускаете, например, фармацевтические товары или детали для автомобилей, и вам необходимо отслеживать их по номеру партии на тот случай, если будут выявлены проблемы производства, требующие отзыва изделий этой партии. Предположим также, что вы упаковываете по 12 изделий (unit) в коробку (box) и по 6 коробок в ящик (case). В этом случае идентификатор изделия состоит из трех частей: номер изделия (от 1 до 12), номер коробки (от 1 до 6) и номер партии (со значением от 1 до наибольшего текущего номера ящика).

Задача отслеживания изделий, похоже, требует хранения трех счетчиков, так что можно предложить такой алгоритм формирования следующего идентификатора:

```
получить последние использованные номера партии, коробки и изделия
unit = unit + 1     # увеличить номер изделия
if (unit > 12)      # начать новую коробку?
{
    unit = 1        # перейти к первому изделию следующей коробки
    box = box + 1
}
if (box > 6)        # начать новый ящик?
{
    box = 1         # перейти к первой коробке следующего ящика
    case = case + 1
}
сохранить полученные номера партии, коробки и изделия
```

Действительно, можно действовать по этому алгоритму. А можно просто присваивать каждому элементу идентификатор последовательности (seq) и извлекать из него соответствующие номера ящика, коробки и изделия. Идентификатор можно получать из столбца AUTO_INCREMENT или от генератора однострочной последовательности. Формулы определения номера ящика, номера коробки и номера изделия для любого изделия по его номеру последовательности таковы:

```
unit = ((seq - 1) % 12) + 1
box = (int ((seq - 1) / 12) % 6) + 1
case = int ((seq - 1)/(6 * 12)) + 1
```

Приведенная ниже таблица иллюстрирует связь между номерами последовательности и соответствующими номерами ящика, коробки и изделия:

```
seq  case   box  unit
  1     1     1     1
 12     1     1    12
 13     1     2     1
 72     1     6    12
 73     2     1     1
144     2     6    12
```

11.18. Последовательная нумерация строк вывода запроса

Задача

Вы хотите пронумеровать выводимые запросом строки.

Решение

Если вы пишете собственную программу, то просто добавьте номера строк самостоятельно.

Обсуждение

Последовательность, никак не связанная с содержимым вашей базы данных, используется для нумерации строк, выводимых запросом. Если вы работаете в API, то можете нумеровать строки, храня счетчик и выводя его текущее значение вместе с содержимым каждой строки. Рассмотрим пример на Python для таблицы insects: здесь просто выводится нумерованный список различных значений столбца origin таблицы:

```
cursor = conn.cursor ()
cursor.execute ("SELECT DISTINCT origin FROM insect")
count = 1
while 1:
    row = cursor.fetchone ()
    if row == None:
```

```
            break
        print count, row[0]
        count = count + 1
    cursor.close ()
```

См. также

Программа *mysql* не поддерживает возможностей явной нумерации строк, хотя вы можете использовать переменную SQL для добавления в вывод запроса дополнительного столбца. Есть и другой, может быть, более устраивающий вас способ, – пропустить вывод *mysql* через другую программу, добавляющую номера строк. Эти приемы описаны в рецепте 1.26.

12

Использование нескольких таблиц

12.0. Введение

Рассмотренные ранее рецепты по большей части использовали одну таблицу. Но даже несложное приложение почти наверняка работает с несколькими таблицами. На некоторые запросы просто невозможно ответить, используя всего одну таблицу, к тому же реальная мощь реляционных баз данных проявляется тогда, когда вы начинаете связывать данные из одних таблиц с другими. Существует ряд причин использования нескольких таблиц:

- Для соединения записей таблиц с целью получения более полной информации, чем та, которую можно извлечь из каждой отдельной таблицы.
- Для хранения промежуточных результатов многоэтапной операции.
- Для вставки, удаления или обновления записей одной таблицы на основе информации из другой.

Когда вы работаете с несколькими таблицами, они могут находиться как в одной и той же, так и в разных базах данных. Может случиться даже так, что ваши таблицы будут принадлежать базам данных, расположенным на разных серверах MySQL. В двух первых случаях вам необходимо уметь ссылаться на столбцы разных таблиц, для чего можно использовать псевдонимы таблиц или указывать имя таблицы вместе с именем базы данных. В третьем случае необходимо открыть соединение с каждым из серверов и установить связь между их данными самостоятельно.

12.1. Соединение строк одной таблицы со строками другой

Задача

Вы хотите написать запрос, который использует информацию более чем из одной таблицы.

Решение

Используйте соединение (join) – запрос, ссылающийся на несколько таблиц и указывающий MySQL на то, как сопоставлять их данные.

Обсуждение

Основная идея соединения в том, что оно комбинирует строки таблицы со строками одной или нескольких других таблиц. Полное соединение таблиц выводит все возможные сочетания строк. Например, соединение 100-строчной таблицы с 200-строчной выводит результат, состоящий из 100×200, то есть 20 000 строк. Для больших таблиц и для соединений более чем двух таблиц результирующее множество может быть просто гигантским и может даже вызвать переполнение временного табличного пространства сервера MySQL. По этой причине, а также потому что абсолютно все комбинации требуются редко, соединение обычно содержит инструкцию `WHERE`, сужающую выборку запроса. В этом разделе представлены основы синтаксиса соединения, а в последующих разделах мы поговорим о том, как соединение может помочь получить ответы на определенные типы вопросов.

Предположим, что вы абсолютно лишены воображения при подборе одежды и каждый день испытываете проблемы с гардеробом. Тогда вы решаете обратиться за помощью к MySQL. Для начала помещаем все ваши рубашки (`shirt`) в одну таблицу, а галстуки (`tie`) – в другую:

```
mysql> CREATE TABLE shirt (item CHAR(20));
mysql> INSERT INTO shirt (item)
    -> VALUES('Pinstripe'),('Tie-Dye'),('Black');
mysql> CREATE TABLE tie (item CHAR(20));
mysql> INSERT INTO tie (item)
    -> VALUES('Fleur de lis'),('Paisley'),('Polka Dot');
```

Чтобы вывести содержимое каждой из таблиц, можно использовать отдельные однотабличные запросы:

```
mysql> SELECT item FROM shirt;
+-----------+
| item      |
+-----------+
| Pinstripe |
| Tie-Dye   |
| Black     |
+-----------+
mysql> SELECT item FROM tie;
+--------------+
| item         |
+--------------+
| Fleur de lis |
| Paisley      |
| Polka Dot    |
+--------------+
```

Можно попросить MySQL вывести различные комбинации предметов гардероба, написав запрос, выполняющий соединение. В соединении имена двух или более таблиц указываются после ключевого слова FROM. В списке столбцов вывода могут присутствовать имена столбцов из любых или всех таблиц соединения или выражения на их основе. В простейшем соединении участвуют две таблицы, из которых выбираются все столбцы. Если инструкция WHERE не указана, то соединение генерирует вывод из всех комбинаций строк. То есть чтобы найти все возможные сочетания рубашек и галстуков, используйте запрос, выполняющий полное соединение двух таблиц:

```
mysql> SELECT * FROM shirt, tie;
+-----------+--------------+
| item      | item         |
+-----------+--------------+
| Pinstripe | Fleur de lis |
| Tie-Dye   | Fleur de lis |
| Black     | Fleur de lis |
| Pinstripe | Paisley      |
| Tie-Dye   | Paisley      |
| Black     | Paisley      |
| Pinstripe | Polka Dot    |
| Tie-Dye   | Polka Dot    |
| Black     | Polka Dot    |
+-----------+--------------+
```

Как видите, каждый элемент таблицы shirt образует пару с каждым элементом таблицы tie. Распечатайте список и повесьте на стену – теперь вам больше не надо мучиться по утрам. Каждый день одевайте то, что отображается в первой неиспользованной строке и вычеркивайте ее из списка.

Список столбцов вывода предыдущего запроса выглядит как *. Для однотабличного запроса список вывода * означает «все столбцы указанной таблицы». Аналогично для соединения это значит «все столбцы всех указанных таблиц», поэтому запрос возвращает столбцы из двух таблиц: shirt и tie. Можно использовать *имя_таблицы*.* для выбора всех столбцов определенной таблицы или *имя_таблицы*.*имя_столбца* для выбора одного столбца таблицы. То есть все приведенные ниже запросы эквиваленты:

```
SELECT * FROM shirt, tie;
SELECT shirt.*, tie.* FROM shirt, tie;
SELECT shirt.*, tie.item FROM shirt, tie;
SELECT shirt.item, tie.* FROM shirt, tie;
SELECT shirt.item, tie.item FROM shirt, tie;
```

Нотация *имя_таблицы*.*имя_столбца*, указывающая имя столбца вместе с именем таблицы, всегда разрешена, но ее можно сократить до *имя_столбца*, если это имя присутствует только в одной из таблиц соединения. В этом случае MySQL может однозначно определить, к какой таблице относится столбец, и не требуется уточнять имя таблицы. Но при соединении таблиц shirt и tie сокращенные имена использовать нельзя, так как обе таблицы содержат столбец item, и поэтому следующий запрос неоднозначен:

```
mysql> SELECT item, item FROM shirt, tie;
ERROR 1052 at line 1: Column: 'item' in field list is ambiguous
```

Если столбцы имеют различные имена, такие как `s_item` и `t_item`, то запрос будет недвусмысленным и без указания имен таблиц:

```
SELECT s_item, p_item FROM shirt, tie;
```

Чтобы сделать запрос более удобочитаемым, полезно указывать имена столбцов, даже если они не необходимы для MySQL. Именно поэтому я стараюсь использовать уточненные имена в примерах соединений.

Если инструкция `WHERE` не ограничивает вывод, то соединение формирует строку вывода для каждой возможной комбинации строк. Для больших таблиц это не очень разумная операция, поэтому на выводимые строки обычно налагаются какие-то условия. Например, если вам надоели поддразнивания коллег относительно вашего галстука в горошек, выберите только *отличные* от данного сочетания предметы вашего туалета:

```
mysql> SELECT shirt.item, tie.item FROM shirt, tie
    -> WHERE tie.item != 'Polka Dot';
+-----------+--------------+
| item      | item         |
+-----------+--------------+
| Pinstripe | Fleur de lis |
| Tie-Dye   | Fleur de lis |
| Black     | Fleur de lis |
| Pinstripe | Paisley      |
| Tie-Dye   | Paisley      |
| Black     | Paisley      |
+-----------+--------------+
```

Вывод можно ограничить и другими способами. Чтобы выбирать сочетания предметов случайным образом, запускайте каждое утро следующий запрос для выбора одной строки полного соединения:[1]

```
mysql> SELECT shirt.item, tie.item FROM shirt, tie
    -> ORDER BY RAND() LIMIT 1;
+---------+--------------+
| item    | item         |
+---------+--------------+
| Tie-Dye | Fleur de lis |
+---------+--------------+
```

Можно выполнять соединение более чем двух таблиц. Давайте создадим таблицу брюк `pants`:

```
mysql> SELECT * FROM pants;
+----------+
| item     |
+----------+
| Plaid    |
```

[1] Инструкция `ORDER BY RAND` рассматривается далее в главе 13.

```
| Striped  |
| Corduroy |
+----------+
```

Теперь можно выбирать комбинации рубашек, галстуков и брюк:

```
mysql> SELECT shirt.item, tie.item, pants.item FROM shirt, tie, pants;
+-----------+--------------+----------+
| item      | item         | item     |
+-----------+--------------+----------+
| Pinstripe | Fleur de lis | Plaid    |
| Tie-Dye   | Fleur de lis | Plaid    |
| Black     | Fleur de lis | Plaid    |
| Pinstripe | Paisley      | Plaid    |
| Tie-Dye   | Paisley      | Plaid    |
| Black     | Paisley      | Plaid    |
| Pinstripe | Polka Dot    | Plaid    |
| Tie-Dye   | Polka Dot    | Plaid    |
| Black     | Polka Dot    | Plaid    |
| Pinstripe | Fleur de lis | Striped  |
| Tie-Dye   | Fleur de lis | Striped  |
| Black     | Fleur de lis | Striped  |
| Pinstripe | Paisley      | Striped  |
| Tie-Dye   | Paisley      | Striped  |
| Black     | Paisley      | Striped  |
| Pinstripe | Polka Dot    | Striped  |
| Tie-Dye   | Polka Dot    | Striped  |
| Black     | Polka Dot    | Striped  |
| Pinstripe | Fleur de lis | Corduroy |
| Tie-Dye   | Fleur de lis | Corduroy |
| Black     | Fleur de lis | Corduroy |
| Pinstripe | Paisley      | Corduroy |
| Tie-Dye   | Paisley      | Corduroy |
| Black     | Paisley      | Corduroy |
| Pinstripe | Polka Dot    | Corduroy |
| Tie-Dye   | Polka Dot    | Corduroy |
| Black     | Polka Dot    | Corduroy |
+-----------+--------------+----------+
```

Естественно, чем больше таблиц вы объединяете, тем больше комбинаций строк, даже если каждая отдельная таблица не очень велика.

Если вы не хотите писать полные имена таблиц в списке столбцов вывода, сопоставьте каждой таблице короткий псевдоним и ссылайтесь на столбцы таблицы, используя псевдонимы:

```
SELECT s.item, t.item, p.item
FROM shirt AS s, tie AS t, pants AS p;
```

В предыдущем разделе псевдонимы не так уж сильно сократили объем ввода, но в сложных запросах, выбирающих множество столбцов, псевдонимы могут значительно упростить жизнь. Кроме того, псевдонимы не просто удобны, но и необходимы в некоторых видах запросов (вы поймете это, когда мы дойдем до рецепта 12.11, представляющего соединения таблицы с самой собой).

12.2. Соединение таблиц разных баз данных

Задача

Вы хотите использовать таблицы в соединении, но они расположены в разных базах данных.

Решение

Используйте спецификатор имени базы данных, чтобы сообщить MySQL, где следует искать таблицы.

Обсуждение

Бывают ситуации, когда необходимо выполнить соединение двух таблиц, расположенных в разных базах данных. Для этого укажите имена столбцов и таблиц достаточно полно для того, чтобы MySQL могла понять, на что именно вы ссылаетесь. Мы использовали таблицы `shirt` и `tie`, неявно подразумевая, что обе они находятся в базе данных `cookbook`, так что можно ссылаться на таблицы, не указывая имени базы данных. Например, следующий запрос извлекает сочетания элементов двух таблиц:

```
mysql> SELECT shirt.item, tie.item FROM shirt, tie;
+-----------+--------------+
| item      | item         |
+-----------+--------------+
| Pinstripe | Fleur de lis |
| Tie-Dye   | Fleur de lis |
| Black     | Fleur de lis |
| Pinstripe | Paisley      |
| Tie-Dye   | Paisley      |
| Black     | Paisley      |
| Pinstripe | Polka Dot    |
| Tie-Dye   | Polka Dot    |
| Black     | Polka Dot    |
+-----------+--------------+
```

Но предположим, что таблица `shirt` находится в базе данных `db1`, а `tie` – в базе данных `db2`. Чтобы сообщить об этом MySQL, перед именем каждой таблицы добавим префикс, определяющий базу данных, к которой относится таблица. Соединение с полной формой указания имен выглядит так:

```
SELECT db1.shirt.item, db2.tie.item FROM db1.shirt, db2.tie;
```

Если текущая база данных не выбрана, или это база данных, отличная от `db1` и `db2`, необходимо указывать имена в полной форме. Однако если `db1` или `db2` является текущей базой данных, то можно обойтись без некоторых спецификаторов. Например, если текущая база данных – `db1`, то можно опустить спецификаторы `db1`:

```
SELECT shirt.item, db2.tie.item FROM shirt, db2.tie;
```

И наоборот, если текущей базой данных является db2, то нет необходимости в префиксах db2:

```
SELECT db1.shirt.item, tie.item FROM db1.shirt, tie;
```

12.3. Ссылка на имена столбцов вывода соединения в программе

Задача

Вам необходимо обработать результирующее множество соединения в программе, но столбцы вывода имеют неуникальные имена.

Решение

Используйте псевдонимы столбцов для присваивания уникальных имен всем столбцам или ссылайтесь на столбцы позиционно.

Обсуждение

Соединения часто извлекают столбцы похожих таблиц, и нередко получается так, что выбранные из разных таблиц столбцы называются одинаково. Обратимся вновь к соединению таблиц shirt, tie и pants, использованных в рецепте 12.1:

```
mysql> SELECT shirt.item, tie.item, pants.item FROM shirt, tie, pants;
+-----------+--------------+----------+
| item      | item         | item     |
+-----------+--------------+----------+
| Pinstripe | Fleur de lis | Plaid    |
| Tie-Dye   | Fleur de lis | Plaid    |
| Black     | Fleur de lis | Plaid    |
| Pinstripe | Paisley      | Plaid    |
...
```

Запрос использует имена таблиц для уточнения принадлежности каждого экземпляра столбца item в списке вывода. Но имена столбцов вывода не отличаются друг от друга, так как MySQL не включает имена таблиц в заголовки столбцов. Если вы обрабатываете результат соединения в программе и извлекаете строки в структуру данных, которая ссылается на значения столбцов по имени, неуникальные имена столбцов могут привести к недоступности некоторых значений. Покажем возможные трудности с помощью фрагмента сценария на Perl:

```
$stmt = qq{
    SELECT shirt.item, tie.item, pants.item
    FROM shirt, tie, pants
};
$sth = $dbh->prepare ($stmt);
$sth->execute ();
# Определим количество столбцов результирующего множества двумя способами:
```

```
# - Проверим атрибут дескриптора предложения NUM_OF_FIELDS
# - Извлечем строку в хеш и посмотрим, сколько ключей он будет содержать
$count1 = $sth->{NUM_OF_FIELDS};
$ref = $sth->fetchrow_hashref ();
$count2 = keys (%{$ref});
print "The statement is: $stmt\n";
print "According to NUM_OF_FIELDS, the result set has $count1 columns\n";
print "The column names are: " . join (",", sort (@{$sth->{NAME}})) . "\n";
print "According to the row hash size, the result set has $count2 columns\n";
print "The column names are: " . join (",", sort (keys (%{$ref}))) . "\n";
```

Сценарий выдает запрос подбора гардероба и определяет количество столбцов результирующего множества, сначала проверяя атрибут NUM_OF_FIELDS, а затем извлекая строку в хеш и подсчитывая количество ключей хеша. В результате выполнения сценария формируется такой вывод:

```
According to NUM_OF_FIELDS, the result set has 3 columns
The column names are: item,item,item
According to the row hash size, the result set has 1 columns
The column names are: item
```

Что-то не так – счетчики столбцов не совпадают. Второй счетчик равен 1, так как неуникальные имена столбцов приводят к сопоставлению нескольких значений столбцов одному элементу хеша. В итоге некоторые значения оказываются утеряны. Для решения проблемы сделайте имена столбцов уникальными с помощью псевдонимов. Например, можно переписать запрос:

```
SELECT shirt.item, tie.item, pants.item
FROM shirt, tie, pants
```

так:

```
SELECT shirt.item AS shirt, tie.item AS tie, pants.item AS pants
FROM shirt, tie, pants
```

Если выполнить такое изменение и заново запустить сценарий, то вывод будет таким:

```
According to NUM_OF_FIELDS, the result set has 3 columns
The column names are: pants,shirt,tie
According to the row hash size, the result set has 3 columns
The column names are: pants,shirt,tie
```

Теперь счетчики столбцов одинаковы, при выборе в хеш значения не были утеряны.

Второй способ решения задачи, не требующий переименования столбцов, заключается в выборке строки не в хеш. Например, вы можете выбрать строку в массив и ссылаться на рубашку, галстук и брюки как на элементы массива с первого по третий:

```
while (my @val = $sth->fetchrow_array ())
{
    print "shirt: $val[0], tie: $val[1], pants: $val[2]\n";
}
```

В других языках проблему конфликтов имен можно решать другими способами. Например, в сценариях на Python дело обстоит несколько иначе, чем в Perl. Если вы извлекаете строку, используя словарь (ближайший аналог Python для хеша Perl), то модуль MySQLdb замечает совпадение имен столбцов и помещает их в словарь, используя ключ, состоящий из имени столбца, перед которым указано имя таблицы. То есть в приведенном ниже запросе ключами словаря будут `item`, `tie.item` и `pants.item`:

```
SELECT shirt.item, tie.item, pants.item
FROM shirt, tie, pants
```

Это означает, что значения столбцов не будут потеряны, но все же нельзя забывать о неуникальных именах. Если вы попытаетесь сослаться на значения столбцов, используя просто их имена, то не получите ожидаемых результатов для тех имен, в начало которых добавлены имена таблиц. Если использовать псевдонимы для обеспечения уникальности всех имен столбцов, то записи словаря получат именно те имена, которые вы им присвоите.

12.4. Нахождение строк одной таблицы, соответствующих строкам другой

Задача

Вы хотите использовать строки одной таблицы для того, чтобы определить строки другой таблицы.

Решение

Используйте соединение с инструкцией `WHERE` для установления соответствия между строками разных таблиц.

Обсуждение

Записи таблиц `shirt`, `tie` и `pants` из рецепта 12.1 никак не связаны друг с другом, поэтому ни одна из их комбинаций не является более осмысленной, чем какая-то другая. Это нормально, так как те примеры были предназначены для иллюстрации выполнения соединения, а не для ответа на вопрос, зачем оно выполняется.

Смысл соединения в том, что вы объединяете информацию из нескольких таблиц, в то время как каждая отдельная таблица содержит только часть нужной вам информации. Строки вывода соединения предоставляют более полные данные, чем строки каждой отдельной таблицы. В основе такой операции часто лежит сопоставление строк одной таблицы строкам другой, для чего таблицы должны содержать один или более столбцов общей информации, которая может использоваться для их логического связывания.

Предположим, что вы начали собирать коллекцию картин, используя две таблицы для записи своих приобретений: таблица `artist` содержит список

художников, чьи произведения вы хотели бы приобрести, а таблица `painting` перечисляет все картины, которые вы уже купили:

```
CREATE TABLE artist
(
    a_id    INT UNSIGNED NOT NULL AUTO_INCREMENT,   # идентификатор художника
    name    VARCHAR(30) NOT NULL,                   # фамилия художника
    PRIMARY KEY (a_id),
    UNIQUE (name)
);

CREATE TABLE painting
(
    a_id    INT UNSIGNED NOT NULL,                  # идентификатор художника
    p_id    INT UNSIGNED NOT NULL AUTO_INCREMENT,   # идентификатор картины
    title   VARCHAR(100) NOT NULL,                  # название картины
    state   VARCHAR(2) NOT NULL,                    # штат покупки
    price   INT UNSIGNED,                           # цена покупки (доллары)
    INDEX (a_id),
    PRIMARY KEY (p_id)
);
```

Вы только что приступили к коллекционированию, поэтому таблицы содержат только следующие записи:

```
mysql> SELECT * FROM artist ORDER BY a_id;
+------+----------+
| a_id | name     |
+------+----------+
|    1 | Da Vinci |
|    2 | Monet    |
|    3 | Van Gogh |
|    4 | Picasso  |
|    5 | Renoir   |
+------+----------+
mysql> SELECT * FROM painting ORDER BY a_id, p_id;
+------+------+-------------------+-------+-------+
| a_id | p_id | title             | state | price |
+------+------+-------------------+-------+-------+
|    1 |    1 | The Last Supper   | IN    |    34 |
|    1 |    2 | The Mona Lisa     | MI    |    87 |
|    3 |    3 | Starry Night      | KY    |    48 |
|    3 |    4 | The Potato Eaters | KY    |    67 |
|    3 |    5 | The Rocks         | IA    |    33 |
|    5 |    6 | Les Deux Soeurs   | NE    |    64 |
+------+------+-------------------+-------+-------+
```

Невысокие цены в столбце `price` таблицы `painting` показывают, что пока что ваша коллекция включает в себя только дешевые копии, а не оригиналы. Но это и понятно – кто же может позволить себе оригиналы?

Каждая из таблиц содержит частичную информацию о коллекции. Например, таблица `artist` не сообщает о том, какие картины написал каждый из

художников, а painting приводит только идентификаторы художников, без фамилий. Для ответа на некоторые вопросы необходимо объединить две таблицы, причем сделать это так, чтобы их записи были сопоставлены друг другу корректно. Для установления такого соответствия необходимо правильно составить инструкцию WHERE. В рецепте 12.1 я упоминал о том, что выполнение полного соединения обычно неразумно, так как формируется очень большой вывод. Еще одна причина, по которой не стоит выполнять полное соединение, – результат может оказаться бессмысленным. Полное соединение таблиц artist и painting – явный тому пример. Оно не содержит инструкции WHERE, поэтому выводит строки с бесполезной информацией:

```
mysql> SELECT * FROM artist, painting;
+------+----------+------+------+-------------------+-------+-------+
| a_id | name     | a_id | p_id | title             | state | price |
+------+----------+------+------+-------------------+-------+-------+
|    1 | Da Vinci |    1 |    1 | The Last Supper   | IN    |    34 |
|    2 | Monet    |    1 |    1 | The Last Supper   | IN    |    34 |
|    3 | Van Gogh |    1 |    1 | The Last Supper   | IN    |    34 |
|    4 | Picasso  |    1 |    1 | The Last Supper   | IN    |    34 |
|    5 | Renoir   |    1 |    1 | The Last Supper   | IN    |    34 |
|    1 | Da Vinci |    1 |    2 | The Mona Lisa     | MI    |    87 |
|    2 | Monet    |    1 |    2 | The Mona Lisa     | MI    |    87 |
|    3 | Van Gogh |    1 |    2 | The Mona Lisa     | MI    |    87 |
|    4 | Picasso  |    1 |    2 | The Mona Lisa     | MI    |    87 |
|    5 | Renoir   |    1 |    2 | The Mona Lisa     | MI    |    87 |
|    1 | Da Vinci |    3 |    3 | Starry Night      | KY    |    48 |
|    2 | Monet    |    3 |    3 | Starry Night      | KY    |    48 |
|    3 | Van Gogh |    3 |    3 | Starry Night      | KY    |    48 |
|    4 | Picasso  |    3 |    3 | Starry Night      | KY    |    48 |
|    5 | Renoir   |    3 |    3 | Starry Night      | KY    |    48 |
|    1 | Da Vinci |    3 |    4 | The Potato Eaters | KY    |    67 |
|    2 | Monet    |    3 |    4 | The Potato Eaters | KY    |    67 |
|    3 | Van Gogh |    3 |    4 | The Potato Eaters | KY    |    67 |
|    4 | Picasso  |    3 |    4 | The Potato Eaters | KY    |    67 |
|    5 | Renoir   |    3 |    4 | The Potato Eaters | KY    |    67 |
|    1 | Da Vinci |    3 |    5 | The Rocks         | IA    |    33 |
|    2 | Monet    |    3 |    5 | The Rocks         | IA    |    33 |
|    3 | Van Gogh |    3 |    5 | The Rocks         | IA    |    33 |
|    4 | Picasso  |    3 |    5 | The Rocks         | IA    |    33 |
|    5 | Renoir   |    3 |    5 | The Rocks         | IA    |    33 |
|    1 | Da Vinci |    5 |    6 | Les Deux Soeurs   | NE    |    64 |
|    2 | Monet    |    5 |    6 | Les Deux Soeurs   | NE    |    64 |
|    3 | Van Gogh |    5 |    6 | Les Deux Soeurs   | NE    |    64 |
|    4 | Picasso  |    5 |    6 | Les Deux Soeurs   | NE    |    64 |
|    5 | Renoir   |    5 |    6 | Les Deux Soeurs   | NE    |    64 |
+------+----------+------+------+-------------------+-------+-------+
```

Очевидно, что вы ведете эти таблицы не для того, чтобы сопоставлять каждому художнику каждую картину, как это проделано в запросе. Ничем не ограниченное соединение выводит массу ненужных строк, так что инструкция WHERE просто необходима для придания смысла запросу. Например, чтобы

вывести список картин с фамилиями художников, можно сопоставить записи двух таблиц, используя простую инструкцию WHERE, которой соответствуют строки с одинаковым значением столбца идентификатора художника (который связывает их друг с другом):

```
mysql> SELECT * FROM artist, painting
    -> WHERE artist.a_id = painting.a_id;
+------+----------+------+------+-------------------+-------+-------+
| a_id | name     | a_id | p_id | title             | state | price |
+------+----------+------+------+-------------------+-------+-------+
|    1 | Da Vinci |    1 |    1 | The Last Supper   | IN    |    34 |
|    1 | Da Vinci |    1 |    2 | The Mona Lisa     | MI    |    87 |
|    3 | Van Gogh |    3 |    3 | Starry Night      | KY    |    48 |
|    3 | Van Gogh |    3 |    4 | The Potato Eaters | KY    |    67 |
|    3 | Van Gogh |    3 |    5 | The Rocks         | IA    |    33 |
|    5 | Renoir   |    5 |    6 | Les Deux Soeurs   | NE    |    64 |
+------+----------+------+------+-------------------+-------+-------+
```

Имена столбцов в инструкции WHERE содержат спецификаторы таблиц, чтобы было понятно, какое именно значение a_id подлежит сравнению. Вывод указывает, кто написал какое произведение и, наоборот, какие картины каждого художника есть в вашей коллекции. Но, возможно, вывод излишне многословен (например, в нем присутствуют два одинаковых столбца a_id; один из таблицы artist, второй – из таблицы painting). Вероятно, вы захотите видеть значения a_id один раз. Или вообще не захотите видеть столбцы идентификаторов. Чтобы исключить их, добавьте список столбцов вывода, в который включены только имена интересующих вас столбцов:

```
mysql> SELECT artist.name, painting.title, painting.state, painting.price
    -> FROM artist, painting
    -> WHERE artist.a_id = painting.a_id;
+----------+-------------------+-------+-------+
| name     | title             | state | price |
+----------+-------------------+-------+-------+
| Da Vinci | The Last Supper   | IN    |    34 |
| Da Vinci | The Mona Lisa     | MI    |    87 |
| Van Gogh | Starry Night      | KY    |    48 |
| Van Gogh | The Potato Eaters | KY    |    67 |
| Van Gogh | The Rocks         | IA    |    33 |
| Renoir   | Les Deux Soeurs   | NE    |    64 |
+----------+-------------------+-------+-------+
```

Добавляя другие условия в инструкцию WHERE, вы можете использовать запросы, сопоставляющие строки таблиц, чтобы отвечать на более специфические вопросы, в том числе:

- Какие произведения создал Ван Гог? Чтобы получить ответ, найдем запись таблицы artist, соответствующую фамилии художника, используем ее значение a_id для нахождения записей в таблице painting и выберем из этих записей название картины:

```
mysql> SELECT painting.title
    -> FROM artist, painting
```

```
    -> WHERE artist.name = 'Van Gogh' AND artist.a_id = painting.a_id;
+-------------------+
| title             |
+-------------------+
| Starry Night      |
| The Potato Eaters |
| The Rocks         |
+-------------------+
```

- Кто написал «Мону Лизу»? Здесь будем двигаться в обратном направлении, используя информацию из таблицы `painting` для нахождения данных в таблице `artist`:

```
mysql> SELECT artist.name
    -> FROM artist, painting
    -> WHERE painting.title = 'The Mona Lisa' AND painting.a_id = artist.a_id;
+----------+
| name     |
+----------+
| Da Vinci |
+----------+
```

- Картины каких мастеров вы покупали в Кентукки и Индиане? Запрос похож на предыдущий, только проверяет другой столбец таблицы `painting` для нахождения множества записей, которое будет объединяться с таблицей `artist`:

```
mysql> SELECT DISTINCT artist.name
    -> FROM artist, painting
    -> WHERE painting.state IN ('KY','IN') AND artist.a_id = painting.a_id;
+----------+
| name     |
+----------+
| Da Vinci |
| Van Gogh |
+----------+
```

  Запрос использует `DISTINCT` для однократного вывода фамилии каждого художника. Попробуйте выполнить запрос без `DISTINCT` и вы увидите, что значение `Van Gogh` будет выведено дважды, так как вы купили две картины Ван Гога в Кентукки.

- Можно использовать соединения совместно с агрегирующими функциями для формирования итогов. Например, чтобы узнать, сколько у вас картин каждого художника, выполните такой запрос:

```
mysql> SELECT artist.name, COUNT(*) AS 'number of paintings'
    -> FROM artist, painting
    -> WHERE artist.a_id = painting.a_id
    -> GROUP BY artist.name;
+----------+--------------------+
| name     | number of paintings |
+----------+--------------------+
```

```
| Da Vinci |                   2 |
| Renoir   |                   1 |
| Van Gogh |                   3 |
+----------+---------------------+
```

Усовершенствуем его, чтобы определить, сколько вы заплатили за картины каждого автора, всего (`total price`) и в среднем за картину (`average price`):

```
mysql> SELECT artist.name,
    -> COUNT(*) AS 'number of paintings',
    -> SUM(painting.price) AS 'total price',
    -> AVG(painting.price) AS 'average price'
    -> FROM artist, painting WHERE artist.a_id = painting.a_id
    -> GROUP BY artist.name;
+----------+---------------------+-------------+---------------+
| name     | number of paintings | total price | average price |
+----------+---------------------+-------------+---------------+
| Da Vinci |                   2 |         121 |       60.5000 |
| Renoir   |                   1 |          64 |       64.0000 |
| Van Gogh |                   3 |         148 |       49.3333 |
+----------+---------------------+-------------+---------------+
```

Обратите внимание на то, что суммарный запрос формирует вывод только для тех художников из таблицы `artist`, картины которых вы действительно приобретали (например, Моне есть в таблице `artist`, но в выводе его нет, так как у вас еще нет ни одной его картины). Если вы хотите, чтобы итоги представляли всех авторов, даже если их произведений нет в вашей коллекции, используйте другой вид соединения – `LEFT JOIN` (см. рецепты 12.5 и 12.8).

Соединения и индексы

Поскольку соединение вполне может привести к обработке большого объема комбинаций строк, хорошо бы убедиться в том, что сравниваемые столбцы проиндексированы. В противном случае производительность может значительно ухудшаться при увеличении размера таблиц. Для таблиц `artist` и `painting` соединения выполняются на основе значений столбца `a_id` каждой таблицы. Если вы вернетесь к предложениям `CREATE TABLE` для этих таблиц (они были приведены в рецепте 12.4), то увидите, что столбец `a_id` индексирован в обеих таблицах.

12.5. Нахождение строк, которым не соответствуют никакие строки другой таблицы

Задача

Вы хотите найти строки одной таблицы, которым не соответствуют никакие строки другой. Или хотите вывести список на основе соединения таблиц, но

включить в него и записи, для которых не найдено соответствия в другой таблице.

Решение

Используйте LEFT JOIN. Начиная с версии MySQL 3.23.25, можно также использовать RIGHT JOIN.

Обсуждение

Предыдущие разделы были посвящены установлению соответствия строк двух таблиц. Но ответы на некоторые вопросы требуют определения того, для каких записей *не* найдено соответствия (или, иначе говоря, какие записи имеют значения, отсутствующие в другой таблице). Например, вы можете захотеть узнать, картин какого художника из таблицы artist у вас еще нет. Аналогичные вопросы могут возникать и в других ситуациях:

- Вы занимаетесь сбытом. У вас есть список потенциальных клиентов и список людей, которые уже разместили заказы. Чтобы сосредоточить свои усилия на тех, кто еще не стал вашим клиентом, необходимо найти участников первого списка, отсутствующих во втором.
- У вас есть один список бейсболистов, а второй – игроков, выполнивших перебежку на базу, за которую засчитывается очко. Вы хотите определить игроков, *не* выполнивших такую перебежку. Необходимо найти тех игроков из первого списка, которые не попали во второй.

Для ответов на такие вопросы вам нужно использовать LEFT JOIN.

Давайте определим, какие художники из таблицы artist отсутствуют в таблице painting. Сейчас таблицы небольшие, так что вы без труда можете просмотреть их визуально и выявить, что у вас нет картин Моне и Пикассо (нет записей painting со значением a_id, равным 2 или 4):

```
mysql> SELECT * FROM artist ORDER BY a_id;
+------+----------+
| a_id | name     |
+------+----------+
|    1 | Da Vinci |
|    2 | Monet    |
|    3 | Van Gogh |
|    4 | Picasso  |
|    5 | Renoir   |
+------+----------+
mysql> SELECT * FROM painting ORDER BY a_id, p_id;
+------+------+------------------+-------+-------+
| a_id | p_id | title            | state | price |
+------+------+------------------+-------+-------+
|    1 |    1 | The Last Supper  | IN    |    34 |
|    1 |    2 | The Mona Lisa    | MI    |    87 |
|    3 |    3 | Starry Night     | KY    |    48 |
|    3 |    4 | The Potato Eaters | KY    |    67 |
```

```
|    3 |    5 | The Rocks          | IA    |    33 |
|    5 |    6 | Les Deux Soeurs    | NE    |    64 |
+------+------+--------------------+-------+-------+
```

Но по мере пополнения коллекции размер таблиц будет увеличиваться, и будет уже не так легко охватить их взглядом и ответить на вопрос в результате простого просмотра. Можно ли получить ответ при помощи SQL? Конечно, хотя первая попытка решения задачи обычно выглядит как нечто вроде представленного ниже запроса, использующего инструкцию `WHERE` для нахождения несоответствий в таблицах:

```
mysql> SELECT * FROM artist, painting WHERE artist.a_id != painting.a_id;
+------+----------+------+------+--------------------+-------+-------+
| a_id | name     | a_id | p_id | title              | state | price |
+------+----------+------+------+--------------------+-------+-------+
|    2 | Monet    |    1 |    1 | The Last Supper    | IN    |    34 |
|    3 | Van Gogh |    1 |    1 | The Last Supper    | IN    |    34 |
|    4 | Picasso  |    1 |    1 | The Last Supper    | IN    |    34 |
|    5 | Renoir   |    1 |    1 | The Last Supper    | IN    |    34 |
|    2 | Monet    |    1 |    2 | The Mona Lisa      | MI    |    87 |
|    3 | Van Gogh |    1 |    2 | The Mona Lisa      | MI    |    87 |
|    4 | Picasso  |    1 |    2 | The Mona Lisa      | MI    |    87 |
|    5 | Renoir   |    1 |    2 | The Mona Lisa      | MI    |    87 |
|    1 | Da Vinci |    3 |    3 | Starry Night       | KY    |    48 |
|    2 | Monet    |    3 |    3 | Starry Night       | KY    |    48 |
|    4 | Picasso  |    3 |    3 | Starry Night       | KY    |    48 |
|    5 | Renoir   |    3 |    3 | Starry Night       | KY    |    48 |
|    1 | Da Vinci |    3 |    4 | The Potato Eaters  | KY    |    67 |
|    2 | Monet    |    3 |    4 | The Potato Eaters  | KY    |    67 |
|    4 | Picasso  |    3 |    4 | The Potato Eaters  | KY    |    67 |
|    5 | Renoir   |    3 |    4 | The Potato Eaters  | KY    |    67 |
|    1 | Da Vinci |    3 |    5 | The Rocks          | IA    |    33 |
|    2 | Monet    |    3 |    5 | The Rocks          | IA    |    33 |
|    4 | Picasso  |    3 |    5 | The Rocks          | IA    |    33 |
|    5 | Renoir   |    3 |    5 | The Rocks          | IA    |    33 |
|    1 | Da Vinci |    5 |    6 | Les Deux Soeurs    | NE    |    64 |
|    2 | Monet    |    5 |    6 | Les Deux Soeurs    | NE    |    64 |
|    3 | Van Gogh |    5 |    6 | Les Deux Soeurs    | NE    |    64 |
|    4 | Picasso  |    5 |    6 | Les Deux Soeurs    | NE    |    64 |
+------+----------+------+------+--------------------+-------+-------+
```

Очевидно, что получен неверный результат! Запрос выводит список всех несовпадающих комбинаций значений двух строк, но вам-то на самом деле нужен список значений `artist`, которых вообще нет в `painting`. Проблема в том, что обычное соединение может выводить только комбинации значений, присутствующих в таблицах. И ничего не может сказать об отсутствующих значениях.

Сталкиваясь с необходимостью нахождения значений одной таблицы, которым не найдено соответствий (или которые отсутствуют) в другой таблице, вы сразу же должны думать: «Ага! Все понятно. Нужно использовать `LEFT JOIN`».

LEFT JOIN (левое соединение) похоже на обычное соединение тем, что оно связывает строки первой (левой) таблицы со строками второй (правой) таблицы. Но в дополнение, если строке левой таблицы не найдено соответствия в правой, LEFT JOIN все же выводит строку, в которой все столбцы правой таблицы установлены в NULL. То есть для того чтобы найти значения, отсутствующие в правой таблице, можно искать NULL. Давайте рассмотрим операцию поэтапно. Сначала выполним обычное соединение для нахождения совпадающих строк:

```
mysql> SELECT * FROM artist, painting
    -> WHERE artist.a_id = painting.a_id;
+------+----------+------+------+-------------------+-------+-------+
| a_id | name     | a_id | p_id | title             | state | price |
+------+----------+------+------+-------------------+-------+-------+
|    1 | Da Vinci |    1 |    1 | The Last Supper   | IN    |    34 |
|    1 | Da Vinci |    1 |    2 | The Mona Lisa     | MI    |    87 |
|    3 | Van Gogh |    3 |    3 | Starry Night      | KY    |    48 |
|    3 | Van Gogh |    3 |    4 | The Potato Eaters | KY    |    67 |
|    3 | Van Gogh |    3 |    5 | The Rocks         | IA    |    33 |
|    5 | Renoir   |    5 |    6 | Les Deux Soeurs   | NE    |    64 |
+------+----------+------+------+-------------------+-------+-------+
```

В этом выводе первый столбец a_id получен из таблицы artist, а второй – из painting.

Теперь сравним этот результат с выводом, полученным при выполнении LEFT JOIN. Синтаксис запроса LEFT JOIN похож на синтаксис обычного соединения, только имена таблиц разделяются не запятой, а словами LEFT JOIN, а столбцы для сравнения задаются в инструкции ON, а не WHERE:

```
mysql> SELECT * FROM artist LEFT JOIN painting
    -> ON artist.a_id = painting.a_id;
+------+----------+------+------+-------------------+-------+-------+
| a_id | name     | a_id | p_id | title             | state | price |
+------+----------+------+------+-------------------+-------+-------+
|    1 | Da Vinci |    1 |    1 | The Last Supper   | IN    |    34 |
|    1 | Da Vinci |    1 |    2 | The Mona Lisa     | MI    |    87 |
|    2 | Monet    | NULL | NULL | NULL              | NULL  |  NULL |
|    3 | Van Gogh |    3 |    3 | Starry Night      | KY    |    48 |
|    3 | Van Gogh |    3 |    4 | The Potato Eaters | KY    |    67 |
|    3 | Van Gogh |    3 |    5 | The Rocks         | IA    |    33 |
|    4 | Picasso  | NULL | NULL | NULL              | NULL  |  NULL |
|    5 | Renoir   |    5 |    6 | Les Deux Soeurs   | NE    |    64 |
+------+----------+------+------+-------------------+-------+-------+
```

Вывод похож на вывод обычного соединения, но LEFT JOIN формирует и строку вывода для тех строк таблицы artist, которым не найдено соответствия в painting. В этих строках вывода все столбцы painting установлены в NULL.

На следующем шаге ограничим вывод только теми строками таблицы artist, которым не найдено соответствия, добавив инструкцию WHERE для поиска значений NULL в столбце painting, имя которого указано в инструкции ON:

```
mysql> SELECT * FROM artist LEFT JOIN painting
    -> ON artist.a_id = painting.a_id
    -> WHERE painting.a_id IS NULL;
+------+---------+------+------+-------+-------+
| a_id | name    | a_id | p_id | title | price |
+------+---------+------+------+-------+-------+
|    2 | Monet   | NULL | NULL | NULL  |  NULL |
|    4 | Picasso | NULL | NULL | NULL  |  NULL |
+------+---------+------+------+-------+-------+
```

Наконец, чтобы вывести только те значения `artist`, которые отсутствуют в `painting`, уменьшим список столбцов вывода, включив в него только столбцы таблицы `artist`:

```
mysql> SELECT artist.* FROM artist LEFT JOIN painting
    -> ON artist.a_id = painting.a_id
    -> WHERE painting.a_id IS NULL;
+------+---------+
| a_id | name    |
+------+---------+
|    2 | Monet   |
|    4 | Picasso |
+------+---------+
```

Рассмотренное левое соединение `LEFT JOIN` перечисляет те значения левой таблицы, которые отсутствуют в правой. Похожую операцию можно использовать для вывода всех значений левой таблицы вместе с индикаторами их наличия в правой таблице. Выполните `LEFT JOIN` для подсчета количества вхождений значения левой таблицы в правую. Нулевой счетчик означает отсутствие значения. Следующий запрос перечисляет всех художников из таблицы `artist` и сообщает о том, есть ли у вас их картины:

```
mysql> SELECT artist.name,
    -> IF(COUNT(painting.a_id)>0,'yes','no') AS 'in collection'
    -> FROM artist LEFT JOIN painting ON artist.a_id = painting.a_id
    -> GROUP BY artist.name;
+----------+---------------+
| name     | in collection |
+----------+---------------+
| Da Vinci | yes           |
| Monet    | no            |
| Picasso  | no            |
| Renoir   | yes           |
| Van Gogh | yes           |
+----------+---------------+
```

Начиная с версии MySQL 3.23.25 вы также можете использовать `RIGHT JOIN` (правое соединение), которое аналогично `LEFT JOIN`, только в нем меняются местами левая и правая таблицы. Другими словами, вывод `RIGHT JOIN` содержит строку для каждой строки правой таблицы, даже если ей не найдено соответствие в левой таблице. Можно преобразовать предыдущее соединение `LEFT JOIN` в `RIGHT JOIN` с выводом того же результата так:

```
mysql> SELECT artist.name,
    -> IF(COUNT(painting.a_id)>0,'yes','no') AS 'in collection'
    -> FROM painting RIGHT JOIN artist ON painting.a_id = artist.a_id
    -> GROUP BY artist.name;
+----------+---------------+
| name     | in collection |
+----------+---------------+
| Da Vinci | yes           |
| Monet    | no            |
| Picasso  | no            |
| Renoir   | yes           |
| Van Gogh | yes           |
+----------+---------------+
```

Далее в книге я для краткости буду говорить только о LEFT JOIN, но все рассуждения применимы и к RIGHT JOIN, если вы поменяете таблицы местами.

Другие способы создания запросов LEFT JOIN и RIGHT JOIN

Если имена сравниваемых столбцов двух таблиц совпадают, то можно использовать альтернативу ON при создании запросов LEFT JOIN и RIGHT JOIN. В синтаксической конструкции USING заменяет ON. Например, два таких запроса эквиваленты:

```
SELECT t1.n, t2.n FROM t1 LEFT JOIN t2 ON t1.n = t2.n;
SELECT t1.n, t2.n FROM t1 LEFT JOIN t2 USING (n);
```

Как и такие:

```
SELECT t1.n, t2.n FROM t1 RIGHT JOIN t2 ON t1.n = t2.n;
SELECT t1.n, t2.n FROM t1 RIGHT JOIN t2 USING (n);
```

Если вы хотите использовать для сравнения все столбцы, присутствующие в обеих таблицах, можно применить NATURAL LEFT JOIN или NATURAL RIGHT JOIN:

```
SELECT t1.n, t2.n FROM t1 NATURAL LEFT JOIN t2;
SELECT t1.n, t2.n FROM t1 NATURAL RIGHT JOIN t2;
```

См. также

Как показано в этом разделе, LEFT JOIN удобно использовать для нахождения значений, которым нет соответствий в другой таблице или для проверки вхождения во вторую таблицу каждого из значений первой. LEFT JOIN можно использовать и для формирования итогов, где в список вывода попадут все элементы, даже те, по которым нечего суммировать. Такая операция часто применяется для характеристики отношений между главной и подчиненной таблицами. Например, LEFT JOIN может выводить отчеты об объемах продаж по клиентам, включая в них даже клиентов, которые ничего не приобретали в течение отчетного периода (см. рецепт 12.8).

Еще одним приложением LEFT JOIN является проверка непротиворечивости данных при получении двух файлов, про которые предполагается, что они связаны друг с другом, и хотелось бы это проверить. (То есть вы хотите проверить ссылочную целостность.) Импортируйте каждый файл в таблицу MySQL и выполните пару предложений LEFT JOIN для определения того, нет ли в одной или другой таблице несвязанных записей, то есть записей, которым не найдено соответствия в другой таблице. (Если такие записи обнаружены и вы хотите их удалить, обратитесь к рецепту 12.21.)

12.6. Нахождение строк с минимальным и максимальным значениями в группе

Задача

Вы хотите определить, какая запись в каждой группе строк таблицы содержит максимальное или минимальное значение указанного столбца. Например, вы хотите узнать, какая из картин каждого художника оказалась наиболее дорогой.

Решение

Создайте временную таблицу для хранения максимума или минимума группы, затем объедините временную таблицу с исходной для извлечения соответствующей записи в каждой группе.

Обсуждение

Во многих задачах требуется нахождение наибольшего или наименьшего значения какого-то столбца, но при этом часто необходимо знать и значения других столбцов строки, содержащей такое значение. Например, вы можете использовать MAX(pop) для определения максимального населения штата в таблице states, но хотелось бы также получить ответ на вопрос о том, какой штат имеет такое население. В рецепте 7.5 было показано, что одним из способов получения ответа является использование переменной SQL:

```
mysql> SELECT @max := MAX(pop) FROM states;
mysql> SELECT * FROM states WHERE pop = @max;
+------------+--------+------------+----------+
| name       | abbrev | statehood  | pop      |
+------------+--------+------------+----------+
| California | CA     | 1850-09-09 | 29760021 |
+------------+--------+------------+----------+
```

Можно использовать и соединение. Сначала выберем максимальное значение численности населения во временную таблицу:

```
mysql> CREATE TABLE tmp SELECT MAX(pop) as maxpop FROM states;
```

Затем объединим временную таблицу с исходной для поиска записи, соответствующей выбранному значению численности населения:

```
mysql> SELECT states.* FROM states, tmp WHERE states.pop = tmp.maxpop;
+------------+--------+------------+----------+
| name       | abbrev | statehood  | pop      |
+------------+--------+------------+----------+
| California | CA     | 1850-09-09 | 29760021 |
+------------+--------+------------+----------+
```

Применяя описанные методы к таблицам `artist` и `painting`, вы сможете ответить на вопросы типа «Какая картина коллекции была самой дорогой, кто ее написал?». Если вы используете переменную SQL, сохраните в ней наибольшую цену, затем используйте переменную для идентификации записи, содержащей такое значение цены, с тем, чтобы извлечь из нее другие столбцы:

```
mysql> SELECT @max_price := MAX(price) FROM painting;
mysql> SELECT artist.name, painting.title, painting.price
    -> FROM artist, painting
    -> WHERE painting.price = @max_price
    -> AND painting.a_id = artist.a_id;
+----------+---------------+-------+
| name     | title         | price |
+----------+---------------+-------+
| Da Vinci | The Mona Lisa |    87 |
+----------+---------------+-------+
```

То же самое можно сделать, создав временную таблицу для хранения максимальной цены, а затем объединив ее с другими таблицами:

```
mysql> CREATE TABLE tmp SELECT MAX(price) AS max_price FROM painting;
mysql> SELECT artist.name, painting.title, painting.price
    -> FROM artist, painting, tmp
    -> WHERE painting.price = tmp.max_price
    -> AND painting.a_id = artist.a_id;
+----------+---------------+-------+
| name     | title         | price |
+----------+---------------+-------+
| Da Vinci | The Mona Lisa |    87 |
+----------+---------------+-------+
```

На первый взгляд кажется, что использование временной таблицы и соединения – это более сложный способ получения ответа на вопрос. Есть ли смысл применять его? Да, потому что он является основой общей техники решения более сложных задач. Предыдущие запросы выводят информацию только для одной самой дорогой картины всей таблицы `painting`. А если бы вопрос стоял так: «Какая картина каждого художника была самой дорогой?». Для ответа на этот вопрос нельзя использовать переменную SQL, так как необходимо найти по одному значению цены на каждого художника, а переменная может хранить только одно значение одновременно. Временная таблица может хранить несколько значений, и соединение может искать соответствия для всех таких значений сразу. Для получения ответа на последний вопрос выберите идентификатор каждого автора и максимальную цену его картины во временную таблицу. Таблица будет содержать не просто максимальную цену картины, а максимумы для каждой группы, где

группа – это картины определенного художника. Затем используйте хранящиеся в таблице `tmp` идентификаторы авторов и цены для сравнения с записями таблицы `painting` и объедините результат с таблицей `artist` для получения фамилий художников:

```
mysql> CREATE TABLE tmp
    -> SELECT a_id, MAX(price) AS max_price FROM painting GROUP BY a_id;
mysql> SELECT artist.name, painting.title, painting.price
    -> FROM artist, painting, tmp
    -> WHERE painting.a_id = tmp.a_id
    -> AND painting.price = tmp.max_price
    -> AND painting.a_id = artist.a_id;
+----------+-------------------+-------+
| name     | title             | price |
+----------+-------------------+-------+
| Da Vinci | The Mona Lisa     |    87 |
| Van Gogh | The Potato Eaters |    67 |
| Renoir   | Les Deux Soeurs   |    64 |
+----------+-------------------+-------+
```

Аналогичные манипуляции можно проводить с другими видами значений, например, со значениями даты и времени. Рассмотрим таблицу `driver_log`, в которой перечислены водители и их поездки:

```
mysql> SELECT name, trav_date, miles
    -> FROM driver_log
    -> ORDER BY name, trav_date;
+-------+------------+-------+
| name  | trav_date  | miles |
+-------+------------+-------+
| Ben   | 2001-11-29 |   131 |
| Ben   | 2001-11-30 |   152 |
| Ben   | 2001-12-02 |    79 |
| Henry | 2001-11-26 |   115 |
| Henry | 2001-11-27 |    96 |
| Henry | 2001-11-29 |   300 |
| Henry | 2001-11-30 |   203 |
| Henry | 2001-12-01 |   197 |
| Suzi  | 2001-11-29 |   391 |
| Suzi  | 2001-12-02 |   502 |
+-------+------------+-------+
```

Одной из возможных задач нахождения максимума в группе для этой таблицы будет вывод последней поездки каждого водителя. Решить ее можно так:

```
mysql> CREATE TABLE tmp
    -> SELECT name, MAX(trav_date) AS trav_date
    -> FROM driver_log GROUP BY name;
mysql> SELECT driver_log.name, driver_log.trav_date, driver_log.miles
    -> FROM driver_log, tmp
    -> WHERE driver_log.name = tmp.name
    -> AND driver_log.trav_date = tmp.trav_date
```

```
    -> ORDER BY driver_log.name;
+-------+------------+-------+
| name  | trav_date  | miles |
+-------+------------+-------+
| Ben   | 2001-12-02 |    79 |
| Henry | 2001-12-01 |   197 |
| Suzi  | 2001-12-02 |   502 |
+-------+------------+-------+
```

См. также

Описанный прием показывает, как находить максимальное значение в группе, выбирая итоговую информацию во временную таблицу и объединяя эту таблицу с исходной. Данный прием используется весьма часто. Одно из его возможных применений – вычисление рейтинга команды, который для каждой группы определяется путем сравнения лучшего результата в группе с результатами всех команд группы. Решению этой задачи посвящен рецепт 12.7.

12.7. Вычисление рейтинга команд

Задача

Вы хотите вычислить рейтинг команд по записям об их победах и поражениях, включая значения «отставания в играх» (GB – games-behind).

Решение

Определите, какая команда занимает первое место, затем объедините результат с исходными записями.

Обсуждение

Рейтинги спортивных команд, соревнующихся друг с другом, обычно определяются по соотношению победы-поражения, при этом командам, занявшим не первое место, присваивается значение «отставания в играх», показывающее, на сколько матчей они отстают от первого места. В разделе будет рассказано о том, как вычислять такие значения. В первом примере для пояснения логики вычислений используется таблица, содержащая всего один набор записей о командах. Затем будет рассмотрена вторая таблица, включающая несколько наборов записей; в этом случае нужно будет использовать соединение, чтобы выполнить независимые вычисления для каждой группы команд.

Рассмотрим таблицу `standings1`, которая содержит один набор записей о бейсбольных командах (они представляют итоговые результаты Северной Лиги за 1902 год):

```
mysql> SELECT team, wins, losses FROM standings1
    -> ORDER BY wins-losses DESC;
+-------------+------+--------+
| team        | wins | losses |
```

```
+-------------+------+--------+
| Winnipeg    |   37 |     20 |
| Crookston   |   31 |     25 |
| Fargo       |   30 |     26 |
| Grand Forks |   28 |     26 |
| Devils Lake |   19 |     31 |
| Cavalier    |   15 |     32 |
+-------------+------+--------+
```

Записи упорядочены по разности побед-поражений; именно так формируются списки команд от первого к последнему месту. Но рейтинг команды обычно включает в себя процент побед каждой команды и число, показывающее, на сколько игр лидер опередил всех остальных. Давайте добавим эту информацию в наш вывод. Вычислить процент просто – это отношение количества побед к общему количеству игр, которое вычисляется так:

```
wins / (wins + losses)
```

Если вы хотите выполнить расчеты для команды, которая еще не приступила к играм, то выражение будет вычислено как `NULL` из-за деления на ноль. Для простоты я буду предполагать, что количество игр отлично от нуля, но если вы хотите обрабатывать это условие, заменяя `NULL` нулем, то выражение можно обобщить так:

```
IFNULL(wins / (wins + losses),0)
```

или:

```
wins / IF(wins=0,1,wins + losses)
```

Определить значение «отставания в играх» несколько сложнее. Оно зависит от отношения побед-поражений двух команд и вычисляется как среднее арифметическое двух значений:

- Количество игр, которое занимающая второе место команда должна выиграть, чтобы иметь то же число побед, что и команда-лидер.
- Количество игр, которое лидер должен проиграть, чтобы иметь то же число поражений, что и команда, занимающая второе место.

Предположим, например, что для двух команд A и B имеются такие записи о победах-поражениях:

```
+------+------+--------+
| team | wins | losses |
+------+------+--------+
| A    |   17 |     11 |
| B    |   14 |     12 |
+------+------+--------+
```

Команда B должна выиграть еще три игры, а команда A – проиграть одну, для того чтобы команды сравнялись. Среднее арифметическое трех и одного равно двум, следовательно, B на две игры позади A. Математически можно выразить «отставание в играх» для вычисления так:

```
((winsA - winsB) + (lossesB - lossesA)) / 2
```

Перегруппируем выражение и получим:

```
((winsA - lossesA) - (winsB - lossesB)) / 2
```

Второе выражение эквивалентно первому, но каждая его составляющая представлена разностью побед и поражений одной команды, а не сравнением результатов двух команд. Со вторым выражением легче работать, поскольку каждый элемент не зависит от другого и определяется записями только одной команды. Первый элемент соответствует разности побед и поражений команды-лидера, так что если начать с вычисления этого значения, все остальные «отставания в играх» можно будет определить по отношению к нему.

У команды, занимающей первое место, самая большая разница количеств побед и поражений. Выполним запрос для нахождения этого значения и сохранения его в переменной:

```
mysql> SELECT @wl_diff := MAX(wins-losses) FROM standings1;
+------------------------------+
| @wl_diff := MAX(wins-losses) |
+------------------------------+
|                           17 |
+------------------------------+
```

Теперь используем эту разность для вывода рейтинга команд, включающего процент выигрышей и значения «отставания в играх»:

```
mysql> SELECT team, wins AS W, losses AS L,
    -> wins/(wins+losses) AS PCT,
    -> (@wl_diff - (wins-losses)) / 2 AS GB
    -> FROM standings1
    -> ORDER BY wins-losses DESC, PCT DESC;
+-------------+------+------+------+------+
| team        | W    | L    | PCT  | GB   |
+-------------+------+------+------+------+
| Winnipeg    |   37 |   20 | 0.65 |    0 |
| Crookston   |   31 |   25 | 0.55 |  5.5 |
| Fargo       |   30 |   26 | 0.54 |  6.5 |
| Grand Forks |   28 |   26 | 0.52 |  7.5 |
| Devils Lake |   19 |   31 | 0.38 | 14.5 |
| Cavalier    |   15 |   32 | 0.32 |   17 |
+-------------+------+------+------+------+
```

Обратимся теперь к некоторым тонкостям форматирования. Процентные отношения в рейтингах обычно представляются тремя разрядами, а значение «отставания в играх» для лидера выводится как знак -, а не 0. Чтобы выводить три десятичных разряда, используем TRUNCATE(*выражение*,3). Чтобы соответствующим образом выводить значение «отставания в играх» для первой команды, поместим *выражение*, вычисляющее столбец отставаний, внутрь вызова IF(), который преобразует 0 в тире:

```
mysql> SELECT team, wins AS W, losses AS L,
    -> TRUNCATE(wins/(wins+losses),3) AS PCT,
    -> IF((@wl_diff - (wins-losses)) = 0,'-',(@wl_diff - (wins-losses))/2) AS GB
```

```
    -> FROM standings1
    -> ORDER BY wins-losses DESC, PCT DESC;
+-------------+------+------+-------+------+
| team        | W    | L    | PCT   | GB   |
+-------------+------+------+-------+------+
| Winnipeg    |   37 |   20 | 0.649 | -    |
| Crookston   |   31 |   25 | 0.553 | 5.5  |
| Fargo       |   30 |   26 | 0.535 | 6.5  |
| Grand Forks |   28 |   26 | 0.518 | 7.5  |
| Devils Lake |   19 |   31 | 0.380 | 14.5 |
| Cavalier    |   15 |   32 | 0.319 | 17   |
+-------------+------+------+-------+------+
```

Эти запросы упорядочивают команды по разности побед-поражений, используя процент побед для сортировки команд с одинаковым значением разности. Конечно, было бы проще упорядочивать значения по процентному отношению, но результат не всегда был бы корректным. Как это ни удивительно, но команда с более низким процентом побед может на самом деле занимать более высокое положение в турнирной таблице, чем команда с более высоким процентом побед. (Обычно так бывает в начале сезона, когда у команд может сильно отличаться количество сыгранных матчей.) Рассмотрим случай с двумя командами A и B, имеющими такие результаты:

```
+------+------+--------+
| team | wins | losses |
+------+------+--------+
| A    |    4 |      1 |
| B    |    2 |      0 |
+------+------+--------+
```

Вычисление для этих команд процента побед и «отставания в играх» приводит к следующим результатам, из которых видно, что команда, занимающая первое место, имеет более низкий процент побед, чем вторая команда:

```
+------+------+------+-------+------+
| team | W    | L    | PCT   | GB   |
+------+------+------+-------+------+
| A    |    4 |    1 | 0.800 | -    |
| B    |    2 |    0 | 1.000 | 0.5  |
+------+------+------+-------+------+
```

Рассмотренные вычисления рейтингов можно выполнить без использования соединения. Они проводятся всего для одной группы команд, так что разность количества побед и поражений лидера может храниться в переменной. Если же множество данных охватывает несколько групп команд, ситуация усложняется. Например, в 1997 году Северная Лига включала два дивизиона: Восточный и Западный. Кроме того, отдельные рейтинги велись для первой и второй половины сезона, так как победители первой половины сезона в каждом дивизионе играли друг с другом за право выступать в Лиге чемпионов. Таблица `standings2` приводит такие записи, упорядоченные по половине сезона, дивизиону и разности побед-поражений:

```
mysql> SELECT half, div, team, wins, losses FROM standings2
    -> ORDER BY half, div, wins-losses DESC;
+------+---------+-----------------+------+--------+
| half | div     | team            | wins | losses |
+------+---------+-----------------+------+--------+
|    1 | Eastern | St. Paul        |   24 |     18 |
|    1 | Eastern | Thunder Bay     |   18 |     24 |
|    1 | Eastern | Duluth-Superior |   17 |     24 |
|    1 | Eastern | Madison         |   15 |     27 |
|    1 | Western | Winnipeg        |   29 |     12 |
|    1 | Western | Sioux City      |   28 |     14 |
|    1 | Western | Fargo-Moorhead  |   21 |     21 |
|    1 | Western | Sioux Falls     |   15 |     27 |
|    2 | Eastern | Duluth-Superior |   22 |     20 |
|    2 | Eastern | St. Paul        |   21 |     21 |
|    2 | Eastern | Madison         |   19 |     23 |
|    2 | Eastern | Thunder Bay     |   18 |     24 |
|    2 | Western | Fargo-Moorhead  |   26 |     16 |
|    2 | Western | Winnipeg        |   24 |     18 |
|    2 | Western | Sioux City      |   22 |     20 |
|    2 | Western | Sioux Falls     |   16 |     26 |
+------+---------+-----------------+------+--------+
```

Для формирования рейтингов необходимо вычислить значения «отставания в играх» отдельно для каждой из четырех комбинаций половины сезона с дивизионом. Начнем с вычисления разности побед-поражений для команды-лидера каждой группы и сохранения значений в специальной таблице `firstplace`:

```
mysql> CREATE TABLE firstplace
    -> SELECT half, div, MAX(wins-losses) AS wl_diff
    -> FROM standings2
    -> GROUP BY half, div;
```

Затем объединим таблицу `firstplace` с исходной, сопоставляя записи каждой команды соответствующую разность побед-поражений для получения ее значения «отставания в играх»:

```
mysql> SELECT wl.half, wl.div, wl.team, wl.wins AS W, wl.losses AS L,
    -> TRUNCATE(wl.wins/(wl.wins+wl.losses),3) AS PCT,
    -> IF((fp.wl_diff - (wl.wins-wl.losses)) = 0,
    -> '-', (fp.wl_diff - (wl.wins-wl.losses)) / 2) AS GB
    -> FROM standings2 AS wl, firstplace AS fp
    -> WHERE wl.half = fp.half AND wl.div = fp.div
    -> ORDER BY wl.half, wl.div, wl.wins-wl.losses DESC, PCT DESC;
+------+---------+-----------------+------+------+-------+-------+
| half | div     | team            | W    | L    | PCT   | GB    |
+------+---------+-----------------+------+------+-------+-------+
|    1 | Eastern | St. Paul        |   24 |   18 | 0.571 | -     |
|    1 | Eastern | Thunder Bay     |   18 |   24 | 0.428 | 6.00  |
|    1 | Eastern | Duluth-Superior |   17 |   24 | 0.414 | 6.50  |
|    1 | Eastern | Madison         |   15 |   27 | 0.357 | 9.00  |
```

```
|    1 | Western | Winnipeg        |   29 |   12 | 0.707 | -     |
|    1 | Western | Sioux City      |   28 |   14 | 0.666 | 1.50  |
|    1 | Western | Fargo-Moorhead  |   21 |   21 | 0.500 | 8.50  |
|    1 | Western | Sioux Falls     |   15 |   27 | 0.357 | 14.50 |
|    2 | Eastern | Duluth-Superior |   22 |   20 | 0.523 | -     |
|    2 | Eastern | St. Paul        |   21 |   21 | 0.500 | 1.00  |
|    2 | Eastern | Madison         |   19 |   23 | 0.452 | 3.00  |
|    2 | Eastern | Thunder Bay     |   18 |   24 | 0.428 | 4.00  |
|    2 | Western | Fargo-Moorhead  |   26 |   16 | 0.619 | -     |
|    2 | Western | Winnipeg        |   24 |   18 | 0.571 | 2.00  |
|    2 | Western | Sioux City      |   22 |   20 | 0.523 | 4.00  |
|    2 | Western | Sioux Falls     |   16 |   26 | 0.380 | 10.00 |
+------+---------+-----------------+------+------+-------+-------+
```

Надо сказать, что такой вывод трудно воспринять. Чтобы сделать его понятнее, вероятно, стоит выполнить запрос в программе и переформатировать его результаты так, чтобы записи каждой группы выводились отдельно. Приведем код на Perl, который начинает новую группу вывода каждый раз, когда встречается новая группа рейтингов. Предполагается, что соединение было выполнено только что, и его результаты доступны через дескриптор предложения `$sth`:

```
my ($cur_half, $cur_div) = ("", "");
while (my ($half, $div, $team, $wins, $losses, $pct, $gb)
            = $sth->fetchrow_array ())
{
    if ($cur_half ne $half || $cur_div ne $div) # новая группа рейтингов?
    {
        # вывести заголовок и запомнить новые значения половины и дивизиона
        print "\n$div Division, season half $half\n";
        printf "%-20s  %3s  %3s  %5s  %s\n", "Team", "W", "L", "PCT", "GB";
        $cur_half = $half;
        $cur_div = $div;
    }
    printf "%-20s  %3d  %3d  %5s  %s\n", $team, $wins, $losses, $pct, $gb;
}
```

Преобразованный вывод выглядит так:

```
Eastern Division, season half 1
Team                  W    L   PCT  GB
St. Paul             24   18  0.57  -
Thunder Bay          18   24  0.43  6.00
Duluth-Superior      17   24  0.41  6.50
Madison              15   27  0.36  9.00

Western Division, season half 1
Team                  W    L   PCT  GB
Winnipeg             29   12  0.71  -
Sioux City           28   14  0.67  1.50
Fargo-Moorhead       21   21  0.50  8.50
Sioux Falls          15   27  0.36  14.50
```

```
Eastern Division, season half 2
Team                    W    L   PCT  GB
Duluth-Superior        22   20  0.52  -
St. Paul               21   21  0.50  1.00
Madison                19   23  0.45  3.00
Thunder Bay            18   24  0.43  4.00

Western Division, season half 2
Team                    W    L   PCT  GB
Fargo-Moorhead         26   16  0.62  -
Winnipeg               24   18  0.57  2.00
Sioux City             22   20  0.52  4.00
Sioux Falls            16   26  0.38  10.00
```

Приведенный выше код, формирующий полнотекстовый вывод, взят из сценария *calc_standings.pl* каталога *joins* дистрибутива `recipes`. Этот же каталог содержит сценарий на PHP, *calc_standings.php*, который реализует другой подход, генерируя вывод в виде HTML-таблиц, что может быть удобнее для формирования рейтингов в веб-среде.

12.8. Вывод списков для записей «главная–подчиненная» и итогов

Задача

Две таблицы соотносятся как «главная-подчиненная» (master-detail), и вы хотите вывести список, который для каждой главной записи выводит соответствующие ей подчиненные, или список, в котором просуммированы подчиненные записи для каждой главной.

Решение

Для решения задачи необходимо использовать соединение, тип которого зависит от конкретной задачи. Чтобы вывести список, содержащий только те главные записи, для которых есть какие-то подчиненные, применяйте обычное соединение на основе первичного ключа главной таблицы. Чтобы вывести список, который включает все главные записи, даже если для них нет подчиненных, используйте `LEFT JOIN`.

Обсуждение

Часто требуется сформировать список записей двух связанных таблиц. Если таблицы имеют связь «главная-подчиненная» (или связь «предок-потомок», parent-child), то указанная запись одной таблицы может соответствовать нескольким записям другой. Этот раздел описывает несколько таких вопросов, которые можно задать (и на которые можно ответить) для таблиц `artist` и `painting`, введенных ранее в этой главе.

Одним из вопросов типа «главная-подчиненная» для этих таблиц может быть такой: «Какой художник написал каждую из картин?». Вот простое со-

единение, сопоставляющее каждую запись таблицы painting соответствующей записи artist на основе значений идентификаторов художников:

```
mysql> SELECT artist.name, painting.title
    -> FROM artist, painting WHERE artist.a_id = painting.a_id
    -> ORDER BY 1, 2;
+----------+-------------------+
| name     | title             |
+----------+-------------------+
| Da Vinci | The Last Supper   |
| Da Vinci | The Mona Lisa     |
| Renoir   | Les Deux Soeurs   |
| Van Gogh | Starry Night      |
| Van Gogh | The Potato Eaters |
| Van Gogh | The Rocks         |
+----------+-------------------+
```

Такого соединения достаточно, пока вас интересуют только те главные записи, которым соответствуют подчиненные. Однако есть и другие вопросы типа «главная-подчиненная», например, «Какие картины написал каждый из авторов?». Этот вопрос похож на предыдущий, но не совсем идентичен ему. И ответ на него будет другим, если есть художники, приведенные в таблице artist, но не представленные в таблице painting. Для вывода правильного ответа необходимо использовать не такой запрос, как в предыдущем примере. В данном случае вывод соединения должен включать те записи одной таблицы, для которых не найдено соответствия в другой. Это разновидность задачи нахождения записей без соответствий (см. рецепт 12.5), поэтому для вывода всех записей artist, вне зависимости от того, существуют ли для них записи painting, используйте LEFT JOIN:

```
mysql> SELECT artist.name, painting.title
    -> FROM artist LEFT JOIN painting ON artist.a_id = painting.a_id
    -> ORDER BY 1, 2;
+----------+-------------------+
| name     | title             |
+----------+-------------------+
| Da Vinci | The Last Supper   |
| Da Vinci | The Mona Lisa     |
| Monet    | NULL              |
| Picasso  | NULL              |
| Renoir   | Les Deux Soeurs   |
| Van Gogh | Starry Night      |
| Van Gogh | The Potato Eaters |
| Van Gogh | The Rocks         |
+----------+-------------------+
```

Строки результирующего множества, которые содержат NULL в столбце title, соответствуют авторам из таблицы artist, картин которых у вас еще нет.

Те же принципы применяются к формированию итогов с использованием главной и подчиненной таблиц. Например, чтобы сформировать такую итоговую информацию о вашей коллекции, как количество картин каждого ху-

дожника, можно спросить: «Сколько картин каждого автора есть в таблице painting?». Чтобы вывести в ответе идентификаторы художников, можно сосчитать картины, выполнив такой запрос:

```
mysql> SELECT a_id, COUNT(a_id) AS count FROM painting GROUP BY a_id;
+------+-------+
| a_id | count |
+------+-------+
|    1 |     2 |
|    3 |     3 |
|    5 |     1 |
+------+-------+
```

Конечно, в таком выводе не очень много смысла, если только вы не помните, какому художнику соответствует каждый идентификатор. Для вывода фамилий художников вместо их идентификаторов объединим таблицу painting с таблицей artist:

```
mysql> SELECT artist.name AS painter, COUNT(painting.a_id) AS count
    -> FROM artist, painting
    -> WHERE artist.a_id = painting.a_id
    -> GROUP BY artist.name;
+----------+-------+
| painter  | count |
+----------+-------+
| Da Vinci |     2 |
| Renoir   |     1 |
| Van Gogh |     3 |
+----------+-------+
```

Можно сформулировать вопрос по-другому: «Сколько картин написал каждый художник?». Это такой же вопрос, как предыдущий, и ответ на него будет таким же, если каждому художнику из таблицы artist соответствует хотя бы одна запись таблицы painting. Но если у вас есть авторы, которые еще не представлены никакими работами в вашей коллекции, то они не будут присутствовать в выводе запроса. Чтобы сформировать итоги и по тем художникам, чьих произведений нет в таблице painting, используйте LEFT JOIN:

```
mysql> SELECT artist.name AS painter, COUNT(painting.a_id) AS count
    -> FROM artist LEFT JOIN painting ON artist.a_id = painting.a_id
    -> GROUP BY artist.name;
+----------+-------+
| painter  | count |
+----------+-------+
| Da Vinci |     2 |
| Monet    |     0 |
| Picasso  |     0 |
| Renoir   |     1 |
| Van Gogh |     3 |
+----------+-------+
```

При создании запросов такого типа легко сделать одну малозаметную ошибку. Предположим, что вы написали запрос несколько иначе:

```
mysql> SELECT artist.name AS painter, COUNT(*) AS count
    -> FROM artist LEFT JOIN painting ON artist.a_id = painting.a_id
    -> GROUP BY artist.name;
+----------+-------+
| painter  | count |
+----------+-------+
| Da Vinci |     2 |
| Monet    |     1 |
| Picasso  |     1 |
| Renoir   |     1 |
| Van Gogh |     3 |
+----------+-------+
```

Теперь у каждого автора есть хотя бы одна картина. Почему? Причина проблемы в использовании `COUNT(*)` вместо `COUNT(painting.a_id)`. Соединение `LEFT JOIN` работает со строками левой таблицы, которым не найдено соответствий, так: генерируется строка, в которой все столбцы правой таблицы установлены в `NULL`. В нашем примере правой таблицей является `painting`. Запрос, использующий `COUNT(painting.a_id)`, работает корректно, так как `COUNT(`*выражение*`)` не учитывает значения `NULL`. Запрос, использующий `COUNT(*)`, работает неправильно, так как он подсчитывает все значения, даже строки, соответствующие отсутствующим художникам.

`LEFT JOIN` подходит и для формирования других видов итогов. Чтобы вывести дополнительные столбцы, содержащие общую и среднюю стоимость картин каждого автора из таблицы `artist`, выполните такой запрос:

```
mysql> SELECT artist.name AS painter,
    -> COUNT(painting.a_id) AS 'number of paintings',
    -> SUM(painting.price) AS 'total price',
    -> AVG(painting.price) AS 'average price'
    -> FROM artist LEFT JOIN painting ON artist.a_id = painting.a_id
    -> GROUP BY artist.name;
+----------+---------------------+-------------+---------------+
| painter  | number of paintings | total price | average price |
+----------+---------------------+-------------+---------------+
| Da Vinci |                   2 |         121 |       60.5000 |
| Monet    |                   0 |           0 |          NULL |
| Picasso  |                   0 |           0 |          NULL |
| Renoir   |                   1 |          64 |       64.0000 |
| Van Gogh |                   3 |         148 |       49.3333 |
+----------+---------------------+-------------+---------------+
```

Обратите внимание на то, что `COUNT()` и `SUM()` равны нулю для художников, чьих картин у вас нет, но `AVG()` содержит `NULL`. Дело в том, что `AVG()` вычисляется как сумма, деленная на количество; если же количество равно нулю, то значение не определено. Чтобы вывести для этого случая среднее значение, равное нулю, измените запрос так, чтобы значение `AVG()` проверялось при помощи `IFNULL()`:

```
mysql> SELECT artist.name AS painter,
    -> COUNT(painting.a_id) AS 'number of paintings',
```

```
    -> SUM(painting.price) AS 'total price',
    -> IFNULL(AVG(painting.price),0) AS 'average price'
    -> FROM artist LEFT JOIN painting ON artist.a_id = painting.a_id
    -> GROUP BY artist.name;
+----------+--------------------+-------------+---------------+
| painter  | number of paintings | total price | average price |
+----------+--------------------+-------------+---------------+
| Da Vinci |                  2 |         121 |       60.5000 |
| Monet    |                  0 |           0 |             0 |
| Picasso  |                  0 |           0 |             0 |
| Renoir   |                  1 |          64 |       64.0000 |
| Van Gogh |                  3 |         148 |       49.3333 |
+----------+--------------------+-------------+---------------+
```

12.9. Заполнение пустых мест в списке с помощью соединения

Задача

Вы хотите вывести итоги для каждой из нескольких категорий, но некоторые из них не представлены в обрабатываемых данных. Следовательно, в итогах будет не хватать категорий.

Решение

Создайте справочную таблицу, в которой будет храниться полный список категорий, и формируйте итоги при помощи соединения `LEFT JOIN` между списком и таблицей, содержащей ваши данные. Тогда в результате будут присутствовать все категории, в том числе и «пустые».

Обсуждение

При выполнении суммарного запроса обычно выводятся записи только для значений, реально присутствующих в данных. Предположим, что вы хотите вывести почасовые итоги для записей таблицы `mail`, которая выглядит так:

```
mysql> SELECT * FROM mail;
+---------------------+---------+---------+---------+---------+---------+
| t                   | srcuser | srchost | dstuser | dsthost | size    |
+---------------------+---------+---------+---------+---------+---------+
| 2001-05-11 10:15:08 | barb    | saturn  | tricia  | mars    |   58274 |
| 2001-05-12 12:48:13 | tricia  | mars    | gene    | venus   |  194925 |
| 2001-05-12 15:02:49 | phil    | mars    | phil    | saturn  |    1048 |
| 2001-05-13 13:59:18 | barb    | saturn  | tricia  | venus   |     271 |
| 2001-05-14 09:31:37 | gene    | venus   | barb    | mars    |    2291 |
| 2001-05-14 11:52:17 | phil    | mars    | tricia  | saturn  |    5781 |
...
```

Чтобы определить, сколько сообщений было отправлено в течение каждого часа дня, выполните следующий запрос:

```
mysql> SELECT HOUR(t) AS hour, COUNT(HOUR(t)) AS count
    -> FROM mail GROUP BY hour;
+------+-------+
| hour | count |
+------+-------+
|    7 |     1 |
|    8 |     1 |
|    9 |     2 |
|   10 |     2 |
|   11 |     1 |
|   12 |     2 |
|   13 |     1 |
|   14 |     1 |
|   15 |     1 |
|   17 |     2 |
|   22 |     1 |
|   23 |     1 |
+------+-------+
```

Однако эти итоги не полные в том смысле, что они содержат только записи для тех часов, которые представлены в таблице `mail`. Чтобы вывести суммарные данные по всем часам дня, включая те, в течение которых сообщения не отправлялись, создайте справочную таблицу с перечнем часов:

```
mysql> CREATE TABLE ref (h INT);
mysql> INSERT INTO ref (h)
    -> VALUES(0),(1),(2),(3),(4),(5),(6),(7),(8),(9),(10),(11),
    -> (12),(13),(14),(15),(16),(17),(18),(19),(20),(21),(22),(23);
```

Затем объедините справочную таблицу и таблицу `mail`, используя `LEFT JOIN`:

```
mysql> SELECT ref.h AS hour, COUNT(HOUR(mail.t)) AS count
    -> FROM ref LEFT JOIN mail ON ref.h = HOUR(mail.t)
    -> GROUP BY hour;
+------+-------+
| hour | count |
+------+-------+
|    0 |     0 |
|    1 |     0 |
|    2 |     0 |
|    3 |     0 |
|    4 |     0 |
|    5 |     0 |
|    6 |     0 |
|    7 |     1 |
|    8 |     1 |
|    9 |     2 |
|   10 |     2 |
|   11 |     1 |
|   12 |     2 |
|   13 |     1 |
|   14 |     1 |
```

```
|   15 |     1 |
|   16 |     0 |
|   17 |     2 |
|   18 |     0 |
|   19 |     0 |
|   20 |     0 |
|   21 |     0 |
|   22 |     1 |
|   23 |     1 |
+------+-------+
```

Теперь в итогах присутствует запись для каждого часа дня. Вывод запроса `LEFT JOIN` содержит строку для каждой записи справочной таблицы вне зависимости от содержимого таблицы `mail`.

Только что рассмотренный пример использует справочную таблицу в сочетании с левым соединением для заполнения пустот в списке категорий. Несколько изменив этот запрос, мы сможем использовать справочную таблицу и для нахождения «дыр» в множестве данных, то есть чтобы определить, какие категории отсутствуют в обрабатываемых данных. Следующий запрос выводит те часы дня, в течение которых не отправлялись сообщения, с помощью инструкции `HAVING`, которая выбирает только итоговые строки с нулевым счетчиком:

```
mysql> SELECT ref.h AS hour, COUNT(HOUR(mail.t)) AS count
    -> FROM ref LEFT JOIN mail ON ref.h = HOUR(mail.t)
    -> GROUP BY hour
    -> HAVING count = 0;
+------+-------+
| hour | count |
+------+-------+
|    0 |     0 |
|    1 |     0 |
|    2 |     0 |
|    3 |     0 |
|    4 |     0 |
|    5 |     0 |
|    6 |     0 |
|   16 |     0 |
|   18 |     0 |
|   19 |     0 |
|   20 |     0 |
|   21 |     0 |
+------+-------+
```

В этом случае можно упростить запрос, используя тот факт, что каждое значение часа присутствует в справочной таблице ровно один раз. То есть инструкция `GROUP BY` не нужна; будем просто искать строки справочной таблицы, которым не соответствует никакая строка `mail`:

```
mysql> SELECT ref.h AS hour
    -> FROM ref LEFT JOIN mail ON ref.h = HOUR(mail.t)
```

```
    -> WHERE mail.t IS NULL;
+------+
| hour |
+------+
|    0 |
|    1 |
|    2 |
|    3 |
|    4 |
|    5 |
|    6 |
|   16 |
|   18 |
|   19 |
|   20 |
|   21 |
+------+
```

Запрос имеет еще одно преимущество: не выводится столбец счетчика, который в любом случае не нужен, так как все его значения равны нулю.

Справочные таблицы, содержащие перечень категорий, чрезвычайно полезны для суммарных запросов, но создание такой таблицы вручную может оказаться сложным и подверженным ошибкам мероприятием. Если список категорий содержит большой объем записей, то, вероятно, лучше написать сценарий, который использует для формирования справочной таблицы границы диапазона значений категории. По существу такие сценарии действуют как итераторы, генерирующие запись для каждого значения диапазона. Следующий сценарий на Perl, *make_date_list.pl*, иллюстрирует такой подход, создавая справочную таблицу, которая содержит строку для каждой даты определенного диапазона:

```
#! /usr/bin/perl -w
# make_date_list.pl - создание таблицы с записью для каждой даты указанного
# диапазона. Таблицу можно использовать в LEFT JOIN с таблицей данных при
# формировании итогов, чтобы убедиться в том, что в итогах присутствуют все даты,
# вне зависимости от наличия в таблице значений данных для указанного дня.

# Используются: make_date_list.pl tbl_name col_name min_date max_date

# Сценарий предполагает, что используется база данных cookbook.

use strict;
use lib qw(/usr/local/apache/lib/perl);
use Cookbook;

# Проверить количество аргументов; представлены минимальные тесты на ISO-формат дат

@ARGV == 4
    or die "Usage: make_date_list.pl tbl_name col_name min_date max_date\n";
my ($tbl_name, $col_name, $min_date, $max_date) = (@ARGV);
$min_date =~ /^\d+\D\d+\D\d+$/
    or die "Minimum date $min_date is not in ISO format\n";
```

```
$max_date =~ /^\d+\D\d+\D\d+$/
    or die "Maximum date $max_date is not in ISO format\n";

my $dbh = Cookbook::connect ();

# Определить количество дней, относящихся к диапазону дат.

my $days = $dbh->selectrow_array (qq{ SELECT TO_DAYS(?) - TO_DAYS(?) + 1 },
                                  undef, $max_date, $min_date);

print "Minimum date: $min_date\n";
print "Maximum date: $max_date\n";
print "Number of days spanned by range: $days\n";
die "Date range is too small\n" if $days < 1;

# Удалить таблицу, если она существует, затем – пересоздать.

$dbh->do ("DROP TABLE IF EXISTS $tbl_name");
$dbh->do (qq{
        CREATE TABLE $tbl_name
        ($col_name DATE NOT NULL, PRIMARY KEY ($col_name))
    });

# Заполнить таблицу записями обо всех датах диапазона.

my $sth = $dbh->prepare (qq{
        INSERT INTO $tbl_name ($col_name) VALUES(DATE_ADD(?,INTERVAL ? DAY))
    });
for (my $i = 0; $i < $days; $i++)
{
    $sth->execute ($min_date, $i);
}

$dbh->disconnect ();
exit (0);
```

Таблицы, генерируемые сценарием *make_date_list.pl*, можно использовать для итогов по дням или для поиска дней, не представленных в таблице. Справочную таблицу дат можно применять и для итогов по календарным дням. Например, можно использовать ее совместно с таблицей `master` базы данных *baseball1.com* чтобы определить количество игроков, родившихся в каждый день года, или найти дни, в которые нет ни одного дня рождения. При создании справочной таблицы календарных дней не забудьте о високосных годах, необходимо, чтобы таблица содержала для них запись о 29 февраля. 2004 год – високосный, поэтому справочную таблицу можно создать так:

```
% make_date_list.pl ref d 2004-01-01 2004-12-31
```

Таблица `master` хранит даты рождения в трех столбцах с именами `birthday`, `birthmonth`, `birthyear`. После создания справочной таблицы используйте следующий запрос для суммирования дней рождений из таблицы `master` для каждого календарного дня:

```
SELECT
MONTH(ref.d) AS month, DAYOFMONTH(ref.d) AS day,
```

```
    COUNT(master.lahmanid) AS count
    FROM ref LEFT JOIN master
        ON MONTH(ref.d) = master.birthmonth
        AND DAYOFMONTH(ref.d) = master.birthday
    GROUP BY month, day;
```

Чтобы проверить, есть ли дни, в которые ни у кого нет дня рождения, выполните:

```
    SELECT MONTH(ref.d) AS month, DAYOFMONTH(ref.d) AS day
    FROM ref LEFT JOIN master
        ON MONTH(ref.d) = master.birthmonth
        AND DAYOFMONTH(ref.d) = master.birthday
    WHERE master.birthmonth IS NULL and master.birthday IS NULL;
```

12.10. Отношение «многие-ко-многим»

Задача

Вы хотите вывести данные из таблиц, при этом записи любой таблицы могут соответствовать нескольким записям другой таблицы.

Решение

Это связь «многие-ко-многим». Необходимо использовать третью таблицу для сопоставления двух исходных, и трехстороннее соединение для вывода соотношений между ними.

Обсуждение

Использованные в предыдущих разделах таблицы `artist` и `painting` имеют связь «один-ко-многим»: определенный художник мог написать много картин, но каждая картина создана ровно одним автором. Связь «один-ко-многим» достаточно проста, и две таблицы могут быть связаны при помощи общего ключа (common key).

Еще проще связь «один-к-одному», часто применяемая для выполнения подстановок, отображающих один набор значений на другой. Например, таблица `states` содержит столбцы `name` и `abbrev`, в которых приводятся полные названия штатов и их аббревиатуры:

```
mysql> SELECT name, abbrev FROM states;
+----------------+--------+
| name           | abbrev |
+----------------+--------+
| Alabama        | AL     |
| Alaska         | AK     |
| Arizona        | AZ     |
| Arkansas       | AR     |
...
```

Это связь «один-к-одному». Ее можно использовать для сопоставления аббревиатурам названий штатов из таблицы painting, которая содержит столбец state, указывающий штат приобретения картины. Без сопоставления записи painting можно вывести так:

```
mysql> SELECT title, state FROM painting ORDER BY state;
+------------------+-------+
| title            | state |
+------------------+-------+
| The Rocks        | IA    |
| The Last Supper  | IN    |
| Starry Night     | KY    |
| The Potato Eaters | KY    |
| The Mona Lisa    | MI    |
| Les Deux Soeurs  | NE    |
+------------------+-------+
```

Если вы хотите видеть полные названия штатов, а не их аббревиатуры, то можете использовать связь «один-к-одному», существующую между столбцами таблицы states. Объедините эту таблицу с painting, используя общие для двух таблиц значения аббревиатур:

```
mysql> SELECT painting.title, states.name AS state
    -> FROM painting, states
    -> WHERE painting.state = states.abbrev
    -> ORDER BY state;
+------------------+----------+
| title            | state    |
+------------------+----------+
| The Last Supper  | Indiana  |
| The Rocks        | Iowa     |
| Starry Night     | Kentucky |
| The Potato Eaters | Kentucky |
| The Mona Lisa    | Michigan |
| Les Deux Soeurs  | Nebraska |
+------------------+----------+
```

Более сложной связью между таблицами является связь «многие-ко-многим», которая имеет место, если записи одной таблицы может соответствовать несколько записей второй, и наоборот. Обычно в книгах по базам данных такая связь иллюстрируется задачей «о деталях и поставщиках» («parts and suppliers»). (Указанную деталь можно получить от разных поставщиков. Как тогда составить список, показывающий, какие детали приходят от каких поставщиков?) Но я уже столько раз видел этот пример, что хочу привести другой. И хотя общая идея остается той же, давайте все же рассмотрим мой пример. Вы с приятелями – страстные поклонники юкера (euchre – карточная игра для четверых, в которую играют парами). Каждый год вы собираетесь вместе, делитесь на пары и организуете товарищеский турнир. Естественно, чтобы избежать споров об итогах каждого матча, вы записываете состав пар и результаты в базу данных. Можно было бы хранить результаты в таблице, куда каждый год вы записываете названия команд-участников

турнира, количества побед и поражений, имена игроков и место их проживания:

```
mysql> SELECT * FROM euchre ORDER BY year, wins DESC, player;
+----------+------+------+--------+----------+-------------+
| team     | year | wins | losses | player   | player_city |
+----------+------+------+--------+----------+-------------+
| Kings    | 2001 |   10 |      2 | Ben      | Cork        |
| Kings    | 2001 |   10 |      2 | Billy    | York        |
| Crowns   | 2001 |    7 |      5 | Melvin   | Dublin      |
| Crowns   | 2001 |    7 |      5 | Tony     | Derry       |
| Stars    | 2001 |    4 |      8 | Franklin | Bath        |
| Stars    | 2001 |    4 |      8 | Wallace  | Cardiff     |
| Sceptres | 2001 |    3 |      9 | Maurice  | Leeds       |
| Sceptres | 2001 |    3 |      9 | Nigel    | London      |
| Crowns   | 2002 |    9 |      3 | Ben      | Cork        |
| Crowns   | 2002 |    9 |      3 | Tony     | Derry       |
| Kings    | 2002 |    8 |      4 | Franklin | Bath        |
| Kings    | 2002 |    8 |      4 | Nigel    | London      |
| Stars    | 2002 |    5 |      7 | Maurice  | Leeds       |
| Stars    | 2002 |    5 |      7 | Melvin   | Dublin      |
| Sceptres | 2002 |    2 |     10 | Billy    | York        |
| Sceptres | 2002 |    2 |     10 | Wallace  | Cardiff     |
+----------+------+------+--------+----------+-------------+
```

Из таблицы видно, что в каждой команде несколько участников и каждый игрок выступал в нескольких командах. В таблице зафиксирована связь «многие-ко-многим», но она не приведена к нормальной форме. Каждая строка хранит избыточную информацию. (Данные о каждой команде записаны несколько раз, как и сведения о каждом игроке.) Лучше представить эту связь «многие-ко-многим» так:

- Хранить название каждой команды, год и победы-поражения единожды в таблице `euchre_team`.
- Хранить имя каждого игрока и город его проживания единожды в таблице `euchre_player`.
- Создать третью таблицу, `euchre_link`, в которой будут храниться соответствия игроков и команд и которая будет служить связкой (link) или мостом (bridge) между двумя исходными таблицами. Чтобы минимизировать информацию, хранящуюся в этой таблице, присвоим уникальный идентификатор каждой команде и каждому игроку в соответствующих им таблицах, и будем хранить в таблице `euchre_link` только такие идентификаторы.

Таблицы игроков и команд будут выглядеть так:

```
mysql> SELECT * FROM euchre_team;
+----+----------+------+------+--------+
| id | name     | year | wins | losses |
+----+----------+------+------+--------+
|  1 | Kings    | 2001 |   10 |      2 |
|  2 | Crowns   | 2001 |    7 |      5 |
```

```
|  3 | Stars    | 2001 |    4 |      8 |
|  4 | Sceptres | 2001 |    3 |      9 |
|  5 | Kings    | 2002 |    8 |      4 |
|  6 | Crowns   | 2002 |    9 |      3 |
|  7 | Stars    | 2002 |    5 |      7 |
|  8 | Sceptres | 2002 |    2 |     10 |
+----+----------+------+------+--------+
mysql> SELECT * FROM euchre_player;
+----+----------+---------+
| id | name     | city    |
+----+----------+---------+
|  1 | Ben      | Cork    |
|  2 | Billy    | York    |
|  3 | Tony     | Derry   |
|  4 | Melvin   | Dublin  |
|  5 | Franklin | Bath    |
|  6 | Wallace  | Cardiff |
|  7 | Nigel    | London  |
|  8 | Maurice  | Leeds   |
+----+----------+---------+
```

Таблица `euchre_link` сопоставляет игроков и команды:

```
mysql> SELECT * FROM euchre_link;
+---------+-----------+
| team_id | player_id |
+---------+-----------+
|       1 |         1 |
|       1 |         2 |
|       2 |         3 |
|       2 |         4 |
|       3 |         5 |
|       3 |         6 |
|       4 |         7 |
|       4 |         8 |
|       5 |         5 |
|       5 |         7 |
|       6 |         1 |
|       6 |         3 |
|       7 |         4 |
|       7 |         8 |
|       8 |         2 |
|       8 |         6 |
+---------+-----------+
```

Чтобы отвечать на вопросы о командах игроков с помощью созданных таблиц, нужно выполнить трехстороннее соединение, используя таблицу `euchre_link` для связывания двух базовых таблиц друг с другом. Рассмотрим несколько примеров:

- Выведем все пары, включающие сведения о командах и их участниках. Этот запрос перечисляет все соответствия между таблицами `euchre_team`

и euchre_player и воспроизводит информацию, которая изначально содержалась в ненормализованной таблице euchre:

```
mysql> SELECT t.name, t.year, t.wins, t.losses, p.name, p.city
    -> FROM euchre_team AS t, euchre_link AS l, euchre_player AS p
    -> WHERE t.id = l.team_id AND p.id = l.player_id
    -> ORDER BY t.year, t.wins DESC, p.name;
+----------+------+------+--------+----------+---------+
| name     | year | wins | losses | name     | city    |
+----------+------+------+--------+----------+---------+
| Kings    | 2001 |   10 |      2 | Ben      | Cork    |
| Kings    | 2001 |   10 |      2 | Billy    | York    |
| Crowns   | 2001 |    7 |      5 | Melvin   | Dublin  |
| Crowns   | 2001 |    7 |      5 | Tony     | Derry   |
| Stars    | 2001 |    4 |      8 | Franklin | Bath    |
| Stars    | 2001 |    4 |      8 | Wallace  | Cardiff |
| Sceptres | 2001 |    3 |      9 | Maurice  | Leeds   |
| Sceptres | 2001 |    3 |      9 | Nigel    | London  |
| Crowns   | 2002 |    9 |      3 | Ben      | Cork    |
| Crowns   | 2002 |    9 |      3 | Tony     | Derry   |
| Kings    | 2002 |    8 |      4 | Franklin | Bath    |
| Kings    | 2002 |    8 |      4 | Nigel    | London  |
| Stars    | 2002 |    5 |      7 | Maurice  | Leeds   |
| Stars    | 2002 |    5 |      7 | Melvin   | Dublin  |
| Sceptres | 2002 |    2 |     10 | Billy    | York    |
| Sceptres | 2002 |    2 |     10 | Wallace  | Cardiff |
+----------+------+------+--------+----------+---------+
```

- Выведем членов определенной команды (2001 год, Crowns):

```
mysql> SELECT p.name, p.city
    -> FROM euchre_team AS t, euchre_link AS l, euchre_player AS p
    -> WHERE t.id = l.team_id AND p.id = l.player_id
    -> AND t.name = 'Crowns' AND t.year = 2001;
+--------+--------+
| name   | city   |
+--------+--------+
| Tony   | Derry  |
| Melvin | Dublin |
+--------+--------+
```

- Выведем команды, за которые выступал определенный игрок (Billy):

```
mysql> SELECT t.name, t.year, t.wins, t.losses
    -> FROM euchre_team AS t, euchre_link AS l, euchre_player AS p
    -> WHERE t.id = l.team_id AND p.id = l.player_id
    -> AND p.name = 'Billy';
+----------+------+------+--------+
| name     | year | wins | losses |
+----------+------+------+--------+
| Kings    | 2001 |   10 |      2 |
| Sceptres | 2002 |    2 |     10 |
+----------+------+------+--------+
```

Обратите внимание на то, что для ответа на вопросы о связях «многие-ко-многим» необходимо использовать трехстороннее соединение, но это не означает, что трехстороннее соединение само по себе подразумевает связь «многие-ко-многим». Ранее в разделе мы объединяли таблицу `states` с таблицей `painting` для сопоставления аббревиатурам названий штатов их полных названий:

```
mysql> SELECT painting.title, states.name AS state
    -> FROM painting, states
    -> WHERE painting.state = states.abbrev
    -> ORDER BY state;
+-------------------+----------+
| title             | state    |
+-------------------+----------+
| The Last Supper   | Indiana  |
| The Rocks         | Iowa     |
| Starry Night      | Kentucky |
| The Potato Eaters | Kentucky |
| The Mona Lisa     | Michigan |
| Les Deux Soeurs   | Nebraska |
+-------------------+----------+
```

Чтобы вывести фамилию художника, написавшего каждую из картин, слегка изменим запрос, объединив результаты с таблицей `artist`:

```
mysql> SELECT artist.name, painting.title, states.name AS state
    -> FROM artist, painting, states
    -> WHERE artist.a_id = painting.a_id AND painting.state = states.abbrev;
+----------+-------------------+----------+
| name     | title             | state    |
+----------+-------------------+----------+
| Da Vinci | The Last Supper   | Indiana  |
| Da Vinci | The Mona Lisa     | Michigan |
| Van Gogh | Starry Night      | Kentucky |
| Van Gogh | The Potato Eaters | Kentucky |
| Van Gogh | The Rocks         | Iowa     |
| Renoir   | Les Deux Soeurs   | Nebraska |
+----------+-------------------+----------+
```

Теперь запрос включает трехстороннее соединение, хотя природа связи между авторами и картинами не изменилась: это связь «один-ко-многим», а не «многие-ко-многим».

12.11. Сравнение таблицы с самой собой

Задача

Вы хотите сравнить записи таблицы с другими записями той же самой таблицы. Например, вы хотите найти в вашей коллекции все картины автора, написавшего «Едоки картофеля». Или хотите узнать, какие штаты из таблицы `states` вступили в Союз в том же году, что и Нью-Йорк (New York).

Или хотите узнать, кто из людей, перечисленных в таблице `profile`, любит одни и те же блюда.

Решение

Задачи, требующие соединения таблицы с ней же самой, решаются с помощью операции самосоединения (self-join). Она во многом похожа на другие соединения, только вы должны всегда использовать псевдонимы таблиц, чтобы по-разному ссылаться на одну и ту же таблицу в запросе.

Обсуждение

Специальным случаем соединения двух таблиц является соединение таблицы с самой собой. Такая операция называется самосоединением. Хотя многим подобная идея может показаться странной, на самом деле она абсолютно законна. После того, как вы свыкнитесь с этой идеей, более чем вероятно, что вы начнете использовать самосоединения достаточно часто, поскольку они очень полезны.

Указанием на необходимость самосоединения является вопрос о парах элементов таблицы, удовлетворяющих некоторым условиям. Например, предположим, что ваше любимая картина – это «Едоки картофеля» («The Potato Eaters»), и вы хотите определить все остальные экспонаты вашей коллекции, созданные тем же автором. Вы можете поступить так:

1. Идентифицируйте строку таблицы `painting`, содержащую название «The Potato Eaters», чтобы можно было ссылаться на ее значение `a_id`.
2. Используйте значение `a_id` для выявления других строк таблицы, имеющих такое же значение `a_id`.
3. Выведите названия картин из найденных строк.

Идентификаторы художников и названия картин, с которых мы начинаем работу, выглядят так:

```
mysql> SELECT a_id, title FROM painting ORDER BY a_id;
+------+------------------+
| a_id | title            |
+------+------------------+
|    1 | The Last Supper  |
|    1 | The Mona Lisa    |
|    3 | Starry Night     |
|    3 | The Potato Eaters |
|    3 | The Rocks        |
|    5 | Les Deux Soeurs  |
+------+------------------+
```

Двухэтапный способ извлечения нужных названий без применения соединения заключается в поиске идентификатора художника в одном запросе и использовании этого идентификатора во втором запросе для выбора соответствующих ему записей:

```
mysql> SELECT @id := a_id FROM painting WHERE title = 'The Potato Eaters';
+-------------+
| @id := a_id |
+-------------+
|           3 |
+-------------+
mysql> SELECT title FROM painting WHERE a_id = @id;
+-------------------+
| title             |
+-------------------+
| Starry Night      |
| The Potato Eaters |
| The Rocks         |
+-------------------+
```

Другое решение, требующее выполнения всего одного запроса, состоит в применении самосоединения. Обратите особое внимание на использование правильной нотации. Многие поначалу пытаются написать запрос, соединяющий таблицу с ней самой, так:

```
mysql> SELECT title FROM painting, painting
    -> WHERE title = 'The Potato Eaters' AND a_id = a_id;
ERROR 1066 at line 1: Not unique table/alias: 'painting'
```

Проблема такого запроса в том, что ссылки на столбцы неоднозначны. MySQL не может определить, на какой экземпляр таблицы `painting` ссылается каждый из столбцов. Решение состоит в присвоении хотя бы одному экземпляру таблицы псевдонима, чтобы можно было отличать столбцы с помощью спецификаторов таблиц. Покажем, как это можно сделать, используя псевдонимы `p1` и `p2` для ссылок на таблицы `painting`:

```
mysql> SELECT p2.title
    -> FROM painting AS p1, painting AS p2
    -> WHERE p1.title = 'The Potato Eaters'
    -> AND p1.a_id = p2.a_id;
+-------------------+
| title             |
+-------------------+
| Starry Night      |
| The Potato Eaters |
| The Rocks         |
+-------------------+
```

Результат запроса типичен для самосоединения: если вы используете некоторое значение в одном экземпляре таблицы («The Potato Eaters») для нахождения соответствующих записей в другом экземпляре (картины того же художника), то вывод содержит и первоначальное значение. В этом есть смысл – в конце концов, значение соответствует самому себе. Если же вы хотите найти только *другие* картины того же автора, необходимо явно исключить исходное значение из вывода:

```
mysql> SELECT p2.title
```

```
    -> FROM painting AS p1, painting AS p2
    -> WHERE p1.title = 'The Potato Eaters' AND p2.title != 'The Potato Eaters'
    -> AND p1.a_id = p2.a_id;
+--------------+
| title        |
+--------------+
| Starry Night |
| The Rocks    |
+--------------+
```

Более общий способ исключения исходного значения – указать, что вы не хотите, чтобы значение в выводимой строке совпадало с исходным:

```
mysql> SELECT p2.title
    -> FROM painting AS p1, painting AS p2
    -> WHERE p1.title = 'The Potato Eaters' AND p1.title != p2.title
    -> AND p1.a_id = p2.a_id;
+--------------+
| title        |
+--------------+
| Starry Night |
| The Rocks    |
+--------------+
```

Рассмотренные запросы использовали значения идентификаторов для сопоставления записей двух экземпляров таблиц, но можно использовать и любые другие типы значений. Например, чтобы при помощи таблицы `states` ответить на вопрос: «Какие штаты вступили в Союз в том же году, что и Нью-Йорк?», будем выполнять попарное сравнение времен, используя составляющую года даты из столбца `statehood`:

```
mysql> SELECT s2.name, s2.statehood
    -> FROM states AS s1, states AS s2
    -> WHERE s1.name = 'New York'
    -> AND YEAR(s1.statehood) = YEAR(s2.statehood)
    -> ORDER BY s2.name;
+----------------+------------+
| name           | statehood  |
+----------------+------------+
| Connecticut    | 1788-01-09 |
| Georgia        | 1788-01-02 |
| Maryland       | 1788-04-28 |
| Massachusetts  | 1788-02-06 |
| New Hampshire  | 1788-06-21 |
| New York       | 1788-07-26 |
| South Carolina | 1788-05-23 |
| Virginia       | 1788-06-25 |
+----------------+------------+
```

Исходное значение (New York) снова попадает в вывод. Если вы хотите изменить ситуацию, добавьте в инструкцию `WHERE` выражение, явно исключающее такое значение:

```
mysql> SELECT s2.name, s2.statehood
    -> FROM states AS s1, states AS s2
    -> WHERE s1.name = 'New York' AND s1.name != s2.name
    -> AND YEAR(s1.statehood) = YEAR(s2.statehood)
    -> ORDER BY s2.name;
+----------------+------------+
| name           | statehood  |
+----------------+------------+
| Connecticut    | 1788-01-09 |
| Georgia        | 1788-01-02 |
| Maryland       | 1788-04-28 |
| Massachusetts  | 1788-02-06 |
| New Hampshire  | 1788-06-21 |
| South Carolina | 1788-05-23 |
| Virginia       | 1788-06-25 |
+----------------+------------+
```

Как и задача нахождения других картин художника, написавшего «Едоков картофеля», задача о государственности штатов может быть решена с помощью переменной SQL и двух запросов. Это верно для любых поисков соответствия конкретной строке таблицы. Но бывают задачи, требующие установления соответствия нескольких пар строк, для которых двухэтапный метод не работает. Предположим, что вы хотите определить, какие пары людей, перечисленных в таблице `profile`, любят одинаковые блюда. В этом случае вывод теоретически может содержать любую пару из таблицы. Фиксированного исходного значения нет, поэтому его невозможно сохранить в переменной.

Для решения такой задачи отлично подходит самосоединение, хотя и предстоит решить вопрос о том, как определить, какие значения `foods` используют общие элементы. Столбец `foods` содержит значения типа `SET`, каждое из которых может указывать несколько любимых блюд, поэтому простого сравнения недостаточно:

- Результат сравнения истинен, только если оба значения `foods` указывают одинаковые наборы блюд, а нам достаточно совпадения элементов.
- Два пустых значения воспринимаются сравнением как равные, хотя на самом деле им не соответствуют общие блюда.

Чтобы выявить значения `SET`, имеющие общие элементы, используем тот факт, что MySQL представляет их как битовые поля и выполняет сравнение при помощи оператора `&` (побитовое «И»), ища пары, имеющие ненулевое пересечение:

```
mysql> SELECT t1.name, t2.name, t1.foods, t2.foods
    -> FROM profile AS t1, profile AS t2
    -> WHERE t1.id != t2.id AND (t1.foods & t2.foods) != 0
    -> ORDER BY t1.name, t2.name;
+------+------+---------------------+---------------------+
| name | name | foods               | foods               |
+------+------+---------------------+---------------------+
| Alan | Brit | curry,fadge         | burrito,curry,pizza |
```

```
| Alan | Fred | curry,fadge          | lutefisk,fadge,pizza |
| Alan | Mara | curry,fadge          | lutefisk,fadge       |
| Alan | Sean | curry,fadge          | burrito,curry        |
| Brit | Alan | burrito,curry,pizza  | curry,fadge          |
| Brit | Carl | burrito,curry,pizza  | eggroll,pizza        |
| Brit | Fred | burrito,curry,pizza  | lutefisk,fadge,pizza |
| Brit | Sean | burrito,curry,pizza  | burrito,curry        |
| Carl | Brit | eggroll,pizza        | burrito,curry,pizza  |
| Carl | Fred | eggroll,pizza        | lutefisk,fadge,pizza |
| Fred | Alan | lutefisk,fadge,pizza | curry,fadge          |
| Fred | Brit | lutefisk,fadge,pizza | burrito,curry,pizza  |
| Fred | Carl | lutefisk,fadge,pizza | eggroll,pizza        |
| Fred | Mara | lutefisk,fadge,pizza | lutefisk,fadge       |
| Mara | Alan | lutefisk,fadge       | curry,fadge          |
| Mara | Fred | lutefisk,fadge       | lutefisk,fadge,pizza |
| Sean | Alan | burrito,curry        | curry,fadge          |
| Sean | Brit | burrito,curry        | burrito,curry,pizza  |
+------+------+----------------------+----------------------+
```

Некоторые задачи с участием самосоединений относятся к разновидности: «Для каких значений нет соответствий?». Примером такого вопроса может быть: «Кто из отправителей сообщений, указанных в таблице `mail`, ни разу не писал самому себе?». Сначала определим, кто кому отправлял сообщения:

```
mysql> SELECT DISTINCT srcuser, dstuser FROM mail
    -> ORDER BY srcuser, dstuser;
+---------+---------+
| srcuser | dstuser |
+---------+---------+
| barb    | barb    |
| barb    | tricia  |
| gene    | barb    |
| gene    | gene    |
| gene    | tricia  |
| phil    | barb    |
| phil    | phil    |
| phil    | tricia  |
| tricia  | gene    |
| tricia  | phil    |
+---------+---------+
```

Некоторые из этих пар показывают, что люди писали сами себе:

```
mysql> SELECT DISTINCT srcuser, dstuser FROM mail
    -> WHERE srcuser = dstuser;
+---------+---------+
| srcuser | dstuser |
+---------+---------+
| phil    | phil    |
| barb    | barb    |
| gene    | gene    |
+---------+---------+
```

Нахождение людей, которые не отправляют письма самим себе, – это задача отсутствия соответствий, для решения которой обычно требуется использовать LEFT JOIN. В данном случае необходимо выполнить левое соединение таблицы mail с ней самой:

```
mysql> SELECT DISTINCT m1.srcuser
    -> FROM mail AS m1 LEFT JOIN mail AS m2
    -> ON m1.srcuser = m2.srcuser AND m2.srcuser = m2.dstuser
    -> WHERE m2.dstuser IS NULL;
+---------+
| srcuser |
+---------+
| tricia  |
+---------+
```

Для каждой записи из таблицы mail запрос подбирает соответствия – совпадающих отправителя и получателя. Для записей, которым не найдено такого соответствия, LEFT JOIN включает в вывод строку, все столбцы m2 которой установлены в NULL. Такие строки определяют отправителей, которые не посылали себе писем.

Используя LEFT JOIN для соединения таблицы с ней самой, можно еще одним способом ответить на вопросы о нахождении максимума в группе, обсуждаемые в рецепте 12.6, без использования второй временной таблицы. В том рецепте мы находили самую дорогую картину каждого художника, используя временную таблицу:

```
mysql> CREATE TABLE tmp
    -> SELECT a_id, MAX(price) AS max_price FROM painting GROUP BY a_id;
mysql> SELECT artist.name, painting.title, painting.price
    -> FROM artist, painting, tmp
    -> WHERE painting.a_id = tmp.a_id
    -> AND painting.price = tmp.max_price
    -> AND painting.a_id = artist.a_id;
+----------+------------------+-------+
| name     | title            | price |
+----------+------------------+-------+
| Da Vinci | The Mona Lisa    |    87 |
| Van Gogh | The Potato Eaters |    67 |
| Renoir   | Les Deux Soeurs  |    64 |
+----------+------------------+-------+
```

А можно поступить по-другому, распознавая картины и извлекая значения из каждой такой строки при помощи LEFT JOIN. Следующий запрос идентифицирует картины:

```
mysql> SELECT p1.a_id, p1.title, p1.price
    -> FROM painting p1
    -> LEFT JOIN painting p2
    -> ON p1.a_id = p2.a_id AND p1.price < p2.price
    -> WHERE p2.a_id IS NULL;
```

```
+------+------------------+-------+
| a_id | title            | price |
+------+------------------+-------+
|    1 | The Mona Lisa    |    87 |
|    3 | The Potato Eaters |    67 |
|    5 | Les Deux Soeurs  |    64 |
+------+------------------+-------+
```

Для вывода фамилий художников результат объединяется с таблицей `artist`:

```
mysql> SELECT artist.name, p1.title, p1.price
    -> FROM (painting p1
    -> LEFT JOIN painting p2
    -> ON p1.a_id = p2.a_id AND p1.price < p2.price), artist
    -> WHERE p2.a_id IS NULL AND p1.a_id = artist.a_id;
+----------+------------------+-------+
| name     | title            | price |
+----------+------------------+-------+
| Da Vinci | The Mona Lisa    |    87 |
| Van Gogh | The Potato Eaters |    67 |
| Renoir   | Les Deux Soeurs  |    64 |
+----------+------------------+-------+
```

Знайте, что задача сравнения таблицы с ней самой не обязательно требует использования самосоединения, даже если можно решить ее таким способом. Рассмотрим, например, таблицу `mail`. Можно определить, кто посылал письма самому себе, используя самосоединение:

```
mysql> SELECT DISTINCT m1.srcuser, m2.dstuser
    -> FROM mail AS m1, mail AS m2
    -> WHERE m1.srcuser = m2.srcuser AND m2.dstuser = m1.srcuser;
+---------+---------+
| srcuser | dstuser |
+---------+---------+
| phil    | phil    |
| barb    | barb    |
| gene    | gene    |
+---------+---------+
```

Но это неразумно. Запросу совсем не нужно сравнивать записи со всеми остальными. Достаточно просто сравнить различные столбцы внутри каждой строки, поэтому вполне можно выполнить обычный запрос без соединения:

```
mysql> SELECT DISTINCT srcuser, dstuser FROM mail
    -> WHERE srcuser = dstuser;
+---------+---------+
| srcuser | dstuser |
+---------+---------+
| phil    | phil    |
| barb    | barb    |
| gene    | gene    |
+---------+---------+
```

12.12. Вычисление разности между последовательными строками

Задача

У вас есть таблица, строки которой содержат последовательные накапливаемые значения, и вы хотите вычислить разности между парами последовательных строк.

Решение

Используйте самосоединение, сопоставляющее пары соседних строк и вычисляющее разность между членами пары.

Обсуждение

Самосоединения полезны, если у вас есть набор абсолютных (или накопленных) значений, которые вы хотите преобразовать в относительные значения, представляющие разности последовательных пар строк. Например, если вы едете в путешествие на автомобиле и записываете количество миль на каждой остановке, то можете вычислить разность расстояний до последовательных точек, чтобы определить расстояние между ними. Рассмотрим таблицу, предствляющую остановки на пути из Сан-Антонио, штат Техас, в Мэдисон, штат Висконсин. Каждая строка содержит общее количество миль, пройденных до данной остановки:

```
mysql> SELECT seq, city, miles FROM trip_log ORDER BY seq;
+-----+------------------+-------+
| seq | city             | miles |
+-----+------------------+-------+
|   1 | San Antonio, TX  |     0 |
|   2 | Dallas, TX       |   263 |
|   3 | Benton, AR       |   566 |
|   4 | Memphis, TN      |   745 |
|   5 | Portageville, MO |   878 |
|   6 | Champaign, IL    |  1164 |
|   7 | Madison, WI      |  1412 |
+-----+------------------+-------+
```

Самосоединение может преобразовать эти накапливаемые значения в последовательные разности, представляющие расстояния от каждого города до следующего. Напишем запрос, который использует номера последовательности в записях для нахождения пар соседних строк и вычисления разностей для каждой пары значений расстояний:

```
mysql> SELECT t1.seq AS seq1,  t2.seq AS seq2,
    -> t1.city AS city1, t2.city AS city2,
    -> t1.miles AS miles1, t2.miles AS miles2,
    -> t2.miles-t1.miles AS dist
    -> FROM trip_log AS t1, trip_log AS t2
```

```
    -> WHERE t1.seq+1 = t2.seq
    -> ORDER BY t1.seq;
+------+------+------------------+------------------+--------+--------+------+
| seq1 | seq2 | city1            | city2            | miles1 | miles2 | dist |
+------+------+------------------+------------------+--------+--------+------+
|    1 |    2 | San Antonio, TX  | Dallas, TX       |      0 |    263 |  263 |
|    2 |    3 | Dallas, TX       | Benton, AR       |    263 |    566 |  303 |
|    3 |    4 | Benton, AR       | Memphis, TN      |    566 |    745 |  179 |
|    4 |    5 | Memphis, TN      | Portageville, MO |    745 |    878 |  133 |
|    5 |    6 | Portageville, MO | Champaign, IL    |    878 |   1164 |  286 |
|    6 |    7 | Champaign, IL    | Madison, WI      |   1164 |   1412 |  248 |
+------+------+------------------+------------------+--------+--------+------+
```

Наличие столбца `seq` в таблице `trip_log` необходимо для вычисления последовательных разностей: с его помощью устанавливается, какая строка предшествует другой строке, и выполняется сопоставление строки с номером *n* строке *n+1*. При выполнении вычислений разностей абсолютных (или накапливаемых) значений подразумевается, что таблица должна содержать столбец последовательности без разрывов (gap). Если таблица содержит столбец последовательности с разрывами, перенумеруйте его. Если в таблице вообще нет такого столбца, добавьте его. Выполнение этих операций описано в рецептах 11.8 и 11.12.

Более сложная ситуация возникает, если нужно вычислить последовательные разности для нескольких столбцов, а затем использовать результаты в вычислении. Таблица `player_stats` предлагает несколько суммарных показателей для бейсбольного игрока в конце каждого месяца игрового сезона: `ab` означает общее количество выступлений в качестве отбивающего (at-bat), а `h` – общее количество отбитых подач (hit) на указанную дату. (Первая запись относится к началу сезона игрока, поэтому значения `ab` и `h` равны нулю.)

```
mysql> SELECT id, date, ab, h, TRUNCATE(IFNULL(h/ab,0),3) AS ba
    -> FROM player_stats ORDER BY id;
+----+------------+-----+----+-------+
| id | date       | ab  | h  | ba    |
+----+------------+-----+----+-------+
|  1 | 2001-04-30 |   0 |  0 | 0.000 |
|  2 | 2001-05-31 |  38 | 13 | 0.342 |
|  3 | 2001-06-30 | 109 | 31 | 0.284 |
|  4 | 2001-07-31 | 196 | 49 | 0.250 |
|  5 | 2001-08-31 | 304 | 98 | 0.322 |
+----+------------+-----+----+-------+
```

Последний столбец результата запроса приводит среднее число очков отбивающего игрока (batting average) на каждую дату. Этот столбец не хранится в таблице, но легко вычисляется как отношение числа отбитых подач к числу выступлений в качестве отбивающего. Результат дает общее представление об изменении частоты попаданий игрока в течение сезона, но при этом не очень понятно, как она менялась от месяца к месяцу. Необходимо вычислить относительные разности между парами строк. Это легко сделать при помощи самосоединения, которое сопоставляет каждой строке *n* строку *n*+1 для вы-

числения разностей между парами поданных и отбитых подач. Эти разности позволяют вычислить среднее число очков отбивающего за каждый месяц:

```
mysql> SELECT
    -> t1.id AS id1, t2.id AS id2,
    -> t2.date,
    -> t1.ab AS ab1, t2.ab AS ab2,
    -> t1.h AS h1, t2.h AS h2,
    -> t2.ab-t1.ab AS abdiff,
    -> t2.h-t1.h AS hdiff,
    -> TRUNCATE(IFNULL((t2.h-t1.h)/(t2.ab-t1.ab),0),3) AS ba
    -> FROM player_stats AS t1, player_stats AS t2
    -> WHERE t1.id+1 = t2.id
    -> ORDER BY t1.id;
+-----+-----+------------+-----+-----+----+----+--------+-------+-------+
| id1 | id2 | date       | ab1 | ab2 | h1 | h2 | abdiff | hdiff | ba    |
+-----+-----+------------+-----+-----+----+----+--------+-------+-------+
|   1 |   2 | 2001-05-31 |   0 |  38 |  0 | 13 |     38 |    13 | 0.342 |
|   2 |   3 | 2001-06-30 |  38 | 109 | 13 | 31 |     71 |    18 | 0.253 |
|   3 |   4 | 2001-07-31 | 109 | 196 | 31 | 49 |     87 |    18 | 0.206 |
|   4 |   5 | 2001-08-31 | 196 | 304 | 49 | 98 |    108 |    49 | 0.453 |
+-----+-----+------------+-----+-----+----+----+--------+-------+-------+
```

Результаты гораздо нагляднее, чем исходная таблица, показывают, что игрок начал сезон хорошо, потом, особенно в июле, у него наблюдался резкий спад. А вот в августе его результаты стали просто замечательными.

12.13. Нарастающий итог и скользящее среднее

Задача

У вас есть ряд наблюдений, сделанных на протяжении какого-то периода времени, и вы хотите вычислить сумму нарастающим итогом (cumulative sum) для каждой точки измерения. Или же хотите получить скользящее среднее (running average) в каждой точке.

Решение

Используйте самосоединение для формирования набора последовательных наблюдений в каждой точке измерения, затем примените агрегирующую функцию для вычисления суммы или среднего значения каждого набора значений.

Обсуждение

В рецепте 12.12 было показано, как самосоединение может формировать относительные значения из абсолютных. Самосоединение может делать и обратное, формируя накапливаемые значения для каждого последовательного этапа серии наблюдений. В следующей таблице приведены результаты измерений количества осадков, сделанных в течение нескольких дней. Значения каждой строки представляют дату наблюдения и количество осадков в дюймах:

```
mysql> SELECT date, precip FROM rainfall ORDER BY date;
+------------+--------+
| date       | precip |
+------------+--------+
| 2002-06-01 |   1.50 |
| 2002-06-02 |   0.00 |
| 2002-06-03 |   0.50 |
| 2002-06-04 |   0.00 |
| 2002-06-05 |   1.00 |
+------------+--------+
```

Чтобы вычислить совокупное количество осадков на указанный день, сложим значение количества осадков в данный день со значениями всех предыдущих дней. Например, совокупное количество осадков на `2002-06-03` определяется так:

```
mysql> SELECT SUM(precip) FROM rainfall WHERE date <= '2002-06-03';
+-------------+
| SUM(precip) |
+-------------+
|        2.00 |
+-------------+
```

Если вы хотите получить совокупные значения для всех дней, представленных в таблице, то подсчет всех этих значений по отдельности будет весьма трудоемким. Самосоединение может получить совокупные значения для всех дней в одном запросе. Используйте один экземпляр таблицы `rainfall` как справочную таблицу и определите для даты каждой строки сумму значений `precip` для всех строк с датами, вплоть до данной даты в другом экземпляре таблицы. Следующий запрос выводит дневное и совокупное количества осадков для каждого дня таблицы:

```
mysql> SELECT t1.date, t1.precip AS 'daily precip',
    -> SUM(t2.precip) AS 'cum. precip'
    -> FROM rainfall AS t1, rainfall AS t2
    -> WHERE t1.date >= t2.date
    -> GROUP BY t1.date;
+------------+--------------+-------------+
| date       | daily precip | cum. precip |
+------------+--------------+-------------+
| 2002-06-01 |         1.50 |        1.50 |
| 2002-06-02 |         0.00 |        1.50 |
| 2002-06-03 |         0.50 |        2.00 |
| 2002-06-04 |         0.00 |        2.00 |
| 2002-06-05 |         1.00 |        3.00 |
+------------+--------------+-------------+
```

Самосоединение можно расширить так, чтобы оно выводило количество дней, прошедших до указанной даты, и вычисляло скользящее среднее для количества осадков за каждый день:

```
mysql> SELECT t1.date, t1.precip AS 'daily precip',
    -> SUM(t2.precip) AS 'cum. precip',
```

```
    -> COUNT(t2.precip) AS days,
    -> AVG(t2.precip) AS 'avg. precip'
    -> FROM rainfall AS t1, rainfall AS t2
    -> WHERE t1.date >= t2.date
    -> GROUP BY t1.date;
+------------+--------------+-------------+------+-------------+
| date       | daily precip | cum. precip | days | avg. precip |
+------------+--------------+-------------+------+-------------+
| 2002-06-01 |         1.50 |        1.50 |    1 |    1.500000 |
| 2002-06-02 |         0.00 |        1.50 |    2 |    0.750000 |
| 2002-06-03 |         0.50 |        2.00 |    3 |    0.666667 |
| 2002-06-04 |         0.00 |        2.00 |    4 |    0.500000 |
| 2002-06-05 |         1.00 |        3.00 |    5 |    0.600000 |
+------------+--------------+-------------+------+-------------+
```

В предыдущем запросе количество прошедших дней и средний объем осадков можно легко получить при помощи `COUNT()` и `AVG()`, так как в таблице нет пропущенных дней. Если же было бы разрешено пропускать дни, вычисление усложнилось бы, так как в этом случае количество дней, прошедших до указанной даты, не было бы равно количеству записей. Давайте удалим из таблицы записи для тех дней, в которые не было осадков, создав тем самым две «дыры»:

```
mysql> DELETE FROM rainfall WHERE precip = 0;
mysql> SELECT date, precip FROM rainfall ORDER BY date;
+------------+--------+
| date       | precip |
+------------+--------+
| 2002-06-01 |   1.50 |
| 2002-06-03 |   0.50 |
| 2002-06-05 |   1.00 |
+------------+--------+
```

Удаление таких записей не изменяет накапливаемую сумму и скользящее среднее, но изменяет способ их вычисления. Если попытаться опять выполнить самосоединение, то результат для количества прошедших дней и среднего количества осадков будет неверным:

```
mysql> SELECT t1.date, t1.precip AS 'daily precip',
    -> SUM(t2.precip) AS 'cum. precip',
    -> COUNT(t2.precip) AS days,
    -> AVG(t2.precip) AS 'avg. precip'
    -> FROM rainfall AS t1, rainfall AS t2
    -> WHERE t1.date >= t2.date
    -> GROUP BY t1.date;
+------------+--------------+-------------+------+-------------+
| date       | daily precip | cum. precip | days | avg. precip |
+------------+--------------+-------------+------+-------------+
| 2002-06-01 |         1.50 |        1.50 |    1 |    1.500000 |
| 2002-06-03 |         0.50 |        2.00 |    2 |    1.000000 |
| 2002-06-05 |         1.00 |        3.00 |    3 |    1.000000 |
+------------+--------------+-------------+------+-------------+
```

Чтобы исправить ошибку, необходимо определять количество дней другим способом. Возьмем минимальную и максимальную даты, участвующие в расчетах каждой суммы, и вычислим количество прошедших дней, используя следующее выражение:

```
TO_DAYS(MAX(t2.date)) - TO_DAYS(MIN(t2.date)) + 1
```

Это значение должно применяться для столбца количества дней и для вычисления среднего. В результате запрос будет таким:

```
mysql> SELECT t1.date, t1.precip AS 'daily precip',
    -> SUM(t2.precip) AS 'cum. precip',
    -> TO_DAYS(MAX(t2.date)) - TO_DAYS(MIN(t2.date)) + 1 AS days,
    -> SUM(t2.precip) / (TO_DAYS(MAX(t2.date)) - TO_DAYS(MIN(t2.date)) + 1)
    -> AS 'avg. precip'
    -> FROM rainfall AS t1, rainfall AS t2
    -> WHERE t1.date >= t2.date
    -> GROUP BY t1.date;
+------------+--------------+-------------+------+-------------+
| date       | daily precip | cum. precip | days | avg. precip |
+------------+--------------+-------------+------+-------------+
| 2002-06-01 |         1.50 |        1.50 |    1 |      1.5000 |
| 2002-06-03 |         0.50 |        2.00 |    3 |      0.6667 |
| 2002-06-05 |         1.00 |        3.00 |    5 |      0.6000 |
+------------+--------------+-------------+------+-------------+
```

Как видно из примера, для вычисления накопленных значений по относительным необходим лишь столбец, позволяющий расположить строки в правильном порядке. (В таблице `rainfall` это столбец `date`.) Значения такого столбца не должны быть последовательными и даже не обязаны быть числами. В этом отличие от вычисления разностей значений для накопленных значений (см. рецепт 12.12), которое требует наличия в таблице столбца с непрерывной последовательностью.

Скользящие средние в примере про осадки вычисляются делением суммы количества осадков за определенное количество дней на количество дней. Если в таблице нет разрывов, количество дней соответствует количеству суммируемых записей, поэтому находить последовательные средние значения очень просто. Если же какие-то записи отсутствуют, вычисления усложняются. То есть необходимо внимательно исследовать данные и выбрать подходящий тип вычисления средних. Следующий пример похож на предыдущие тем, что он тоже вычисляет суммы нарастающим итогом и скользящее среднее, но вычисления он выполняет по-другому.

Следующая таблица приводит характеристики марафонца на каждом этапе 26-километровой дистанции. Значения каждой строки показывают длину этапа в километрах и время, затраченное бегуном на его преодоление. Другими словами, величины относятся к интервалам внутри марафона, то есть являются относительными:

```
mysql> SELECT stage, km, t FROM marathon ORDER BY stage;
```

```
+-------+----+----------+
| stage | km | t        |
+-------+----+----------+
|     1 |  5 | 00:15:00 |
|     2 |  7 | 00:19:30 |
|     3 |  9 | 00:29:20 |
|     4 |  5 | 00:17:50 |
+-------+----+----------+
```

Чтобы вычислить накапливаемые расстояния в километрах для каждого этапа, выполним самосоединение:

```
mysql> SELECT t1.stage, t1.km, SUM(t2.km) AS 'cum. km'
    -> FROM marathon AS t1, marathon AS t2
    -> WHERE t1.stage >= t2.stage
    -> GROUP BY t1.stage;
+-------+----+---------+
| stage | km | cum. km |
+-------+----+---------+
|     1 |  5 |       5 |
|     2 |  7 |      12 |
|     3 |  9 |      21 |
|     4 |  5 |      26 |
+-------+----+---------+
```

Получить накопленные расстояния легко, так как их можно просто складывать. Накопление времени чуть сложнее. Необходимо преобразовать время в секунды, суммировать полученные значения, затем преобразовать сумму обратно в значение времени. Чтобы вычислить среднюю скорость бегуна на каждом этапе, вычислим отношение накопленного расстояния к накопленному времени. Теперь соберем все вместе и напишем такой запрос:

```
mysql> SELECT t1.stage, t1.km, t1.t,
    -> SUM(t2.km) AS 'cum. km',
    -> SEC_TO_TIME(SUM(TIME_TO_SEC(t2.t))) AS 'cum. t',
    -> SUM(t2.km)/(SUM(TIME_TO_SEC(t2.t))/(60*60)) AS 'avg. km/hour'
    -> FROM marathon AS t1, marathon AS t2
    -> WHERE t1.stage >= t2.stage
    -> GROUP BY t1.stage;
+-------+----+----------+---------+----------+--------------+
| stage | km | t        | cum. km | cum. t   | avg. km/hour |
+-------+----+----------+---------+----------+--------------+
|     1 |  5 | 00:15:00 |       5 | 00:15:00 |      20.0000 |
|     2 |  7 | 00:19:30 |      12 | 00:34:30 |      20.8696 |
|     3 |  9 | 00:29:20 |      21 | 01:03:50 |      19.7389 |
|     4 |  5 | 00:17:50 |      26 | 01:21:40 |      19.1020 |
+-------+----+----------+---------+----------+--------------+
```

Видно, что бегун прибавил темп на втором этапе гонки, но потом, видимо, в результате усталости, только снижал его.

12.14. Управление порядком вывода запроса с помощью соединения

Задача

Вы хотите упорядочить вывод запроса, используя ту его характеристику, которую невозможно указать в инструкции ORDER BY. Например, вы хотите отсортировать строки по подгруппам, выводя первыми те группы, в которых больше всего строк, а последними – группы с наименьшим количеством строк. Но «количество строк группы» – это не свойство отдельных строк, поэтому вы не можете упорядочивать по нему.

Решение

Извлеките информацию о порядке вывода и сохраните в другой таблице. Затем объедините исходную таблицу с производной, используя производную для управления порядком сортировки.

Обсуждение

Обычно при сортировке результата запроса применяется инструкция ORDER BY (или GROUP BY) для указания имени столбца или столбцов, по которым нужно производить сортировку. Но иногда требуется выполнить сортировку по значениям, которые не содержатся в упорядочиваемых строках. Именно так дело обстоит с использованием групповых характеристик для упорядочивания строк. Следующий пример использует записи таблицы driver_log для иллюстрации вышесказанного:

```
mysql> SELECT * FROM driver_log ORDER BY id;
+--------+-------+------------+-------+
| rec_id | name  | trav_date  | miles |
+--------+-------+------------+-------+
|      1 | Ben   | 2001-11-30 |   152 |
|      2 | Suzi  | 2001-11-29 |   391 |
|      3 | Henry | 2001-11-29 |   300 |
|      4 | Henry | 2001-11-27 |    96 |
|      5 | Ben   | 2001-11-29 |   131 |
|      6 | Henry | 2001-11-26 |   115 |
|      7 | Suzi  | 2001-12-02 |   502 |
|      8 | Henry | 2001-12-01 |   197 |
|      9 | Ben   | 2001-12-02 |    79 |
|     10 | Henry | 2001-11-30 |   203 |
+--------+-------+------------+-------+
```

Такой запрос сортирует записи по столбцу идентификаторов, содержащемуся в строках. А если нам нужно вывести список и упорядочить его на основе суммарного значения, которое не присутствует в строках? Это немного сложнее. Предположим, что вы хотите вывести записи каждого водителя в порядке возрастания даты, при этом начать с тех водителей, которые проехали

самое больше расстояние. Выполнить подобную операцию при помощи суммарного запроса невозможно, так как тогда нельзя будет вывести записи для отдельных водителей. Но и обойтись без суммарного запроса невозможно, так как для сортировки необходимо итоговое значение. Выйдем из затруднительного положения, создав новую таблицу, содержащую суммарные значения, и объединим ее с исходной. Так мы сможем и вывести индивидуальные записи, и отсортировать их по суммарным значениям.

Чтобы сохранить итоговую информацию о километраже водителей, выполним следующие предложения:

```
mysql> CREATE TABLE tmp
    -> SELECT name, SUM(miles) AS driver_miles FROM driver_log GROUP BY name;
```

Сформируем значения, которые необходимы для размещения имен (`name`) в нужном порядке:

```
mysql> SELECT * FROM tmp ORDER BY driver_miles DESC;
+-------+--------------+
| name  | driver_miles |
+-------+--------------+
| Henry |          911 |
| Suzi  |          893 |
| Ben   |          362 |
+-------+--------------+
```

Затем используем значения `name` для соединения суммарной таблицы с таблицей `driver_log`, пользуясь значениями `driver_miles` для упорядочивания результата. Приведенный ниже запрос выводит общее количество километров. Это сделано только для того, чтобы было понятнее, как сортируются записи, нет никакой необходимости в их выводе. Нужны они только в инструкции `ORDER BY`.

```
mysql> SELECT tmp.driver_miles, driver_log.*
    -> FROM driver_log, tmp
    -> WHERE driver_log.name = tmp.name
    -> ORDER BY tmp.driver_miles DESC, driver_log.trav_date;
+--------------+--------+-------+------------+-------+
| driver_miles | rec_id | name  | trav_date  | miles |
+--------------+--------+-------+------------+-------+
|          911 |      6 | Henry | 2001-11-26 |   115 |
|          911 |      4 | Henry | 2001-11-27 |    96 |
|          911 |      3 | Henry | 2001-11-29 |   300 |
|          911 |     10 | Henry | 2001-11-30 |   203 |
|          911 |      8 | Henry | 2001-12-01 |   197 |
|          893 |      2 | Suzi  | 2001-11-29 |   391 |
|          893 |      7 | Suzi  | 2001-12-02 |   502 |
|          362 |      5 | Ben   | 2001-11-29 |   131 |
|          362 |      1 | Ben   | 2001-11-30 |   152 |
|          362 |      9 | Ben   | 2001-12-02 |    79 |
+--------------+--------+-------+------------+-------+
```

12.15. Преобразование подзапросов в операции соединения

Задача

Вы хотите использовать запрос, который включает в себя подзапрос, но MySQL будет поддерживать подзапросы только с версии 4.1.

Решение

Во многих случаях подзапрос можно переписать как соединение. Или вы можете написать программу, которая имитирует подзапрос. Вы даже можете заставить *mysql* генерировать предложения SQL, имитирующие подзапрос.

Обсуждение

Предположим, что у вас есть две таблицы `t1` и `t2` с таким содержимым:

```
mysql> SELECT col1 FROM t1;
+------+
| col1 |
+------+
| a    |
| b    |
| c    |
+------+
mysql> SELECT col2 FROM t2;
+------+
| col2 |
+------+
| b    |
| c    |
| d    |
+------+
```

Теперь предположим, что вы хотите найти те значения `t1`, которые содержатся и в `t2`, или значения `t1`, которые не содержатся в `t2`. На такие вопросы можно ответить при помощи подзапросов – вложения одного предложения `SELECT` внутрь другого, но MySQL не поддерживает подзапросы в версиях, предшествующих 4.1. Этот раздел посвящен тому, как обойти эту проблему.

Следующий запрос содержит подзапрос `IN()`, который выводит строки `t1`, значения столбца `col1` которых совпадает со значениями `col2` таблицы `t2`:

```
SELECT col1 FROM t1 WHERE col1 IN (SELECT col2 FROM t2);
```

Фактически это запрос поиска соответствий, так что его можно переписать при помощи простого соединения:

```
mysql> SELECT t1.col1 FROM t1, t2 WHERE t1.col1 = t2.col2;
+------+
| col1 |
```

```
+------+
| b    |
| c    |
+------+
```

На обратный вопрос (строки `t1`, не имеющие соответствий в `t2`) можно ответить, используя подзапрос `NOT IN()`:

```
SELECT col1 FROM t1 WHERE col1 NOT IN (SELECT col2 FROM t2);
```

Это задача отсутствия соответствий, которую можно решить при помощи `LEFT JOIN`, разновидности соединения, обсуждавшегося в рецепте 12.5. В нашем случае подзапрос `NOT IN()` эквивалентен такому левому соединению:

```
mysql> SELECT t1.col1 FROM t1 LEFT JOIN t2 ON t1.col1 = t2.col2
    -> WHERE t2.col2 IS NULL;
+------+
| col1 |
+------+
| a    |
+------+
```

В программе подзапрос можно имитировать, поработав над результатами двух запросов. Предположим, что вы хотите имитировать подзапрос `IN()`, который выбирает совпадающие значения двух таблиц:

```
SELECT * FROM t1 WHERE col1 IN (SELECT col2 FROM t2);
```

Если вы ожидаете, что внутренний `SELECT` вернет достаточно небольшое количество значений `col2`, то того же результата, что и подзапрос, можно достичь путем извлечения этих значений и формирования инструкции `IN()`, которая будет искать их в `col1`. Например, запрос `SELECT col2 FROM t2` выводит значения `b`, `c` и `d`. Используя этот результат, можно выбрать соответствующие значения `col1` таким запросом:

```
SELECT col1 FROM t1 WHERE col1 IN ('b','c','d')
```

Рассмотрим фрагмент кода на Python:

```
cursor = conn.cursor ()
cursor.execute ("SELECT col2 FROM t2")
if cursor.rowcount > 0:     # ничего не делать, если нет значений
    val = []                # список для хранения значений данных
    s = ""                  # строка для хранения заполнителей
    # сформировать строку с заполнителями: %s,%s,%s,...
    for (col2,) in cursor.fetchall ():  # извлечь значение col2 из каждой строки
        if s != "":
            s = s + ","     # разделить заполнители запятыми
        s = s + "%s"        # добавить заполнитель
        val.append (col2)   # добавить значение в список значений
    stmt = "SELECT col1 FROM t1 WHERE col1 IN (" + s + ")"
    cursor.execute (stmt, val)
    for (col1,) in cursor.fetchall ():  # извлечь значения col1 из результата
        print col1
cursor.close ()
```

Если вы предполагаете, что значений col2 будет очень много, можно сформировать отдельные предложения SELECT для каждого из них:

```
SELECT col1 FROM t1 WHERE col1 = 'b'
SELECT col1 FROM t1 WHERE col1 = 'c'
SELECT col1 FROM t1 WHERE col1 = 'd'
```

В программе это можно сделать так:

```
cursor = conn.cursor ()
cursor2 = conn.cursor ()
cursor.execute ("SELECT col2 FROM t2")
for (col2,) in cursor.fetchall ():  # извлечь значение col2 из каждой строки
    stmt = "SELECT col1 FROM t1 WHERE col1 = %s"
    cursor2.execute ("SELECT col1 FROM t1 WHERE col1 = %s", (col2,))
    for (col1,) in cursor2.fetchall (): # извлечь значения col1 из результата
        print col1
cursor.close ()
cursor2.close ()
```

Если у вас так много значений col2, что вы не хотите писать одну огромную инструкцию IN(), но не хотите и выдавать несметное множество отдельных предложений SELECT, то можно скомбинировать два этих способа. Разбейте множество значений col2 на более мелкие группы и создайте для каждой из них инструкцию IN(). У вас появится ряд менее громоздких запросов, каждый из которых ищет несколько значений:

```
SELECT col1 FROM t1 WHERE col1 IN (первая группа значений col2)
SELECT col1 FROM t1 WHERE col1 IN (вторая группа значений col2)
SELECT col1 FROM t1 WHERE col1 IN (третья группа значений col2)
...
```

Комбинированный подход можно реализовать так:

```
grp_size = 1000             # количество идентификаторов, выбираемых одновременно
cursor = conn.cursor ()
cursor.execute ("SELECT col2 FROM t2")
if cursor.rowcount > 0:     # ничего не делать, если нет значений
    col2 = []               # список для хранения значений данных
    for (val,) in cursor.fetchall ():   # извлечь значение col2 из каждой строки
        col2.append (val)
    nvals = len (col2)
    i = 0
    while i < nvals:
        if nvals < i + grp_size:
            j = nvals
        else:
            j = i + grp_size
        group = col2[i : j]
        s = ""                  # строка для хранения заполнителей
        val_list = []
        # сформировать строку с заполнителями: %s,%s,%s,...
        for val in group:
            if s != "":
```

```
                s = s + ","     # разделить заполнители запятыми
            s = s + "%s"        # добавить заполнитель
            val_list.append (val)   # добавить значение в список значений
        stmt = "SELECT col1 FROM t1 WHERE col1 IN (" + s + ")"
        print stmt
        cursor.execute (stmt, val_list)
        for (col1,) in cursor.fetchall (): # извлечь значение col1 из каждой строки
            print col1
        i = i + grp_size        # перейти к следующей группе значений
cursor.close ()
```

Имитировать в программе подзапрос NOT IN() несколько сложнее, чем подзапрос IN(). Подзапрос выглядит так:

```
SELECT col1 FROM t1 WHERE col1 NOT IN (SELECT col2 FROM t2);
```

Описываемый метод лучше всего работает для небольшого количества значений col1 и col2, так как приходится в хранить в памяти как минимум значения, возвращенные внутренним SELECT, чтобы можно было сравнивать их со значениями, возвращенными внешним SELECT. Приведенный пример хранит в памяти оба множества. Сначала извлечем значения col1 и col2:

```
cursor = conn.cursor ()
cursor.execute ("SELECT col1 FROM t1")
col1 = []
for (val, ) in cursor.fetchall ():
    col1.append (val)
cursor.execute ("SELECT col2 FROM t2")
col2 = []
for (val, ) in cursor.fetchall ():
    col2.append (val)
cursor.close ()
```

Затем проверим каждое значение col1 на предмет присутствия в наборе значений col2. Если не присутствует, то оно удовлетворяет условию NOT IN() подзапроса:

```
for val1 in col1:
    present = 0
    for val2 in col2:
        if val1 == val2:
            present = 1
            break
    if not present:
        print val1
```

Код выполняет просмотр (lookup) значений col2 в содержащем их массиве. Можно повысить эффективность этой операции, используя ассоциативную структуру данных. Например, в Perl или Python можно поместить значения col2 в хеш или словарь. Пример такой реализации приведен в рецепте 10.28.

Еще одним способом имитации подзапросов (по крайней мере, подзапросов типа IN()), является формирование необходимых предложений SQL внутри

одного экземпляра *mysql* и передача их другому экземпляру на исполнение. Посмотрите на результат такого запроса:

```
mysql> SELECT CONCAT('SELECT col1 FROM t1 WHERE col1 = \'', col2, '\';')
    -> FROM t2;
+-----------------------------------------------------------+
| CONCAT('SELECT col1 FROM t1 WHERE col1 = \'', col2, '\';') |
+-----------------------------------------------------------+
| SELECT col1 FROM t1 WHERE col1 = 'b';                      |
| SELECT col1 FROM t1 WHERE col1 = 'c';                      |
| SELECT col1 FROM t1 WHERE col1 = 'd';                      |
+-----------------------------------------------------------+
```

Этот запрос извлекает значения `col2` из `t2` и использует их для формирования набора предложений `SELECT`, которые ищут соответствующие значения `col1` в `t1`. Если вы выдаете запрос в пакетном режиме и подавляете вывод заголовков, то *mysql* выводит только текст предложений SQL безо всякого обрамления. Этот вывод можно передать другому экземпляру *mysql* для выполнения запросов. Результат будет таким же, как при выполнении подзапроса. Предлагаю один из способов выполнения процедуры, в котором предполагается, что предложение `SELECT`, содержащее выражение `CONCAT()`, хранится в файле *make_select.sql*:

```
% mysql -N cookbook < make_select.sql > tmp
```

Для *mysql* указана опция *-N*, определяющая необходимость подавления вывода заголовков, так что они не попадут в файл вывода *tmp*. Содержимое *tmp* будет таким:

```
SELECT col1 FROM t1 WHERE col1 = 'b';
SELECT col1 FROM t1 WHERE col1 = 'c';
SELECT col1 FROM t1 WHERE col1 = 'd';
```

Чтобы выполнить запросы из файла и сформировать вывод, имитируя подзапрос, используйте такую команду:

```
% mysql -N cookbook < tmp
b
c
```

Второй экземпляр *mysql* также содержит опцию *-N*, поскольку в противном случае вывод будет содержать строку заголовка каждого выполненного предложения `SELECT` (попробуйте не указывать *-N* и посмотрите, что получится).

Важным ограничением использования *mysql* для формирования предложений SQL является следующее: если значения `col2` содержат кавычки или другие специальные символы, то генерируемые запросы будут построены некорректно.[1]

[1] В версии MySQL 4.0.3 появилась функция `QUOTE()`, позволяющая экранировать специальные символы, с тем чтобы их можно было использовать в предложениях SQL.

12.16. Параллельный выбор записей из нескольких таблиц

Задача

Вы хотите выбирать строки одну за другой из нескольких таблиц или выбирать несколько наборов строк из одной таблицы – все в одно результирующее множество.

Решение

Используйте операцию UNION для соединения нескольких результирующих множеств в одно.

Обсуждение

Соединение JOIN удобно для расположения столбцов из разных таблиц вместе бок о бок. Но оно не подходит, если вас интересует результирующее множество, которое содержит набор строк из нескольких таблиц, следующих одна за другой, или несколько наборов строк одной таблицы. Для таких операций удобно использовать объединение – UNION. Объединение UNION позволяет запустить несколько предложений SELECT и «вертикально» склеить их результаты. Вы получаете вывод в одном результирующем множестве вместо того, чтобы запускать несколько запросов и получать несколько результирующих множеств.

Операция UNION появилась в версии MySQL 4.0. В этом разделе показано, как ее использовать, и описано несколько способов решения задач в старых версиях MySQL.

Предположим, что у вас есть две таблицы со списками потенциальных и фактических клиентов (prospect и customer) и третья (vendor), перечисляющая ваших поставщиков. Вы хотите создать единый список рассылки, объединив имена и адреса из всех трех таблиц. UNION обеспечивает решение такой задачи. Предположим, что таблицы имеют такое содержимое:

```
mysql> SELECT * FROM prospect;
+---------+-------+-----------------------+
| fname   | lname | addr                  |
+---------+-------+-----------------------+
| Peter   | Jones | 482 Rush St., Apt. 402 |
| Bernice | Smith | 916 Maple Dr.         |
+---------+-------+-----------------------+
mysql> SELECT * FROM customer;
+-----------+------------+---------------------+
| last_name | first_name | address             |
+-----------+------------+---------------------+
| Peterson  | Grace      | 16055 Seminole Ave. |
| Smith     | Bernice    | 916 Maple Dr.       |
| Brown     | Walter     | 8602 1st St.        |
+-----------+------------+---------------------+
```

```
mysql> SELECT * FROM vendor;
+-------------------+---------------------+
| company           | street              |
+-------------------+---------------------+
| ReddyParts, Inc.  | 38 Industrial Blvd. |
| Parts-to-go, Ltd. | 213B Commerce Park. |
+-------------------+---------------------+
```

Столбцы таблиц похожи, но не идентичны: `prospect` и `customer` используют разные имена для столбцов имени и фамилии, а в таблице `vendor` всего один столбец для названия компании. Все это не имеет никакого значения для `UNION`; единственное, в чем необходимо убедиться, – изо всех таблиц должно выбираться одинаковое количество столбцов и в одинаковом порядке. Следующий запрос показывает, как выбирать имена и адреса из трех таблиц одновременно:

```
mysql> SELECT fname, lname, addr FROM prospect
    -> UNION
    -> SELECT first_name, last_name, address FROM customer
    -> UNION
    -> SELECT company, '', street FROM vendor;
+-------------------+----------+-----------------------+
| fname             | lname    | addr                  |
+-------------------+----------+-----------------------+
| Peter             | Jones    | 482 Rush St., Apt. 402 |
| Bernice           | Smith    | 916 Maple Dr.         |
| Grace             | Peterson | 16055 Seminole Ave.   |
| Walter            | Brown    | 8602 1st St.          |
| ReddyParts, Inc.  |          | 38 Industrial Blvd.   |
| Parts-to-go, Ltd. |          | 213B Commerce Park.   |
+-------------------+----------+-----------------------+
```

Имена и типы в результирующем множестве определяются именами и типами столбцов, извлеченных первым предложением `SELECT`. Обратите внимание, что по умолчанию `UNION` удаляет повторения; `Bernice Smith` присутствует и в таблице `prospect`, и в таблице `customer`, но результирующее множество содержит только одну такую запись. Если вы хотите выбрать все записи, включая повторения, укажите после первого ключевого слова `UNION` ключевое слово `ALL`:

```
mysql> SELECT fname, lname, addr FROM prospect
    -> UNION ALL
    -> SELECT first_name, last_name, address FROM customer
    -> UNION
    -> SELECT company, '', street FROM vendor;
+-------------------+----------+-----------------------+
| fname             | lname    | addr                  |
+-------------------+----------+-----------------------+
| Peter             | Jones    | 482 Rush St., Apt. 402 |
| Bernice           | Smith    | 916 Maple Dr.         |
| Grace             | Peterson | 16055 Seminole Ave.   |
```

```
| Bernice           | Smith    | 916 Maple Dr.          |
| Walter            | Brown    | 8602 1st St.           |
| ReddyParts, Inc.  |          | 38 Industrial Blvd.    |
| Parts-to-go, Ltd. |          | 213B Commerce Park.    |
+-------------------+----------+------------------------+
```

Поскольку необходимо выбирать одинаковое количество столбцов из всех таблиц, предложение SELECT для таблицы vendor (в которой всего один столбец для названия компании) извлекает фиктивный (пустой) столбец названия. Можно было бы обеспечить одинаковое количество столбцов и другим способом: объединив столбцы имени и фамилии таблиц prospect и customer в один столбец:

```
mysql> SELECT CONCAT(lname,', ',fname) AS name, addr FROM prospect
    -> UNION
    -> SELECT CONCAT(last_name,', ',first_name), address FROM customer
    -> UNION
    -> SELECT company, street FROM vendor;
+-------------------+------------------------+
| name              | addr                   |
+-------------------+------------------------+
| Jones, Peter      | 482 Rush St., Apt. 402 |
| Smith, Bernice    | 916 Maple Dr.          |
| Peterson, Grace   | 16055 Seminole Ave.    |
| Brown, Walter     | 8602 1st St.           |
| ReddyParts, Inc.  | 38 Industrial Blvd.    |
| Parts-to-go, Ltd. | 213B Commerce Park.    |
+-------------------+------------------------+
```

Для того чтобы упорядочить все результирующее множество, добавьте инструкцию ORDER BY после последнего предложения SELECT. Если вы в инструкции ORDER BY ссылаетесь на столбцы по имени, то это должны быть имена из первого предложения SELECT, так как именно они используются в результирующем множестве. Например, для сортировки по полю name выполните следующее:

```
mysql> SELECT CONCAT(lname,', ',fname) AS name, addr FROM prospect
    -> UNION
    -> SELECT CONCAT(last_name,', ',first_name), address FROM customer
    -> UNION
    -> SELECT company, street FROM vendor
    -> ORDER BY name;
+-------------------+------------------------+
| name              | addr                   |
+-------------------+------------------------+
| Brown, Walter     | 8602 1st St.           |
| Jones, Peter      | 482 Rush St., Apt. 402 |
| Parts-to-go, Ltd. | 213B Commerce Park.    |
| Peterson, Grace   | 16055 Seminole Ave.    |
| ReddyParts, Inc.  | 38 Industrial Blvd.    |
| Smith, Bernice    | 916 Maple Dr.          |
+-------------------+------------------------+
```

В MySQL есть возможность сортировки результатов отдельных предложений SELECT внутри UNION. Для этого заключим указанное предложение SELECT (включая его инструкцию ORDER BY) в скобки:

```
mysql> (SELECT CONCAT(lname,', ',fname) AS name, addr
    -> FROM prospect ORDER BY 1)
    -> UNION
    -> (SELECT CONCAT(last_name,', ',first_name), address
    -> FROM customer ORDER BY 1)
    -> UNION
    -> (SELECT company, street FROM vendor ORDER BY 1);
+------------------+-----------------------+
| name             | addr                  |
+------------------+-----------------------+
| Jones, Peter     | 482 Rush St., Apt. 402 |
| Smith, Bernice   | 916 Maple Dr.         |
| Brown, Walter    | 8602 1st St.          |
| Peterson, Grace  | 16055 Seminole Ave.   |
| Parts-to-go, Ltd. | 213B Commerce Park.  |
| ReddyParts, Inc. | 38 Industrial Blvd.   |
+------------------+-----------------------+
```

Похожий синтаксис можно использовать и для инструкции LIMIT. То есть вы можете ограничить все результирующее множество, используя инструкцию LIMIT в самом конце предложения, а можете применять ее к отдельным запросам SELECT, заключая их в скобки. В некоторых случаях разумно использовать ORDER BY в сочетании с LIMIT. Предположим, вы хотите выбрать счастливого призера какой-то рекламной кампании. Чтобы выбрать победителя случайным образом из всего результирующего множества для трех таблиц, сделайте следующее:

```
mysql> SELECT CONCAT(lname,', ',fname) AS name, addr FROM prospect
    -> UNION
    -> SELECT CONCAT(last_name,', ',first_name), address FROM customer
    -> UNION
    -> SELECT company, street FROM vendor
    -> ORDER BY RAND() LIMIT 1;
+----------------+--------------------+
| name           | addr               |
+----------------+--------------------+
| Peterson, Grace | 16055 Seminole Ave. |
+----------------+--------------------+
```

Чтобы выбрать победителя в каждой таблице, сделайте так:

```
mysql> (SELECT CONCAT(lname,', ',fname) AS name, addr
    -> FROM prospect ORDER BY RAND() LIMIT 1)
    -> UNION
    -> (SELECT CONCAT(last_name,', ',first_name), address
    -> FROM customer ORDER BY RAND() LIMIT 1)
    -> UNION
    -> (SELECT company, street
```

```
    -> FROM vendor ORDER BY RAND() LIMIT 1);
+------------------+---------------------+
| name             | addr                |
+------------------+---------------------+
| Smith, Bernice   | 916 Maple Dr.       |
| ReddyParts, Inc. | 38 Industrial Blvd. |
+------------------+---------------------+
```

Если вас удивит результат (почему выбрано не три строки?), вспомните о присутствии Bernice в двух таблицах и о том, что UNION удаляет повторения. Если случилось так, что и первый и второй запросы SELECT выбрали Bernice, одна из строк результата будет удалена. (Если в трех таблицах нет повторяющихся строк, запрос всегда будет возвращать три строки.) Естественно, вы можете обеспечить вывод трех записей, используя UNION ALL или запуская предложения SELECT по отдельности.

Если у вас более ранняя версия MySQL, чем 4.0, вы не можете использовать UNION. Но можете получить те же результаты, создав временную таблицу, сохраняя результаты нескольких запросов SELECT в этой таблице и выбирая ее содержимое. В MySQL 3.23 можно создать временную таблицу (CREATE TABLE ... SELECT для первого предложения SELECT), а затем последовательно извлекать в нее другие результирующие множества:

```
mysql> CREATE TABLE tmp SELECT CONCAT(lname,', ',fname) AS name, addr
    -> FROM prospect;
mysql> INSERT INTO tmp (name, addr)
    -> SELECT CONCAT(last_name,', ',first_name), address
    -> FROM customer;
mysql> INSERT INTO tmp (name, addr)
    -> SELECT company, street FROM vendor;
```

Если ваша версия MySQL старше, чем 3.23, сначала создайте таблицу, затем выберите в нее каждую запись:

```
mysql> CREATE TABLE tmp (name CHAR(40), addr CHAR(40));
mysql> INSERT INTO tmp (name, addr)
    -> SELECT CONCAT(lname,', ',fname), addr
    -> FROM prospect;
mysql> INSERT INTO tmp (name, addr)
    -> SELECT CONCAT(last_name,', ',first_name), address
    -> FROM customer;
mysql> INSERT INTO tmp (name, addr)
    -> SELECT company, street FROM vendor;
```

После вставки отдельных результатов во временную таблицу выберите ее содержимое:

```
mysql> SELECT * FROM tmp;
+------------------+------------------------+
| name             | addr                   |
+------------------+------------------------+
| Jones, Peter     | 482 Rush St., Apt. 402 |
| Smith, Bernice   | 916 Maple Dr.          |
```

```
| Peterson, Grace   | 16055 Seminole Ave.    |
| Smith, Bernice    | 916 Maple Dr.          |
| Brown, Walter     | 8602 1st St.           |
| ReddyParts, Inc.  | 38 Industrial Blvd.    |
| Parts-to-go, Ltd. | 213B Commerce Park.    |
+-------------------+------------------------+
```

Обратите внимание на то, что результат больше похож на `UNION ALL`, а не на `UNION`, так как повторения не удалены. Чтобы удалить из вывода дубликаты, создадим таблицу с уникальным индексом для столбцов `name` и `addr`:

```
mysql> CREATE TABLE tmp (name CHAR(40), addr CHAR(40), UNIQUE (name, addr));
mysql> INSERT INTO ...
...
mysql> SELECT * FROM tmp;
+-------------------+------------------------+
| name              | addr                   |
+-------------------+------------------------+
| Brown, Walter     | 8602 1st St.           |
| Jones, Peter      | 482 Rush St., Apt. 402 |
| Parts-to-go, Ltd. | 213B Commerce Park.    |
| Peterson, Grace   | 16055 Seminole Ave.    |
| ReddyParts, Inc.  | 38 Industrial Blvd.    |
| Smith, Bernice    | 916 Maple Dr.          |
+-------------------+------------------------+
```

Если таблица создается без уникального индекса, то можно удалить повторения при извлечении, используя `DISTINCT`, хотя это менее эффективно.

12.17. Вставка записей в таблицу, включающую значения из другой

Задача

Вам нужно вставить запись в таблицу, требующую значение идентификатора. Вы же знаете только соответствующее идентификатору имя, но не сам идентификатор.

Решение

Если предположить, что у вас есть справочная таблица, сопоставляющая имена идентификаторам, то можно создать запись с помощью предложения `INSERT INTO ... SELECT`, в котором `SELECT` выполняет просмотр имен для получения значения соответствующего идентификатора.

Обсуждение

Справочные таблицы часто использовались в этой главе в соединениях, обычно для сопоставления идентификаторам или кодам значений описательных имен. Но справочные таблицы полезны не только для предложений

SELECT, они могут помочь и при создании новых записей. Давайте вернемся к таблицам artist и painting, содержащим информацию о вашей коллекции произведений искусства. Предположим, что вы едете в Миннесоту, где удается приобрести всего за 51 доллар репродукцию картины «Жонглеры» («Les jongleurs») Ренуара. Ренуар уже отмечен в таблице artist, так что нет необходимости добавлять туда новую запись. Но нужно добавить запись в таблицу painting. Чтобы создать ее, необходимо сохранить идентификатор художника, название картины, штат покупки и цену. Вы знаете все, кроме идентификатора автора, и не хотите сами искать его в таблице artist. Раз Ренуар там уже есть, почему бы не позволить MySQL найти для вас его идентификатор? Для добавления новой записи выполним предложение INSERT ... SELECT. Укажем все известные значения в списке столбцов вывода SELECT и используем инструкцию WHERE для поиска идентификатора художника по его фамилии:

```
mysql> INSERT INTO painting (a_id, title, state, price)
    -> SELECT a_id, 'Les jongleurs', 'MN', 51
    -> FROM artist WHERE name = 'Renoir';
```

Естественно, вам не захочется писать вручную полный текст этого запроса при каждой новой покупке. Но легко можно написать небольшой сценарий, который по фамилии художника, названию картины, штату и стоимости покупки формировал бы запрос и выполнял его для вас. Также можно написать код для проверки вхождения художника в таблицу artist и, в случае его отсутствия, генерирования нового значения идентификатора для художника перед выполнением вставки новой записи в painting. Просто выполните следующий запрос перед созданием новой строки в таблице painting:

```
INSERT IGNORE INTO artist (name) VALUES('фамилия художника');
```

12.18. Обновление одной таблицы на основе значений другой

Задача

Вам нужно обновить существующие записи одной таблицы на основе содержимого записей другой таблицы, но MySQL еще не позволяет использовать соединения в инструкции WHERE предложений UPDATE. Так что у вас нет способа сопоставить две таблицы.

Решение

Создайте новую таблицу, заполненную данными результата соединения исходной таблицы и таблицы, содержащей новую информацию. Затем замените исходную таблицу новой. Или напишите программу, выбирающую информацию из связанной таблицы, и выполните необходимые запросы для обновления исходной таблицы. Или используйте *mysql* для формирования и выполнения запросов.

Обсуждение

Иногда при обновлении записей одной таблицы необходимо сослаться на записи другой таблицы. Вспомним таблицу states, использовавшуюся в нескольких предыдущих рецептах и содержащую строки типа:

```
mysql> SELECT * FROM states;
+----------------+--------+------------+----------+
| name           | abbrev | statehood  | pop      |
+----------------+--------+------------+----------+
| Alaska         | AK     | 1959-01-03 |   550043 |
| Alabama        | AL     | 1819-12-14 |  4040587 |
| Arkansas       | AR     | 1836-06-15 |  2350725 |
| Arizona        | AZ     | 1912-02-14 |  3665228 |
...
```

Теперь предположим, что вы хотите добавить в эту таблицу несколько новых столбцов, используя информацию из другой таблицы city, содержащей такие данные о каждом штате, как его столица и крупнейший (по количеству жителей) город:

```
mysql> SELECT * FROM city;
+---------------+---------------+---------------+
| state         | capital       | largest       |
+---------------+---------------+---------------+
| Alabama       | Montgomery    | Birmingham    |
| Alaska        | Juneau        | Anchorage     |
| Arizona       | Phoenix       | Phoenix       |
| Arkansas      | Little Rock   | Little Rock   |
...
```

Достаточно просто добавить в таблицу states новые столбцы capital и largest с помощью предложения ALTER TABLE. Но как тогда изменять строки, чтобы заполнить новые столбцы соответствующими значениями? Удобнее всего выполнить запрос UPDATE, использующий синтаксис соединения в инструкции WHERE:

```
UPDATE states,city
SET states.capital = city.capital, states.largest = city.largest
WHERE states.name = city.state;
```

Но, к сожалению, так поступить нельзя, поскольку MySQL еще не поддерживает эту возможность. Вторым решением могло бы быть использование подзапроса в инструкции WHERE, но подзапросы планируется включить только в версию MySQL 4.1. Что же остается? Конечно, не хотелось бы обновлять каждую строку вручную. Это невероятно трудоемко, да и глупо, учитывая то, что вся новая информация уже хранится в таблице city. Таблицы states и city содержат общий ключ (названия штатов), и эту информацию можно использовать для связывания таблиц и выполнения обновления. Есть несколько путей достижения того же результата, что и многотабличное обновление:

- Создайте новую таблицу, похожую на исходную, но содержащую дополнительные столбцы, добавленные из связанной таблицы city. Заполните

новую таблицу данными, используя результат соединения таблиц states и city, затем замените исходную таблицу новой.

- Напишите программу, использующую информацию из таблицы city для формирования и выполнения предложений UPDATE, которые обновляют по одному штату в каждом запросе в таблице states.
- Используйте *mysql* для формирования предложений UPDATE.

Обновление связанной таблицы путем ее замены

Замена таблицы выполняется так. Чтобы расширить таблицу states, включив в нее столбцы capital и largest из таблицы city, создайте таблицу tmp, аналогичную states, и добавьте в нее столбцы capital и largest:

```
CREATE TABLE tmp
(
    name        VARCHAR(30) NOT NULL,   # название штата
    abbrev      CHAR(2) NOT NULL,       # двухсимвольная аббревиатура
    statehood   DATE,                   # дата вступления в Союз
    pop         BIGINT,                 # население на 4/1990
    capital     VARCHAR(30),            # столица
    largest     VARCHAR(30),            # наиболее населенный город
    PRIMARY KEY (abbrev)
);
```

Затем заполните tmp данными, используя результат соединения между states и city, которое сопоставляет строки по названиям штатов:

```
INSERT INTO tmp (name, abbrev, statehood, pop, capital, largest)
SELECT
    states.name, states.abbrev, states.statehood, states.pop,
    city.capital, city.largest
FROM
    states LEFT JOIN city ON states.name = city.state;
```

Заметьте, что запрос использует соединение LEFT JOIN. Предположим, что таблица city не полная и не содержит по строке для каждого штата. Тогда обычное соединение не сможет сформировать строку вывода для штатов, отсутствующих в таблице city, в результате в таблице tmp не будет записей для таких штатов, даже если они присутствуют в таблице states. Нехорошо! LEFT JOIN обеспечивает формирование строки вывода SELECT для каждой строки states вне зависимости от ее соответствия строке таблицы city. Любой штат, отсутствующий в таблице city, будет содержать значения NULL в столбцах capital и largest таблицы tmp, что вполне подходит для тех случаев, когда вам неизвестны названия городов, а формирование неполной строки, конечно же, предпочтительнее полной потери строки.

В результате таблица tmp похожа на исходную, но содержит два новых столбца, capital и largest (можете проверить). Убедившись в том, что таблица tmp нас устраивает, используем ее для замены исходной таблицы states:

```
DROP TABLE states;
ALTER TABLE tmp RENAME TO states;
```

Если вы хотите исключить возможность даже кратковременной недоступности таблицы states, выполните замену таким способом:

```
RENAME TABLE states TO states_old, tmp TO states;
DROP TABLE states_old;
```

Обновление связанной таблицы из программы

Метод замены таблиц эффективен, так как всю работу делает сервер. Но он больше всего подходит для тех случаев, когда изменяются все или почти все строки таблицы. Если же обновляется всего несколько строк, проще изменить таблицу «на месте» там, где необходимо. Кроме того, замена таблицы требует как минимум удвоения исходного пространства таблицы states на время выполнения процедуры обновления. Если обновляется большая таблица, то вы можете не захотеть использовать все это пространство.

Второй способ обновления таблицы на основе информации из связанной таблицы заключается в считывании данных из связанной таблицы и использования их для формирования предложений UPDATE. Например, чтобы обновить таблицу states информацией, хранящейся в таблице city, прочитайте названия городов и используйте их для создания и запуска ряда таких запросов:

```
UPDATE states SET capital = 'Montgomery', largest = 'Birmingham'
WHERE name = 'Alabama';
UPDATE states SET capital = 'Juneau', largest = 'Anchorage'
WHERE name = 'Alaska';
UPDATE states SET capital = 'Phoenix', largest = 'Phoenix'
WHERE name = 'Arizona';
UPDATE states SET capital = 'Little Rock', largest = 'Little Rock'
WHERE name = 'Arkansas';
...
```

Для выполнения такой процедуры сначала измените таблицу states так, чтобы она содержала новые столбцы:[1]

```
ALTER TABLE states ADD capital VARCHAR(30), ADD largest VARCHAR(30);
```

Затем напишите программу, которая читает таблицу city и использует ее содержимое для формирования предложений UPDATE, изменяющих таблицу states. Рассмотрим пример сценария *update_cities.pl*, выполняющего такие операции:

```
#! /usr/bin/perl -w
# update_cities.pl - обновление столбцов capital и largest таблицы states
# с использованием содержимого таблицы city.  Предполагается, что таблица states
# была изменена так, что теперь содержит столбцы capital и largest.

use strict;
use lib qw(/usr/local/apache/lib/perl);
use Cookbook;
```

[1] Если вы уже изменили таблицу states, выполняя процедуру замены таблицы, то сначала восстановите ее исходное состояние, удалив столбцы capital и largest.

```
my $dbh = Cookbook::connect ();

my $sth = $dbh->prepare ("SELECT state, capital, largest FROM city");
$sth->execute ();
while (my ($state, $capital, $largest) = $sth->fetchrow_array ())
{
    $dbh->do ("UPDATE states SET capital = ?, largest = ? WHERE name = ?",
                undef, $capital, $largest, $state);
}

$dbh->disconnect ();

exit (0);
```

Все имена таблиц и столбцов встроены в сценарий, что делает его узко специальным. Вы можете обобщить процесс, написав функцию, которая принимает параметры, указывающие имена таблиц, столбцы, которые должны использоваться для сопоставления записей в двух таблицах, и столбцы для обновления строк. Один из путей решения задачи реализован в сценарии *update_related.pl* из каталога *joins* дистрибутива `recipes`.

Обновление связанной таблицы с помощью mysql

Если ваши значения данных не требуют какой-то специальной обработки внутренних кавычек или других специальных символов, то формировать и обрабатывать предложения `UPDATE` можно с помощью *mysql*. Похожая техника была описана в рецепте 12.17 при использовании *mysql* для имитации подзапроса.

Поместите следующее предложение в файл *update_cities.sql*:

```
SELECT CONCAT('UPDATE states SET capital = \'',capital,
    '\', largest = \'',largest,'\' WHERE name = \'',state,'\';')
FROM city;
```

Запрос читает строки таблицы `city` и использует их для формирования предложения, обновляющего таблицу `states`. Выполним запрос и сохраним результат в *tmp*:

```
% mysql -N cookbook < update_cities.sql > tmp
```

Файл *tmp* будет содержать предложения, похожие на запросы, сформированные сценарием *update_cities.pl*. Считая, что столбцы `capital` и `largest` добавлены в таблицу `states`, вы можете выполнить предложения для обновления таблицы так:

```
% mysql cookbook < tmp
```

12.19. Создание справочной таблицы с помощью соединения

Задача

Таблица хранит длинные описания в столбце идентификаторов. Вы хотите преобразовать этот столбец в короткие значения идентификаторов и исполь-

зовать описания для создания справочной таблицы, сопоставляющей идентификаторы описаниям.

Решение

Используйте один из способов обновления связанной таблицы, описанных в рецепте 12.18.

Обсуждение

Общий подход для экономии места – хранить в таблице идентификационные номера или коды вместо описательных строк. Повышается и производительность, поскольку числа быстрее индексировать и извлекать, чем строки. (Для запросов, в которых необходим вывод имен, объедините значения идентификаторов со справочной таблицей вида идентификатор-название.) Когда вы создаете новую таблицу, помните об этой стратегии и сразу проектируйте таблицу так, чтобы ее можно было использовать со справочной таблицей. Но может случаться так, что у вас уже есть таблица с описательными строками, которую можно преобразовать так, чтобы она использовала значения идентификаторов. В этом разделе описано, как создать справочную таблицу, сопоставляющую каждому описанию его идентификатор, и как преобразовать описания в идентификаторы в исходной таблице. Будем использовать ALTER TABLE в сочетании с приемами обновления связанной таблицы.

Предположим, вы коллекционируете монеты и решили отслеживать свою коллекцию с помощью базы данных, используя такую таблицу:

```
CREATE TABLE coin
(
    id      INT UNSIGNED NOT NULL AUTO_INCREMENT,
    date    CHAR(5) NOT NULL,   # 4 цифры года + буква[1]
    denom   CHAR(20) NOT NULL,  # название номинала (например, Lincoln cent)
    PRIMARY KEY (id)
);
```

Каждой монете автоматически присваивается идентификатор как значение AUTO_INCREMENT; кроме того, записываются год выпуска каждой монеты и ее название номинала (denomination name). Введенные записи выглядят так:

```
mysql> SELECT * FROM coin;
+----+-------+--------------------+
| id | date  | denom              |
+----+-------+--------------------+
|  1 | 1944s | Lincoln cent       |
|  2 | 1977  | Roosevelt dime     |
|  3 | 1955d | Lincoln cent       |
|  4 | 1938  | Jefferson nickel   |
|  5 | 1964  | Kennedy half dollar |
|  6 | 1959  | Lincoln cent       |
```

[1] Буква обозначает место, где сделана монета (принято в США). – *Примеч. лит. ред.*

```
|  7 | 1945  | Jefferson nickel    |
|  8 | 1905  | Buffalo nickel      |
|  9 | 1924  | Mercury head dime   |
| 10 | 2001  | Roosevelt dime      |
| 11 | 1937  | Mercury head dime   |
| 12 | 1977  | Kennedy half dollar |
+----+-------+---------------------+
```

Таблица хранит информацию, которая вам необходима, но вы замечаете, что на указание названий номиналов монет в каждой записи тратится много места, и все станет еще серьезнее, когда в таблице будет много записей. Было бы эффективнее хранить в таблице `coin` идентификаторы названий номиналов монет, и при необходимости находить название номинала в таблице `denom`, содержащей все названия номиналов монет и их идентификаторы. (Преимущество такого метода для небольшой таблицы может быть неочевидным, но когда коллекция вырастет до 10 000 экземпляров, экономия пространства от хранения номеров вместо строк станет весьма ощутимой.)

Процедура создания справочной таблицы и преобразования исходной таблицы `coin` такова:

1. Создайте справочную таблицу `denom` для хранения соответствий идентификаторов названиям.
2. Заполните таблицу `denom` названиями номиналов из исходной таблицы `coin`.
3. Замените названия номиналов монет в таблице `coin` на соответствующие значения идентификаторов.

Таблица `denom` должна включать названия всех номиналов монет и соответствующие идентификаторы, поэтому ее структура может быть такой:

```
CREATE TABLE denom
(
    denom_id    INT UNSIGNED NOT NULL AUTO_INCREMENT,
    name        CHAR(20) NOT NULL,
    PRIMARY KEY (denom_id)
);
```

Чтобы заполнить таблицу данными, вставьте в нее те названия номиналов, которые присутствуют в таблице `coin`. Используйте `SELECT DISTINCT`, поскольку каждое название номинала должно быть введено ровно один раз:

```
INSERT INTO denom (name) SELECT DISTINCT denom FROM coin;
```

Предложение `INSERT` добавляет в таблицу `denom` только название номинала; `denom_id` – это столбец `AUTO_INCREMENT`, так что MySQL автоматически присвоит ему последовательные значения. Результирующая таблица будет такой:

```
+----------+---------------------+
| denom_id | name                |
+----------+---------------------+
|        1 | Lincoln cent        |
|        2 | Roosevelt dime      |
|        3 | Jefferson nickel    |
```

```
|        4 | Kennedy half dollar |
|        5 | Buffalo nickel      |
|        6 | Mercury head dime   |
+----------+---------------------+
```

В MySQL версии 3.23 и выше вы можете создать таблицу denom и заполнить ее данными в одном предложении CREATE TABLE ... SELECT:

```
CREATE TABLE denom
(
    denom_id    INT UNSIGNED NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (denom_id)
)
SELECT DISTINCT denom AS name FROM coin;
```

Следующим шагом после создания таблицы denom является преобразование названий номиналов в таблице coin в соответствующие идентификаторы:

- Создайте таблицу tmp, похожую на coin, но содержащую столбец denom_id вместо столбца denom.
- Заполните tmp результатами соединения таблиц coin и denom.
- Используйте таблицу tmp для замены исходной таблицы coin.

Для создания таблицы tmp выполните предложение CREATE TABLE, которое похоже на первоначальное предложение для создания coin, но подставьте в нем столбец denom_id вместо столбца denom:

```
CREATE TABLE tmp
(
    id          INT UNSIGNED NOT NULL AUTO_INCREMENT,
    date        CHAR(5) NOT NULL,   # 4 цифры года + буква
    denom_id    INT UNSIGNED NOT NULL,  # идентификатор названия номинала
    PRIMARY KEY (id)
);
```

Затем заполните tmp, используя соединение coin и denom:

```
INSERT INTO tmp (id, date, denom_id)
SELECT coin.id, coin.date, denom.denom_id
FROM coin, denom
WHERE coin.denom = denom.name;
```

Наконец, замените исходную таблицу coin таблицей tmp:

```
DROP TABLE coin;
ALTER TABLE tmp RENAME TO coin;
```

В MySQL версии 3.23 и выше вы можете создать и заполнить таблицу tmp в одном предложении:

```
CREATE TABLE tmp
(
    PRIMARY KEY (id)
)
SELECT coin.id, coin.date, denom.denom_id
FROM coin, denom
WHERE coin.denom = denom.name;
```

Теперь, как и раньше, замените coin на tmp.

В качестве еще одного способа преобразования таблицы coin после создания таблицы denom можно предложить изменение coin на месте, без использования таблицы tmp:

1. Добавьте столбец denom_id в таблицу coin при помощи ALTER TABLE.
2. Заполните значение denom_id каждой строки идентификатором, соответствующим ее названию номинала denom.
3. Удалите столбец denom.

Чтобы выполнить процедуру, добавьте в coin столбец для хранения идентификаторов:

```
ALTER TABLE coin ADD denom_id INT UNSIGNED NOT NULL;
```

Затем заполните столбец denom_id соответствующими идентификаторами, используя соответствие названий номиналов идентификаторам, хранящимся в таблице denom. Приведем небольшой сценарий, обновляющий значения идентификаторов в таблице coin для одного номинала за один проход:

```
#! /usr/bin/perl -w
# update_denom.pl - Для каждого названия номинала из таблицы denom обновить записи
# таблицы coin, содержащие такое название номинала, соответствующим идентификатором.

use strict;
use lib qw(/usr/local/apache/lib/perl);
use Cookbook;

my $dbh = Cookbook::connect ();

my $sth = $dbh->prepare ("SELECT denom_id, name FROM denom");
$sth->execute ();
while (my ($denom_id, $name) = $sth->fetchrow_array ())
{
    # Для записей таблицы coin с указанным названием номинала добавить
    # соответствующее значение denom_id из таблицы denom
    $dbh->do ("UPDATE coin SET denom_id = ? WHERE denom = ?",
                undef, $denom_id, $name);
}

$dbh->disconnect ();
exit (0);
```

Этот сценарий извлекает каждую пару идентификатор/название номинала из таблицы denom и формирует предложение UPDATE для изменения всех строк таблицы coin, содержащих название номинала, путем установки их значений denom_id в соответствующий идентификатор. Когда сценарий завершит работу, все строки таблицы coin будут содержать обновленные значения denom_id. Теперь столбец denom больше не нужен, и от него можно избавиться:

```
ALTER TABLE coin DROP denom;
```

Какой бы способ вы ни использовали для преобразования таблицы coin, ее содержимое в результате будет выглядеть так:

```
mysql> SELECT * FROM coin;
+----+-------+----------+
| id | date  | denom_id |
+----+-------+----------+
|  1 | 1944s |        1 |
|  2 | 1977  |        2 |
|  3 | 1955d |        1 |
|  4 | 1938  |        3 |
|  5 | 1964  |        4 |
|  6 | 1959  |        1 |
|  7 | 1945  |        3 |
|  8 | 1905  |        5 |
|  9 | 1924  |        6 |
| 10 | 2001  |        2 |
| 11 | 1937  |        6 |
| 12 | 1977  |        4 |
+----+-------+----------+
```

Если вам нужно вывести в результате запроса строки `coin` с названиями номиналов, а не их идентификаторами, выполните соединение, используя `denom` как справочную таблицу:

```
mysql> SELECT coin.id, coin.date, denom.name
    -> FROM coin, denom
    -> WHERE coin.denom_id = denom.denom_id;
+----+-------+--------------------+
| id | date  | name               |
+----+-------+--------------------+
|  1 | 1944s | Lincoln cent       |
|  2 | 1977  | Roosevelt dime     |
|  3 | 1955d | Lincoln cent       |
|  4 | 1938  | Jefferson nickel   |
|  5 | 1964  | Kennedy half dollar |
|  6 | 1959  | Lincoln cent       |
|  7 | 1945  | Jefferson nickel   |
|  8 | 1905  | Buffalo nickel     |
|  9 | 1924  | Mercury head dime  |
| 10 | 2001  | Roosevelt dime     |
| 11 | 1937  | Mercury head dime  |
| 12 | 1977  | Kennedy half dollar |
+----+-------+--------------------+
```

Это напоминает содержимое исходной таблицы `coin`, хотя таблица уже не хранит длинное описание в каждой строке.

А как насчет вставки новых элементов в таблицу `coin`? При работе с исходной таблицей вы бы вводили в каждую строку название номинала монеты. Теперь же, когда номинал преобразован в значения идентификаторов, используйте предложение `INSERT INTO ... SELECT` для поиска идентификатора номинала по названию. Например, для того чтобы ввести монету 10 центов (dime) 1962 года с Рузвельтом, используйте такое предложение:

```
INSERT INTO coin (date, denom_id)
SELECT 1962, denom_id FROM denom WHERE name = 'Roosevelt dime';
```

Это прием более подробно описан в рецепте 12.17.

12.20. Удаление связанных строк в нескольких таблицах

Задача

Вы хотите удалить связанные строки из нескольких таблиц. Например, если у вас есть таблицы, связанные как «главная-подчиненная» или «предок-потомок», то удаление родительской записи обычно требует удаления всех соответствующих дочерних записей.

Решение

Есть несколько вариантов решения задачи. MySQL 4.0 поддерживает каскадное удаление, разрешая многотабличный синтаксис DELETE. Можно заменять таблицу новыми версиями, которые содержат только те записи, которые *не* должны быть удалены. Можно написать программу, формирующую соответствующие предложения DELETE для каждой таблицы, или использовать для этого *mysql*.

Обсуждение

Приложения, использующие связанные таблицы, часто должны обрабатывать их одновременно. Предположим, что вы используете MySQL для записи информации о содержимом имеющихся у вас дистрибутивов программ. Главная (родительская) таблица хранит название дистрибутива, номер версии и дату выпуска. Подчиненная (или дочерняя) таблица приводит информацию о файлах дистрибутива, представляя собой ведомость содержимого дистрибутива. Для обеспечения соответствия родительской и дочерней записей в каждой родительской записи имеется уникальный идентификатор, который хранится и в дочерней записи. Таблицы можно определить так:

```
CREATE TABLE swdist_head
(
    dist_id     INT UNSIGNED NOT NULL AUTO_INCREMENT,   # идентификатор дистрибутива
    name        VARCHAR(40),                            # название дистрибутива
    ver_num     NUMERIC(5,2),                           # номер версии
    rel_date    DATE NOT NULL,                          # дата выпуска
    PRIMARY KEY (dist_id)
);
CREATE TABLE swdist_item
(
    dist_id     INT UNSIGNED NOT NULL,  # идентификатор родительского дистрибутива
    dist_file   VARCHAR(255) NOT NULL   # имя файла дистрибутива
);
```

В примерах данного раздела будем полагать, что таблицы содержат такие записи:

```
mysql> SELECT * FROM swdist_head ORDER BY name, ver_num;
+---------+------------+---------+------------+
| dist_id | name       | ver_num | rel_date   |
+---------+------------+---------+------------+
|       1 | DB Gadgets |    1.59 | 1996-03-25 |
|       3 | DB Gadgets |    1.60 | 1998-12-26 |
|       4 | DB Gadgets |    1.61 | 1998-12-28 |
|       2 | NetGizmo   |    3.02 | 1998-11-10 |
|       5 | NetGizmo   |    4.00 | 2001-08-04 |
+---------+------------+---------+------------+
mysql> SELECT * FROM swdist_item ORDER BY dist_id, dist_file;
+---------+---------------+
| dist_id | dist_file     |
+---------+---------------+
|       1 | db-gadgets.sh |
|       1 | README        |
|       2 | NetGizmo.exe  |
|       2 | README.txt    |
|       3 | db-gadgets.sh |
|       3 | README        |
|       3 | README.linux  |
|       4 | db-gadgets.sh |
|       4 | README        |
|       4 | README.linux  |
|       4 | README.solaris |
|       5 | NetGizmo.exe  |
|       5 | README.txt    |
+---------+---------------+
```

Таблицы описывают дистрибутивы трех версий DB Gadgets и двух версий NetGizmo. Но таблицы трудно воспринимать по отдельности, так что давайте используем соединение строк двух таблиц для вывода информации об указанном дистрибутиве. Например, следующий запрос выводит данные, хранимые для DB Gadgets версии 1.60:

```
mysql> SELECT swdist_head.dist_id, swdist_head.name,
    -> swdist_head.ver_num, swdist_head.rel_date, swdist_item.dist_file
    -> FROM swdist_head, swdist_item
    -> WHERE swdist_head.name = 'DB Gadgets' AND swdist_head.ver_num = 1.60
    -> AND swdist_head.dist_id = swdist_item.dist_id;
+---------+------------+---------+------------+---------------+
| dist_id | name       | ver_num | rel_date   | dist_file     |
+---------+------------+---------+------------+---------------+
|       3 | DB Gadgets |    1.60 | 1998-12-26 | README        |
|       3 | DB Gadgets |    1.60 | 1998-12-26 | README.linux  |
|       3 | DB Gadgets |    1.60 | 1998-12-26 | db-gadgets.sh |
+---------+------------+---------+------------+---------------+
```

Использование внешних ключей для обеспечения ссылочной целостности

СУБД предоставляет возможность, способствующую обеспечению ссылочной целостности (referential integrity) таблиц, – определение внешних ключей. То есть при создании таблицы можно явно указать в ее определении, что первичный ключ родительской таблицы (например, столбец `dist_id` таблицы `swdist_head`) является родителем ключа другой таблицы (столбца `dist_id` таблицы `swdist_item`). Определив столбец идентификаторов в дочерней таблице как внешний ключ для столбца идентификаторов родительской таблицы, можно наложить ограничения на выполнение незаконных операций. Например, можно предотвратить создание дочерней записи с идентификатором, не содержащимся в родительской таблице, или удаление родительских записей без предварительного удаления соответствующих дочерних. Внешний ключ также может обеспечивать каскадное удаление: если удаляется родительская запись, то процессор базы данных распространяет операцию удаления на все дочерние таблицы, и дочерние записи автоматически удаляются. Таблицы типа InnoDB в MySQL поддерживают внешние ключи, начиная с версии 3.23.44, а каскадное удаление – с версии 3.23.50. Кроме того, в MySQL 4.1 планируется реализовать поддержку внешних ключей для всех типов таблиц.

Аналогично при удалении дистрибутива необходимо обработать обе таблицы. DB Gadgets 1.60 имеет идентификатор 3, поэтому одним из способов избавления от него был бы запуск вручную таких предложений `DELETE` для каждой из таблиц:

```
mysql> DELETE FROM swdist_head WHERE dist_id = 3;
mysql> DELETE FROM swdist_item WHERE dist_id = 3;
```

Это просто и быстро, но если вы забудете о том, что необходимо выполнить удаление в двух таблицах (что гораздо вероятнее, чем вы можете подумать), то могут возникнуть проблемы. Нарушится ссылочная целостность: возникнут родительские записи без дочерних или дочерние записи, ссылающиеся на несуществующих родителей. Кроме того, ручное удаление неудобно в тех случаях, когда нужно удалить большое количество дистрибутивов или когда вы заранее не знаете, какие дистрибутивы необходимо удалить. Предположим, вы хотите произвести чистку и удалить все старые записи, оставив только последнюю версию каждого дистрибутива. (Например, из таблицы, содержащей данные о дистрибутивах DB Gadgets версий 1.59, 1.60 и 1.61, будут удалены записи для версий 1.59 и 1.60.) Вероятно, вы будете определять, какие именно дистрибутивы должны быть удалены, при помощи запроса, выбирающего идентификаторы старых дистрибутивов. Но что делать дальше? Запрос может вывести множество идентификаторов, и вы вряд ли захотите удалять каждый дистрибутив вручную. Этого и не нужно. Есть ряд возможностей удаления записей из нескольких таблиц:

- Использовать многотабличный синтаксис DELETE, доступный в версии MySQL 4.0.0. Вы сможете написать запрос, который занимается идентификацией и удалением записей двух таблиц одновременно. Не нужно помнить о необходимости запуска нескольких предложений DELETE при каждом удалении записей из связанных таблиц.
- Подойти к задаче с другой стороны: выбрать записи, которые не должны быть удалены, в новые таблицы, затем заменить исходные таблицы новыми. Результат будет таким же, как при удалении ненужных записей.
- Использовать программу для определения идентификаторов дистрибутивов, которые подлежат удалению, и формирования соответствующих предложений DELETE. Можно использовать самостоятельно написанную программу или уже существующую, например, *mysql*.

В оставшейся части раздела подробно описаны все упомянутые способы и их применение для решения задачи удаления старых дистрибутивов. Каждый пример будет удалять записи из таблиц swdist_head и swdist_item, поэтому необходимо создавать их и заполнять записями перед опробованием каждого из методов, чтобы всегда начинать работу с одной и той же точки. Для этого можно воспользоваться сценарием *swdist_create.sql* из каталога *joins* дистрибутива recipes. Сценарии, реализующие все способы удаления записей из нескольких таблиц, находятся в этом же каталоге.

Каскадное удаление при помощи многотабличного предложения DELETE

Начиная с MySQL версии 4.0.0 предложение DELETE поддерживает синтаксис, позволяющий идентифицировать записи для удаления в нескольких таблицах и удалить их все в одном предложении. Чтобы использовать эту возможность для удаления дистрибутивов программ из таблиц swdist_head и swdist_item, определим идентификаторы удаляемых дистрибутивов, затем применим список к таблицам.

Начнем с определения того, какая версия каждого дистрибутива является последней, и выберем названия и номера версий в отдельную таблицу. Следующий запрос выбирает название каждого дистрибутива и его последнюю версию:

```
mysql> CREATE TABLE tmp
    -> SELECT name, MAX(ver_num) AS newest
    -> FROM swdist_head
    -> GROUP BY name;
```

Получившаяся таблица выглядит так:

```
mysql> SELECT * FROM tmp;
+------------+--------+
| name       | newest |
+------------+--------+
| DB Gadgets |   1.61 |
```

```
| NetGizmo   |   4.00 |
+------------+--------+
```

Теперь определим идентификаторы тех дистрибутивов, которые старше перечисленных в таблице tmp:

```
mysql> CREATE TABLE tmp2
    -> SELECT swdist_head.dist_id, swdist_head.name, swdist_head.ver_num
    -> FROM swdist_head, tmp
    -> WHERE swdist_head.name = tmp.name AND swdist_head.ver_num < tmp.newest;
```

Обратите внимание на то, что на самом деле необходимо выбрать только столбец dist_id в tmp2. Пример выбирает и название, и номер версии, чтобы вы, посмотрев на tmp2, могли легко убедиться в том, что выбранные идентификаторы действительно соответствуют старым дистрибутивам, которые следует удалить:

```
mysql> SELECT * FROM tmp2;
+---------+------------+---------+
| dist_id | name       | ver_num |
+---------+------------+---------+
|       1 | DB Gadgets |    1.59 |
|       3 | DB Gadgets |    1.60 |
|       2 | NetGizmo   |    3.02 |
+---------+------------+---------+
```

Таблица не содержит идентификаторов для DB Gadgets 1.61 и NetGizmo 4.00, представляющих последние дистрибутивы.

Теперь применим список идентификаторов таблицы tmp2 к таблице дистрибутивов, используя многотабличное предложение DELETE. Общая форма этого предложения такова:

```
DELETE список_таблиц1 FROM список_таблиц2 WHERE условия;
```

В списке *список_таблиц1* перечислены имена таблиц, из которых нужно удалить записи. В списке *список_таблиц2* указаны таблицы, используемые в инструкции WHERE, задающей условия идентификации удаляемых записей. Каждый из списков таблиц может включать одно или несколько имен, разделенных запятыми. В нашем случае таблицы, из которых удаляются записи, – это swdist_head и swdist_item. Таблицы, используемые для идентификации удаляемых записей, – это те же swdist_head и swdist_item, а также tmp2:

```
mysql> DELETE swdist_head, swdist_item
    -> FROM tmp2, swdist_head, swdist_item
    -> WHERE tmp2.dist_id = swdist_head.dist_id
    -> AND tmp2.dist_id = swdist_item.dist_id;
```

Результирующие таблицы будут такими:

```
mysql> SELECT * FROM swdist_head;
+---------+------------+---------+------------+
| dist_id | name       | ver_num | rel_date   |
+---------+------------+---------+------------+
|       4 | DB Gadgets |    1.61 | 1998-12-28 |
```

```
|       5 | NetGizmo   |    4.00 | 2001-08-04 |
+---------+------------+---------+------------+
mysql> SELECT * FROM swdist_item;
+---------+---------------+
| dist_id | dist_file     |
+---------+---------------+
|       4 | README        |
|       4 | README.linux  |
|       4 | README.solaris |
|       4 | db-gadgets.sh |
|       5 | README.txt    |
|       5 | NetGizmo.exe  |
+---------+---------------+
```

В рассматриваемых таблицах предложение DELETE работает так, как и предполагалось. Но знайте, что если таблицы будут содержать родительские записи, которые следует удалить, но для которых нет соответствующих дочерних записей, то ничего не получится. Инструкция WHERE не найдет соответствий для родительской записи в дочерней и, следовательно, не выберет родительскую запись для удаления. Чтобы обеспечить выбор и удаление родительской записи даже при отсутствии у нее дочерних записей, используйте LEFT JOIN:

```
mysql> DELETE swdist_head, swdist_item
    -> FROM tmp2 LEFT JOIN swdist_head ON tmp2.dist_id = swdist_head.dist_id
    -> LEFT JOIN swdist_item ON swdist_head.dist_id = swdist_item.dist_id;
```

Соединение LEFT JOIN обсуждалось в рецепте 12.5.

Выполнение многотабличного удаления путем замены таблицы

Можно удалить связанные строки нескольких таблиц, выбрав только записи, которые *не* подлежат удалению, в новые таблицы и затем заменив исходные таблицы новыми. Это особенно удобно, если вы хотите удалить больше записей, чем останется для хранения.

Начнем с создания двух таблиц tmp_head и tmp_item, имеющих ту же структуру, что и таблицы swdist_head и swdist_item:

```
CREATE TABLE tmp_head
(
    dist_id     INT UNSIGNED NOT NULL AUTO_INCREMENT,   # идентификатор дистрибутива
    name        VARCHAR(40),                            # название дистрибутива
    ver_num     NUMERIC(5,2),                           # номер версии
    rel_date    DATE NOT NULL,                          # дата выпуска
    PRIMARY KEY (dist_id)
);
CREATE TABLE tmp_item
(
    dist_id     INT UNSIGNED NOT NULL,  # идентификатор родительского дистрибутива
    dist_file   VARCHAR(255) NOT NULL   # имя файла дистрибутива
);
```

Затем определим идентификаторы дистрибутивов, которые вы хотите хранить (то есть последнюю версию каждого дистрибутива). Идентификаторы определяются при помощи запросов, похожих на только что описанные в разделе про многотабличные удаления:

```
mysql> CREATE TABLE tmp
    -> SELECT name, MAX(ver_num) AS newest
    -> FROM swdist_head
    -> GROUP BY name;
mysql> CREATE TABLE tmp2
    -> SELECT swdist_head.dist_id
    -> FROM swdist_head, tmp
    -> WHERE swdist_head.name = tmp.name AND swdist_head.ver_num = tmp.newest;
```

Теперь выберем в новые таблицы те записи, которые должны остаться:

```
mysql> INSERT INTO tmp_head
    -> SELECT swdist_head.*
    -> FROM swdist_head, tmp2
    -> WHERE swdist_head.dist_id = tmp2.dist_id;
mysql> INSERT INTO tmp_item
    -> SELECT swdist_item.*
    -> FROM swdist_item, tmp2
    -> WHERE swdist_item.dist_id = tmp2.dist_id;
```

Наконец, заменим исходные таблицы новыми:

```
mysql> DROP TABLE swdist_head;
mysql> ALTER TABLE tmp_head RENAME TO swdist_head;
mysql> DROP TABLE swdist_item;
mysql> ALTER TABLE tmp_item RENAME TO swdist_item;
```

Выполнение многотабличного удаления из программы

Два предыдущих способа удаления связанных строк из нескольких таблиц используют только средства SQL. Другой подход заключается в написании программы, формирующей для вас предложения `DELETE`. Программа должна определить ключевые значения (идентификаторы дистрибутивов) для удаляемых записей и обработать ключи, преобразовав их в соответствующие предложения `DELETE`. Определять идентификаторы можно тем же способом, что и в предыдущих разделах, но у вас появляется некоторая свобода при выборе того, как использовать их для удаления записей:

- Обрабатывать каждый идентификатор отдельно. Формировать предложения `DELETE`, удаляющие записи из таблиц по одному идентификатору за раз.
- Обрабатывать идентификаторы как группу. Сформировать инструкцию `IN()`, содержащую все идентификаторы, и использовать ее для каждой таблицы для одновременного удаления всех совпадающих значений.
- Если список идентификаторов очень велик, разбить его на более мелкие группы для создания менее громоздких инструкций `IN()`.

- Можно решить задачу и подойдя к ней с другой стороны. Выберем идентификаторы тех дистрибутивов, которые следует сохранить, и используем их для формирования инструкции NOT IN(), удаляющей все остальные дистрибутивы. Обычно это менее эффективно, поскольку MySQL не применяет индекс для операций NOT IN().

Я покажу, как реализовать каждый из методов на Perl.

Для каждого из трех первых вариантов начнем с формирования списка идентификаторов дистрибутивов для удаляемых записей:

```
# Определение последней версии каждого дистрибутива

$dbh->do ("CREATE TABLE tmp
           SELECT name, MAX(ver_num) AS newest
           FROM swdist_head
           GROUP BY name");

# Определение идентификаторов более старых версий, чем указанная.

my $ref = $dbh->selectcol_arrayref (
           "SELECT swdist_head.dist_id
           FROM swdist_head, tmp
           WHERE swdist_head.name = tmp.name
           AND swdist_head.ver_num < tmp.newest");

# selectcol_arrayref() возвращает ссылку на список. Преобразуем ссылку в список,
# который будет пуст, если $ref - это undef или указывает на пустой список.

my @val = ($ref ? @{$ref} : ());
```

На данный момент @val содержит список значений идентификаторов удаляемых записей. Для их индивидуальной обработки выполните такой цикл:

```
# Используем список идентификаторов для удаления записей,
# по одному идентификатору единовременно.

foreach my $val (@val)
{
    $dbh->do ("DELETE FROM swdist_head WHERE dist_id = ?", undef, $val);
    $dbh->do ("DELETE FROM swdist_item WHERE dist_id = ?", undef, $val);
}
```

Цикл генерирует предложения вроде таких:

```
DELETE FROM swdist_head WHERE dist_id = '1'
DELETE FROM swdist_item WHERE dist_id = '1'
DELETE FROM swdist_head WHERE dist_id = '3'
DELETE FROM swdist_item WHERE dist_id = '3'
DELETE FROM swdist_head WHERE dist_id = '2'
DELETE FROM swdist_item WHERE dist_id = '2'
```

Недостаток этого способа в том, что для больших таблиц список идентификаторов может быть весьма объемистым, и придется формировать множество предложений DELETE. Для повышения эффективности объедините идентификаторы в одну инструкцию IN(), где укажите их все сразу. Сгенерируйте

список идентификаторов тем же способом, что и раньше, затем обрабатывайте список так:[1]

```
# Используем список идентификаторов для удаления записей по всем идентификаторам
# сразу. Если список пуст – не волнуемся, удалять нечего.

if (@val)
{
    # формируем список заполнителей "?", разделенных запятыми, по одному на значение
    my $where = "WHERE dist_id IN (" . join (",", ("?") x @val) . ")";
    $dbh->do ("DELETE FROM swdist_head $where", undef, @val);
    $dbh->do ("DELETE FROM swdist_item $where", undef, @val);
}
```

Теперь формируется всего одно предложение `DELETE` для каждой таблицы:

```
DELETE FROM swdist_head WHERE dist_id IN ('1','3','2')
DELETE FROM swdist_item WHERE dist_id IN ('1','3','2')
```

Если список идентификаторов *невероятно* велик, может возникнуть опасность формирования предложения `DELETE`, длина которого будет превышать максимальную разрешенную для запроса (по умолчанию – один мегабайт). В данном случае можно разбить список идентификаторов на мелкие группы и использовать каждую из них для построения короткой инструкции `IN()`:

```
# Используем список идентификаторов для удаления записей,
# используя единовременно часть списка.

my $grp_size = 1000;    # количество идентификаторов для одновременного удаления

for (my $i = 0; $i < @val; $i += $grp_size)
{
    my $j = (@val < $i + $grp_size ? @val : $i + $grp_size);
    my @group = @val[$i .. $j-1];
    # формируем список заполнителей "?", разделенных запятыми, по одному на значение
    my $where = "WHERE dist_id IN (" . join (",", ("?") x @group) . ")";
    $dbh->do ("DELETE FROM swdist_head $where", undef, @group);
    $dbh->do ("DELETE FROM swdist_item $where", undef, @group);
}
```

Все предыдущие программные методы находят идентификаторы удаляемых записей, затем удаляют их. Можно получить тот же результат, используя обратную логику: выбирать идентификаторы записей, которые планируется хранить, и удалять все остальное. Такой подход удобен в тех случаях, когда вы предполагаете, что останется меньше записей, чем будет удалено. Для реализации такой логики определяем новейшую версию каждого дистрибутива и находим соответствующие идентификаторы. Затем используем список идентификаторов для формирования инструкции `NOT IN()`:

---

[1] В Perl нельзя связать массив с заполнителем, но можно построить строку запроса так, чтобы она содержала нужное количество символов `?` (см. рецепт 2.6). Затем нужно передать массив для связывания в предложение, и каждый элемент будет связан с соответствующим заполнителем.

```
# Определяем новейшую версию для каждого дистрибутива

$dbh->do ("CREATE TABLE tmp
            SELECT name, MAX(ver_num) AS newest
            FROM swdist_head
            GROUP BY name");

# Определяем идентификаторы этих версий.

my $ref = $dbh->selectcol_arrayref (
            "SELECT swdist_head.dist_id
            FROM swdist_head, tmp
            WHERE swdist_head.name = tmp.name
            AND swdist_head.ver_num = tmp.newest");

# selectcol_arrayref() возвращает ссылку на список. Преобразуем ссылку в список,
# который будет пуст, если $ref – это undef или указывает на пустой список.

my @val = ($ref ? @{$ref} : ());

# используем список идентификаторов для удаления записей всех "остальных"
# идентификаторов сразу. Инструкция WHERE пуста, если пуст список
# (в этом случае ни одну запись не нужно хранить, все могут быть удалены).

my $where = "";
if (@val)
{
    # формируем список заполнителей "?", разделенных запятыми, по одному на значение
    $where = "WHERE dist_id NOT IN (" . join (",", ("?") x @val) . ")";
}
$dbh->do ("DELETE FROM swdist_head $where", undef, @val);
$dbh->do ("DELETE FROM swdist_item $where", undef, @val);
```

Обратите внимание, что при использовании такой обратной логики *необходимо* использовать весь список идентификаторов в одной инструкции `NOT IN()`. При попытке разбить список на более мелкие группы и использовать `NOT IN()` для каждой из них содержимое таблиц будет полностью уничтожено, даже если вы этого не хотите.

Выполнение многотабличного удаления с помощью mysql

Если ключи, указывающие, какие записи нужно удалить, не содержат кавычек и других специальных символов, то вы можете формировать предложения `DELETE` при помощи *mysql*. Ключи таблиц дистрибутивов (значения `dist_id`) целые, так что они допускают применение этого решения. Сформируйте список идентификаторов с помощью тех же запросов, которые описаны в разделе, посвященном многотабличному предложению `DELETE`, затем используйте этот список для создания предложений `DELETE`:

```
CREATE TABLE tmp
SELECT name, MAX(ver_num) AS newest
FROM swdist_head
```

```
GROUP BY name;

CREATE TABLE tmp2
SELECT swdist_head.dist_id
FROM swdist_head, tmp
WHERE swdist_head.name = tmp.name AND swdist_head.ver_num < tmp.newest;

SELECT CONCAT('DELETE FROM swdist_head WHERE dist_id=',dist_id,';') FROM tmp2;
SELECT CONCAT('DELETE FROM swdist_item WHERE dist_id=',dist_id,';') FROM tmp2;
```

Если эти предложения содержатся в файле *swdist_mysql_delete.sql*, выполните его, чтобы вывести ряд предложений DELETE:

```
% mysql -N cookbook < swdist_mysql_delete.sql > tmp
```

Файл *tmp* будет таким:

```
DELETE FROM swdist_head WHERE dist_id=1;
DELETE FROM swdist_head WHERE dist_id=3;
DELETE FROM swdist_head WHERE dist_id=2;
DELETE FROM swdist_item WHERE dist_id=1;
DELETE FROM swdist_item WHERE dist_id=3;
DELETE FROM swdist_item WHERE dist_id=2;
```

Выполним его содержимое так:

```
% mysql cookbook < tmp
```

12.21. Выявление и удаление несвязанных записей

Задача

У вас есть связанные таблицы (например, имеющие связь «главная-подчиненная»). Но вы подозреваете, что некоторые строки ни с чем не связаны и могут быть удалены.

Решение

Используйте LEFT JOIN для определения отсутствия соответствий и удалите выявленные значения, применяя приемы из рецепта 12.20. Или используйте процедуру замены таблицы, которая выбирает связанные записи в новую таблицу и заменяет ею исходную.

Обсуждение

В предыдущем разделе было рассказано о том, как одновременно удалять связанные записи из нескольких таблиц, используя связь, существующую между таблицами. Иногда возникает обратная задача: необходимо удалить записи на основе *отсутствия* связи. Такие ситуации обычно возникают, если у вас есть таблицы, которые предполагаются сопоставленными друг другу, но некоторые из записей не связаны ни с какими записями другой таблицы.

Это может произойти случайно, например, при удалении родительской записи без удаления соответствующих дочерних, или наоборот. А может быть и ожидаемым последствием какого-то умышленного действия. Предположим, что форум он-лайн использует родительскую таблицу, в которой перечислены все темы обсуждений, и дочернюю таблицу, в которой хранятся все сообщения по заданной теме. Если удалить из дочерней таблицы старые записи, это может привести к тому, что у какой-то родительской записи больше не будет ни одной дочерней. Отсутствие недавних сообщений по теме, вероятно, означает, что дискуссия затихла, и родительскую запись можно удалять из таблицы тем. Тогда вы удаляете все множество дочерних записей, четко осознавая, что операция может оставить родительские записи без дочерних и сделает их кандидатами на удаление.

Однако сейчас перед вами связанные таблицы с несвязанными записями. Для восстановления целостности таблиц необходимо выявить записи, не имеющие соответствий в связанной таблице, и удалить их:

- Для выявления несвязанных записей используйте `LEFT JOIN`, так как это задача отсутствия соответствий (см. рецепт 12.5).
- Для удаления записей, идентификаторы которых не соответствуют записям связанной таблицы, используйте для удаления записей из нескольких связанных таблиц приемы, подобные предложенным в рецепте 12.20.

В примерах будут использоваться таблицы дистрибутивов программного обеспечения `swdist_head` и `swdist_item` из рецепта 12.20. Создайте таблицы в их исходном состоянии, используя сценарий *swdist_create.sql* из каталога *joins* дистрибутива `recipes`. Они будут выглядеть так:

```
mysql> SELECT * FROM swdist_head;
+---------+------------+---------+------------+
| dist_id | name       | ver_num | rel_date   |
+---------+------------+---------+------------+
|       1 | DB Gadgets |    1.59 | 1996-03-25 |
|       2 | NetGizmo   |    3.02 | 1998-11-10 |
|       3 | DB Gadgets |    1.60 | 1998-12-26 |
|       4 | DB Gadgets |    1.61 | 1998-12-28 |
|       5 | NetGizmo   |    4.00 | 2001-08-04 |
+---------+------------+---------+------------+
mysql> SELECT * FROM swdist_item;
+---------+---------------+
| dist_id | dist_file     |
+---------+---------------+
|       1 | README        |
|       1 | db-gadgets.sh |
|       3 | README        |
|       3 | README.linux  |
|       3 | db-gadgets.sh |
|       4 | README        |
|       4 | README.linux  |
|       4 | README.solaris |
|       4 | db-gadgets.sh |
```

```
|       2 | README.txt     |
|       2 | NetGizmo.exe   |
|       5 | README.txt     |
|       5 | NetGizmo.exe   |
+---------+----------------+
```

На данный момент записи таблиц полностью соответствуют друг другу: для каждого значения dist_id из родительской таблицы существует как минимум одна дочерняя запись, а у каждой дочерней записи есть родительская. Чтобы нарушить целостность связи для наших целей, удалим несколько строк из каждой таблицы:

```
mysql> DELETE FROM swdist_head WHERE dist_id IN (1,4);
mysql> DELETE FROM swdist_item WHERE dist_id IN (2,5);
```

В результате в обеих таблицах появятся несвязанные записи:

```
mysql> SELECT * FROM swdist_head;
+---------+------------+---------+------------+
| dist_id | name       | ver_num | rel_date   |
+---------+------------+---------+------------+
|       2 | NetGizmo   |    3.02 | 1998-11-10 |
|       3 | DB Gadgets |    1.60 | 1998-12-26 |
|       5 | NetGizmo   |    4.00 | 2001-08-04 |
+---------+------------+---------+------------+
mysql> SELECT * FROM swdist_item;
+---------+----------------+
| dist_id | dist_file      |
+---------+----------------+
|       1 | README         |
|       1 | db-gadgets.sh  |
|       3 | README         |
|       3 | README.linux   |
|       3 | db-gadgets.sh  |
|       4 | README         |
|       4 | README.linux   |
|       4 | README.solaris |
|       4 | db-gadgets.sh  |
+---------+----------------+
```

Беглый осмотр показывает, что только для дистрибутива 3 есть записи в двух таблицах. Дистрибутивы 2 и 5 из таблицы swdist_head не сопоставлены никаким записям из таблицы swdist_item. И наоборот, дистрибутивам 1 и 4 таблицы swdist_item не соответствуют никакие записи из таблицы swdist_head.

Теперь необходимо выявить несвязанные записи (каким-то способом, отличным от визуального контроля) и удалить их. Это задача для LEFT JOIN. Например, чтобы найти родительские записи без дочерних в таблице swdist_head, используем такой запрос:

```
mysql> SELECT swdist_head.dist_id AS 'unmatched swdist_head IDs'
    -> FROM swdist_head LEFT JOIN swdist_item
    -> ON swdist_head.dist_id = swdist_item.dist_id
    -> WHERE swdist_item.dist_id IS NULL;
```

```
+---------------------------+
| unmatched swdist_head IDs |
+---------------------------+
|                         2 |
|                         5 |
+---------------------------+
```

И наоборот, чтобы найти идентификаторы «осиротевших» дочерних записей из таблицы `swdist_item`, поменяйте таблицы местами:

```
mysql> SELECT swdist_item.dist_id AS 'unmatched swdist_item IDs'
    -> FROM swdist_item LEFT JOIN swdist_head
    -> ON swdist_item.dist_id = swdist_head.dist_id
    -> WHERE swdist_head.dist_id IS NULL;
+---------------------------+
| unmatched swdist_item IDs |
+---------------------------+
|                         1 |
|                         1 |
|                         4 |
|                         4 |
|                         4 |
|                         4 |
+---------------------------+
```

Заметьте, что в данном случае идентификатор появится в списке несколько раз, если у отсутствующего родителя было несколько потомков. В зависимости от способа удаления несвязанных записей вы можете захотеть использовать `DISTINCT` для выбора идентификатора каждой несвязанной дочерней записи только единожды:

```
mysql> SELECT DISTINCT swdist_item.dist_id AS 'unmatched swdist_item IDs'
    -> FROM swdist_item LEFT JOIN swdist_head
    -> ON swdist_item.dist_id = swdist_head.dist_id
    -> WHERE swdist_head.dist_id IS NULL;
+---------------------------+
| unmatched swdist_item IDs |
+---------------------------+
|                         1 |
|                         4 |
+---------------------------+
```

После выявления несвязанных записей остается только избавиться от них. Можно применить один из способов, аналогичных представленным в рецепте 12.20:

- Использование идентификаторов в многотабличном предложении `DELETE`. Вы будете одновременно удалять строки только из одной таблицы, но синтаксис этой формы `DELETE` все же удобен, так как он позволяет идентифицировать удаляемые записи путем соединения связанных таблиц.
- Запуск программы, которая выбирает несвязанные идентификаторы и использует их для формирования предложений `DELETE`.

Чтобы использовать многотабличное предложение DELETE для удаления несвязанных записей, просто возьмите предложение SELECT, используемое для выявления этих записей, и замените все начало до ключевого слова FROM на DELETE *имя_таблицы*. Например, предложение SELECT, выбирающее родительские записи без дочерних, выглядит так:

```
SELECT swdist_head.dist_id AS 'unmatched swdist_head IDs'
FROM swdist_head LEFT JOIN swdist_item
    ON swdist_head.dist_id = swdist_item.dist_id
WHERE swdist_item.dist_id IS NULL;
```

Соответствующее предложение DELETE будет таким:

```
DELETE swdist_head
FROM swdist_head LEFT JOIN swdist_item
    ON swdist_head.dist_id = swdist_item.dist_id
WHERE swdist_item.dist_id IS NULL;
```

Запрос, определяющий потомков без родителей, выглядит так:

```
SELECT swdist_item.dist_id AS 'unmatched swdist_item IDs'
FROM swdist_item LEFT JOIN swdist_head
    ON swdist_item.dist_id = swdist_head.dist_id
WHERE swdist_head.dist_id IS NULL;
```

Соответствующее предложение DELETE удаляет их:

```
DELETE swdist_item
FROM swdist_item LEFT JOIN swdist_head
    ON swdist_item.dist_id = swdist_head.dist_id
WHERE swdist_head.dist_id IS NULL;
```

Чтобы удалить несвязанные записи из программы, выберите список идентификаторов и преобразуйте его в набор предложений DELETE. Рассмотрим программу на Perl, которая делает это сначала для родительской таблицы, а затем для дочерней:

```
#! /usr/bin/perl -w
use strict;
use lib qw(/usr/local/apache/lib/perl);
use Cookbook;

my $dbh = Cookbook::connect ();

# Определение идентификаторов родительских записей без дочерних

my $ref = $dbh->selectcol_arrayref (
            "SELECT swdist_head.dist_id
            FROM swdist_head LEFT JOIN swdist_item
                ON swdist_head.dist_id = swdist_item.dist_id
            WHERE swdist_item.dist_id IS NULL");

# selectcol_arrayref() возвращает ссылку на список. Преобразуем ссылку в список,
# который будет пуст, если $ref – это undef или указывает на пустой список.

my @val = ($ref ? @{$ref} : ());
```

```
# Используем список идентификаторов для удаления записей по всем идентификаторам
# сразу. Если список пуст – не волнуемся, удалять нечего.

if (@val)
{
    # формируем список заполнителей "?", разделенных запятыми, по одному на значение
    my $where = "WHERE dist_id IN (" . join (",", ("?") x @val) . ")";
    $dbh->do ("DELETE FROM swdist_head $where", undef, @val);
}

# Повторяем процедуру для дочерней таблицы. Используем SELECT DISTINCT,
# чтобы каждый идентификатор выбирался один раз.

$ref = $dbh->selectcol_arrayref (
            "SELECT DISTINCT swdist_item.dist_id
            FROM swdist_item LEFT JOIN swdist_head
                ON swdist_item.dist_id = swdist_head.dist_id
            WHERE swdist_head.dist_id IS NULL");

@val = ($ref ? @{$ref} : ());

if (@val)
{
    # формируем список заполнителей "?", разделенных запятыми, по одному на значение
    my $where = "WHERE dist_id IN (" . join (",", ("?") x @val) . ")";
    $dbh->do ("DELETE FROM swdist_item $where", undef, @val);
}

$dbh->disconnect ();

exit (0);
```

Программа использует `IN()` для удаления всех необходимых записей из указанной таблицы разом. Другие возможности удаления описаны в рецепте 12.20.

Для формирования предложений `DELETE` также можно использовать *mysql*; сценарий, показывающий, как это делается, приведен в каталоге *joins* дистрибутива `recipes`.

Другой подход к решению задачи реализуется процедурой замены таблицы. Этот метод подходит к вопросу с другой стороны. Вместо того чтобы находить и удалять несвязанные записи, будем находить и хранить связанные. Например, можно использовать соединение для выбора связанные записей в новую таблицу. Затем заменить ею исходную таблицу. Несвязанные записи не подхватываются соединением, то есть фактически удаляются при замене исходной таблицы новой.

Процедура замены таблицы работает так. Для таблицы `swdist_head` создаем новую таблицу с той же структурой:

```
CREATE TABLE tmp
(
    dist_id     INT UNSIGNED NOT NULL AUTO_INCREMENT,   # идентификатор дистрибутива
    name        VARCHAR(40),                            # название дистрибутива
```

```
    ver_num     NUMERIC(5,2),                          # номер версии
    rel_date    DATE NOT NULL,                         # дата выпуска
    PRIMARY KEY (dist_id)
);
```

Затем выбираем в таблицу `tmp` те записи `swdist_head`, для которых есть соответствия в таблице `swdist_item`:

```
INSERT IGNORE INTO tmp
SELECT swdist_head.*
FROM swdist_head, swdist_item
WHERE swdist_head.dist_id = swdist_item.dist_id;
```

Обратите внимание на то, что запрос использует `INSERT IGNORE`; родительская запись может соответствовать нескольким дочерним, но нам нужен только один экземпляр ее идентификатора. (Признаком неиспользования `IGNORE` является завершение запроса с ошибкой типа «duplicate key».)

Наконец, заменяем исходную таблицу новой:

```
DROP TABLE swdist_head;
ALTER TABLE tmp RENAME TO swdist_head;
```

Процедура замены дочерней таблицы таблицей, содержащей только связанные дочерние записи, аналогична, но без необходимости использования `IGNORE` – каждому потомку соответствует только один предок:

```
CREATE TABLE tmp
(
    dist_id     INT UNSIGNED NOT NULL,  # идентификатор родительского дистрибутива
    dist_file   VARCHAR(255) NOT NULL   # имя файла дистрибутива
);

INSERT INTO tmp
SELECT swdist_item.*
FROM swdist_head, swdist_item
WHERE swdist_head.dist_id = swdist_item.dist_id;

DROP TABLE swdist_item;
ALTER TABLE tmp RENAME TO swdist_item;
```

12.22. Одновременное использование нескольких серверов MySQL

Задача

Вы хотите выполнить запрос, который использует таблицы в базах данных, расположенных на разных серверах MySQL.

Решение

Эту задачу невозможно решить средствами только SQL. Можно обойти ее, открывая отдельные соединения с каждым сервером и самостоятельно свя-

зывая информацию из двух таблиц. Есть и другой способ – копировать одну из таблиц с одного сервера на другой, чтобы работать с обеими таблицами на одном сервере.

Обсуждение

На всем протяжении главы я неявно предполагал, что все таблицы, вовлеченные в операции с несколькими таблицами, хранятся на одном сервере MySQL. Если это предположение неверно, то работать с таблицами становится сложнее. Соединение с сервером MySQL определяется конкретным сервером. Вы не можете написать предложение SQL, ссылающееся на таблицы, хранящиеся на другом сервере. (Я встречал заявления о том, что это можно сделать, но всегда оказывалось, что они на самом деле ничем не подкреплены.)

Рассмотрим задачу на примере таблиц `artist` и `painting`. Предположим, что вы хотите определить названия картин да Винчи. Необходимо определить идентификатор да Винчи в таблице `artist` и сопоставить его записям таблицы `painting`. Если обе таблицы расположены в одной базе данных, то можно найти картины, используя запрос, выполняющий объединение таблиц:

```
mysql> SELECT painting.title
    -> FROM artist, painting
    -> WHERE artist.name = 'Da Vinci' AND artist.a_id = painting.a_id;
+-----------------+
| title           |
+-----------------+
| The Last Supper |
| The Mona Lisa   |
+-----------------+
```

Если таблицы относятся к разным базам данным, но управляются одним сервером MySQL, необходимо лишь слегка изменить запрос, включив спецификаторы базы данных (см. рецепт 12.2). Для наших таблиц запрос был бы таким:

```
mysql> SELECT db2.painting.title
    -> FROM db1.artist, db2.painting
    -> WHERE db1.artist.name = 'Da Vinci'
    -> AND db1.artist.a_id = db2.painting.a_id;
+-----------------+
| title           |
+-----------------+
| The Last Supper |
| The Mona Lisa   |
+-----------------+
```

Если же таблицы `artist` и `painting` находятся на разных серверах, невозможно выдать один запрос для выполнения соединения между ними. Вы должны отправить запрос на один сервер, чтобы извлечь идентификатор художника:

```
mysql> SELECT a_id FROM artist WHERE name = 'Da Vinci';
```

```
+------+
| a_id |
+------+
|    1 |
+------+
```

Затем использовать значение a_id (1) для формирования второго запроса, отправляемого другому серверу:

```
mysql> SELECT title FROM painting WHERE a_id = 1;
+-----------------+
| title           |
+-----------------+
| The Last Supper |
| The Mona Lisa   |
+-----------------+
```

Для простого примера предложено достаточно простое решение. Простота объясняется тем, что извлекается всего одно значение из первой таблицы, а информация выводится только из второй таблицы. Если же потребуется выводить фамилию художника вместе с названием картины, и выполнить такую операцию нужно будет для нескольких художников, то задача соответственно усложнится. Можно решить ее, написав программу, имитирующую соединение:

1. Открыть отдельное соединение с каждым сервером базы данных.
2. Запустить цикл, выбирающий идентификаторы и фамилии художников для сервера, хранящего таблицу artist.
3. При каждом проходе цикла использовать текущий идентификатор художника для формирования запроса, который ищет в таблице painting строки, соответствующие значению идентификатора. Отправить запрос серверу, управляющему таблицей painting. По мере извлечения названий картин выводить их вместе с фамилией текущего художника.

При таком подходе имитируется соединение между таблицами, расположенными на разных серверах. Кстати, этот же способ можно использовать при работе с таблицами разных СУБД. (Например, именно так вы можете имитировать соединение таблицы MySQL с таблицей PostgreSQL.) В этом есть, однако, некоторое трюкачество, так что в подобной ситуации вы можете предпочесть копирование одной из таблиц с одного сервера на другой. Тогда с таблицами можно будет работать как с расположенными на одном сервере, выполняя необходимое соединение между ними. О копировании таблиц с сервера на сервер рассказано в рецепте 10.16.

13

Статистические методы

13.0. Введение

В главе рассматривается использование основных статистических методов. Большая часть рецептов основана на информации из более ранних глав, например, на способах получения итоговой информации, описанных в главе 7. Приведенные примеры представляют дополнительные возможности использования материала тех глав. В общих чертах можно сказать, что поднимаемые вопросы делятся на три группы:

- Способы снятия характеристик данных, такие как вычисление описательных статистических показателей, вывод частотного распределения, подсчет отсутствующих значений, а также вычисление регрессий методом наименьших квадратов или коэффициентов корреляции.
- Способы рандомизации, такие как генерирование случайных чисел и применение их для рандомизации множества строк или для произвольного выбора элементов из строк.
- Присваивание рангов.

Статистические исследования затрагивают такой объем разнообразных тем, что эта глава может предложить только поверхностные сведения и охватить лишь несколько областей, в которых для статистического анализа можно применять MySQL. Обратите внимание на то, что некоторые статистические характеристики можно определять разными способами (например, как вы будете вычислять стандартное отклонение – для n степеней свободы или для $n-1$?). Поэтому если используемое мною определение какого-то термина не совпадает с вашими предпочтениями, вам придется несколько адаптировать приведенные запросы и алгоритмы.

Сценарии примеров главы находятся в каталоге *stats* дистрибутива `recipes`, а сценарии создания некоторых таблиц – в каталоге *tables*.

13.1. Получение описательных статистических показателей

Задача

Вы хотите охарактеризовать набор данных, используя общие описательные или сводные статистические показатели.

Решение

Многие общие описательные статистические показатели множества, такие как среднее и стандартное отклонение, можно получить, применяя к данным агрегирующие функции.

Обсуждение

Предположим, что у вас есть таблица `testscore` с результатами тестирования: идентификатор испытуемого (`subject`), возраст (`age`), пол (`sex`) и количество баллов (`score`):

```
mysql> SELECT subject, age, sex, score FROM testscore ORDER BY subject;
+---------+-----+-----+-------+
| subject | age | sex | score |
+---------+-----+-----+-------+
|       1 |   5 | M   |     5 |
|       2 |   5 | M   |     4 |
|       3 |   5 | F   |     6 |
|       4 |   5 | F   |     7 |
|       5 |   6 | M   |     8 |
|       6 |   6 | M   |     9 |
|       7 |   6 | F   |     4 |
|       8 |   6 | F   |     6 |
|       9 |   7 | M   |     8 |
|      10 |   7 | M   |     6 |
|      11 |   7 | F   |     9 |
|      12 |   7 | F   |     7 |
|      13 |   8 | M   |     9 |
|      14 |   8 | M   |     6 |
|      15 |   8 | F   |     7 |
|      16 |   8 | F   |    10 |
|      17 |   9 | M   |     9 |
|      18 |   9 | M   |     7 |
|      19 |   9 | F   |    10 |
|      20 |   9 | F   |     9 |
+---------+-----+-----+-------+
```

Анализ набора наблюдений разумно начать с получения некоторых описательных статистических показателей, обобщающих характеристики данных. Подобные статистические показатели включают:

- Количество наблюдений, сумма баллов и диапазон (минимум и максимум).

- Параметры сдвига (central tendency), такие как среднее значение (mean), медиана и мода.
- Параметры вариации, такие как стандартная девиация или дисперсия (variance).

Все эти величины, за исключением медианы и моды, могут быть получены путем вызова агрегирующих функций:

```
mysql> SELECT COUNT(score) AS n,
    -> SUM(score) AS sum,
    -> MIN(score) AS minimum,
    -> MAX(score) AS maximum,
    -> AVG(score) AS mean,
    -> STD(score) AS 'std. dev.'
    -> FROM testscore;
+----+------+---------+---------+--------+-----------+
| n  | sum  | minimum | maximum | mean   | std. dev. |
+----+------+---------+---------+--------+-----------+
| 20 |  146 |       4 |      10 | 7.3000 |    1.7916 |
+----+------+---------+---------+--------+-----------+
```

Агрегирующие функции в запросе обрабатывают только значения не-NULL наблюдений. Если вы используете NULL для представления отсутствующих значений, то можете захотеть получить дополнительно оценку того, как много значений отсутствует (см. рецепт 13.4).

Запрос не затрагивает дисперсию, в MySQL нет функции для ее вычисления. Но дисперсия – это просто квадрат стандартной девиации, так что ее легко получить так:

```
STD(score) * STD(score)
```

STDDEV() – это синоним STD().

Стандартную девиацию можно использовать для идентификации выбросов (outliers) – значений, которые расположены нехарактерно далеко от среднего. Например, чтобы выбрать значения, которые отделены от среднего более чем тремя стандартными девиациями, сделаем что-то типа:

```
SELECT @mean := AVG(score), @std := STD(score) FROM testscore;
SELECT score FROM testscore WHERE ABS(score-@mean) > @std * 3;
```

Для множества из *n* значений стандартная девиация, выдаваемая STD(), вычисляется для *n* степеней свободы. Это эквивалентно такому вычислению стандартной девиации без использования агрегирующей функции (@ss представляет сумму квадратов):

```
mysql> SELECT
    -> @n := COUNT(score),
    -> @sum := SUM(score),
    -> @ss := SUM(score*score)
    -> FROM testscore;
mysql> SELECT @var := ((@n * @ss) - (@sum * @sum)) / (@n * @n);
mysql> SELECT SQRT(@var);
```

```
+------------+
| SQRT(@var) |
+------------+
|   1.791647 |
+------------+
```

Чтобы получить стандартную девиацию для $n-1$ степеней свободы, сделайте следующее:

```
mysql> SELECT
    -> @n := COUNT(score),
    -> @sum := SUM(score),
    -> @ss := SUM(score*score)
    -> FROM testscore;
mysql> SELECT @var := ((@n * @ss) - (@sum * @sum)) / (@n * (@n - 1));
mysql> SELECT SQRT(@var);
+------------+
| SQRT(@var) |
+------------+
|   1.838191 |
+------------+
```

Или, что проще:

```
mysql> SELECT @n := COUNT(score) FROM testscore;
mysql> SELECT STD(score)*SQRT(@n/(@n-1)) FROM testscore;
+----------------------------+
| STD(score)*SQRT(@n/(@n-1)) |
+----------------------------+
|                   1.838191 |
+----------------------------+
```

В MySQL нет встроенных функций для вычисления моды и медианы набора данных, но вы можете получить их самостоятельно. Мода – это наиболее часто встречающееся значение. Чтобы выявить его, подсчитаем количество вхождений в множество каждого значения и посмотрим, какое из них самое большое:

```
mysql> SELECT score, COUNT(score) AS count
    -> FROM testscore GROUP BY score ORDER BY count DESC;
+-------+-------+
| score | count |
+-------+-------+
|     9 |     5 |
|     6 |     4 |
|     7 |     4 |
|     4 |     2 |
|     8 |     2 |
|    10 |     2 |
|     5 |     1 |
+-------+-------+
```

В данном случае модальным значением является 9.

Медиана множества упорядоченных значения вычисляется так:[1]

- Если значений нечетное количество, то медиана – это центральное значение.
- Если значений четное количество, то медиана – это среднее двух центральных значений множества.

На основе этого определения вычислим медиану множества наблюдений, хранящихся в базе данных:

- Создаем запрос для подсчета количества наблюдений. По полученному счетчику определяем, одно или два значения будут использованы для вычисления медианы, и каковы их индексы в упорядоченном множестве наблюдений.
- Создаем запрос с инструкцией `ORDER BY` для упорядочивания наблюдений и инструкцией `LIMIT` для извлечения центрального значения (значений).
- Вычисляем среднее для выбранного значения или значений.

Например, если таблица `t` содержит столбец `score` с 37 значениями (нечетное количество), необходимо выбрать одно значение, используя такой запрос:

```
SELECT score FROM t ORDER BY 1 LIMIT 18,1
```

Если столбец содержит 38 значений (четное количество), запрос будет таким:

```
SELECT score FROM t ORDER BY 1 LIMIT 18,2
```

Теперь выбираем значение или значения, возвращенные запросом, и вычисляем медиану как их среднее.

Приведем сценарий на Perl, реализующий вычисление медианы множества. Он принимает дескриптор базы данных и имена таблицы и столбца, содержащих наблюдения, затем генерирует запрос, извлекающий соответствующие значения, и возвращает их среднее:

```
sub median
{
my ($dbh, $tbl_name, $col_name) = @_;
my ($count, $limit);

    $count = $dbh->selectrow_array ("SELECT COUNT($col_name) FROM $tbl_name");
    return undef unless $count > 0;
    if ($count % 2 == 1)    # нечетное количество значений,
                            # вернуть центральное значение
    {
        $limit = sprintf ("LIMIT %d,1", ($count-1)/2);
    }
    else              # четное количество значений, вернуть два центральных значения
    {
```

[1] Обратите внимание, что приведенное определение медианы не является полным – в нем не говорится, что делать в случае, если в множестве содержатся дубликаты центральных значений.

```
        $limit = sprintf ("LIMIT %d,2", $count/2 - 1);
    }

    my $sth = $dbh->prepare (
                    "SELECT $col_name FROM $tbl_name ORDER BY 1 $limit");
    $sth->execute ();
    my ($n, $sum) = (0, 0);
    while (my $ref = $sth->fetchrow_arrayref ())
    {
        ++$n;
        $sum += $ref->[0];
    }
    return ($sum / $n);
}
```

Этот прием работает для множества значений, хранящихся в базе данных. Если же вы уже выбрали упорядоченное множество данных в массив `@val`, то медиану можно вычислить по-другому:

```
if (@val == 0)              # если массив пуст, то медиана не определена
{
    $median = undef;
}
elsif (@val % 2 == 1)       # если количество элементов массива нечетное,
                            # медиана – это его центральный элемент
{
    $median = $val[(@val-1)/2];
}
else                       # если количество элементов массива четное, медиана – это
                            # среднее значение двух его центральных элементов
    $median = ($val[@val/2 - 1] + $val[@val/2]) / 2;
}
```

Код написан для массивов, нумерация элементов в которых начинается с 0. Если же вы используете язык с индексом массивов, начинающимся с 1, то подкорректируйте алгоритм соответствующим образом.

13.2. Групповые описательные статистические показатели

Задача

Вы хотите получить описательные статистические показатели для каждой подгруппы множества наблюдений.

Решение

Применяйте агрегирующие функции, используя инструкцию `GROUP BY` для разбиения наблюдений на соответствующие группы.

Обсуждение

В предыдущем разделе было показано, как вычислять описательные статистические показатели для всего множества наблюдений таблицы `testscore`. Чтобы получить более точную информацию, можно использовать `GROUP BY` для распределения наблюдений по группам и вычисления статистических показателей для каждой группы в отдельности. Например, для испытуемых в таблице `testscore` приводятся такие параметры, как пол и возраст, так что можно вычислить статистические показатели для каждого возраста или пола (или сразу для того и другого):

По возрасту:

```
mysql> SELECT age, COUNT(score) AS n,
    -> SUM(score) AS sum,
    -> MIN(score) AS minimum,
    -> MAX(score) AS maximum,
    -> AVG(score) AS mean,
    -> STD(score) AS 'std. dev.'
    -> FROM testscore
    -> GROUP BY age;
+-----+---+------+---------+---------+--------+-----------+
| age | n | sum  | minimum | maximum | mean   | std. dev. |
+-----+---+------+---------+---------+--------+-----------+
|   5 | 4 |   22 |       4 |       7 | 5.5000 |    1.1180 |
|   6 | 4 |   27 |       4 |       9 | 6.7500 |    1.9203 |
|   7 | 4 |   30 |       6 |       9 | 7.5000 |    1.1180 |
|   8 | 4 |   32 |       6 |      10 | 8.0000 |    1.5811 |
|   9 | 4 |   35 |       7 |      10 | 8.7500 |    1.0897 |
+-----+---+------+---------+---------+--------+-----------+
```

По полу:

```
mysql> SELECT sex, COUNT(score) AS n,
    -> SUM(score) AS sum,
    -> MIN(score) AS minimum,
    -> MAX(score) AS maximum,
    -> AVG(score) AS mean,
    -> STD(score) AS 'std. dev.'
    -> FROM testscore
    -> GROUP BY sex;
+-----+----+------+---------+---------+--------+-----------+
| sex | n  | sum  | minimum | maximum | mean   | std. dev. |
+-----+----+------+---------+---------+--------+-----------+
| M   | 10 |   71 |       4 |       9 | 7.1000 |    1.7000 |
| F   | 10 |   75 |       4 |      10 | 7.5000 |    1.8574 |
+-----+----+------+---------+---------+--------+-----------+
```

По возрасту и полу:

```
mysql> SELECT age, sex, COUNT(score) AS n,
    -> SUM(score) AS sum,
    -> MIN(score) AS minimum,
```

```
    -> MAX(score) AS maximum,
    -> AVG(score) AS mean,
    -> STD(score) AS 'std. dev.'
    -> FROM testscore
    -> GROUP BY age, sex;
+-----+-----+---+------+---------+---------+--------+-----------+
| age | sex | n | sum  | minimum | maximum | mean   | std. dev. |
+-----+-----+---+------+---------+---------+--------+-----------+
|   5 | M   | 2 |    9 |       4 |       5 | 4.5000 |    0.5000 |
|   5 | F   | 2 |   13 |       6 |       7 | 6.5000 |    0.5000 |
|   6 | M   | 2 |   17 |       8 |       9 | 8.5000 |    0.5000 |
|   6 | F   | 2 |   10 |       4 |       6 | 5.0000 |    1.0000 |
|   7 | M   | 2 |   14 |       6 |       8 | 7.0000 |    1.0000 |
|   7 | F   | 2 |   16 |       7 |       9 | 8.0000 |    1.0000 |
|   8 | M   | 2 |   15 |       6 |       9 | 7.5000 |    1.5000 |
|   8 | F   | 2 |   17 |       7 |      10 | 8.5000 |    1.5000 |
|   9 | M   | 2 |   16 |       7 |       9 | 8.0000 |    1.0000 |
|   9 | F   | 2 |   19 |       9 |      10 | 9.5000 |    0.5000 |
+-----+-----+---+------+---------+---------+--------+-----------+
```

13.3. Получение частотного распределения

Задача

Вы хотите узнать частоту вхождения каждого значения в таблицу.

Решение

Сгенерируйте частотное распределение, которое является сводной характеристикой содержимого вашего множества данных.

Обсуждение

Методы вычисления сводных характеристик группы часто используются для разбиения значений на категории согласно количеству их вхождений во множество, то есть для получения частотного распределения. Для таблицы `testscore` частотное распределение выглядит так:

```
mysql> SELECT score, COUNT(score) AS occurrence
    -> FROM testscore GROUP BY score;
+-------+------------+
| score | occurrence |
+-------+------------+
|     4 |          2 |
|     5 |          1 |
|     6 |          4 |
|     7 |          4 |
|     8 |          2 |
|     9 |          5 |
|    10 |          2 |
+-------+------------+
```

Если выразить результаты в процентах от общего количества значений, то получится относительное частотное распределение. Для того чтобы разделить множество наблюдений на категории и вывести для каждой из них процент количества вхождений, используйте один запрос для получения общего количества наблюдений, а второй – для вычисления процента для каждой группы:

```
mysql> SELECT @n := COUNT(score) FROM testscore;
mysql> SELECT score, (COUNT(score)*100)/@n AS percent
    -> FROM testscore GROUP BY score;
+-------+---------+
| score | percent |
+-------+---------+
|     4 |      10 |
|     5 |       5 |
|     6 |      20 |
|     7 |      20 |
|     8 |      10 |
|     9 |      25 |
|    10 |      10 |
+-------+---------+
```

Приведенные распределения просто суммируют количество значений для каждого результата в баллах. Но если набор данных содержит большое количество несовпадающих значений, и вы хотите, чтобы распределение выводило только небольшое количество категорий, можно искусственно разбить значения на категории и вывести счетчик для каждой из них (см. приемы, описанные в рецепте 7.12).

Частотное распределение часто применяется при экспорте результатов в графическую программу. В отсутствие такой программы для визуального представления распределения вы можете сформировать простую ASCII-диаграмму и в MySQL. Например, чтобы вывести ASCII-гистограмму для счетчиков результатов тестов, преобразуйте счетчики в строки символов `*`:

```
mysql> SELECT score, REPEAT('*',COUNT(score)) AS occurrences
    -> FROM testscore GROUP BY score;
+-------+-------------+
| score | occurrences |
+-------+-------------+
|     4 | **          |
|     5 | *           |
|     6 | ****        |
|     7 | ****        |
|     8 | **          |
|     9 | *****       |
|    10 | **          |
+-------+-------------+
```

Для вывода диаграммы относительного частотного распределения используйте процентное соотношение:

```
mysql> SELECT @n := COUNT(score) FROM testscore;
mysql> SELECT score, REPEAT('*',(COUNT(score)*100)/@n) AS percent
    -> FROM testscore GROUP BY score;
+-------+--------------------------+
| score | percent                  |
+-------+--------------------------+
|     4 | **********               |
|     5 | *****                    |
|     6 | ********************     |
|     7 | ********************     |
|     8 | **********               |
|     9 | ************************ |
|    10 | **********               |
+-------+--------------------------+
```

Конечно, ASCII-диаграммы далеки от совершенства, но они обеспечивают быстрый способ получения общей картины распределения наблюдений без дополнительного инструментария.

Если вы формируете частотное распределение для набора категорий, часть которых не присутствует в ваших наблюдениях, то такие категории не будут отражены в выводе. Для того чтобы принудительно отобразить каждую категорию, используйте справочную таблицу и левое объединение (метод обсуждался в рецепте 12.9). Для таблицы `testscore` можно было бы ввести диапазон баллов от 0 до 10, тогда справочная таблица должна была бы содержать все значения диапазона:

```
mysql> CREATE TABLE ref (score INT);
mysql> INSERT INTO ref (score)
    -> VALUES(0),(1),(2),(3),(4),(5),(6),(7),(8),(9),(10);
```

Теперь объединяем справочную таблицу с результатами тестов и формируем частотное распределение:

```
mysql> SELECT ref.score, COUNT(testscore.score) AS occurrences
    -> FROM ref LEFT JOIN testscore ON ref.score = testscore.score
    -> GROUP BY ref.score;
+-------+-------------+
| score | occurrences |
+-------+-------------+
|     0 |           0 |
|     1 |           0 |
|     2 |           0 |
|     3 |           0 |
|     4 |           2 |
|     5 |           1 |
|     6 |           4 |
|     7 |           4 |
|     8 |           2 |
|     9 |           5 |
|    10 |           2 |
+-------+-------------+
```

Распределение содержит строки для результатов от 0 до 3, при том что ни один из них ранее не отображался в предыдущих частотных распределениях.

Тот же принцип используется и при получении относительных распределений частот:

```
mysql> SELECT @n := COUNT(score) FROM testscore;
mysql> SELECT ref.score, (COUNT(testscore.score)*100)/@n AS percent
    -> FROM ref LEFT JOIN testscore ON ref.score = testscore.score
    -> GROUP BY ref.score;
+-------+---------+
| score | percent |
+-------+---------+
|     0 |       0 |
|     1 |       0 |
|     2 |       0 |
|     3 |       0 |
|     4 |      10 |
|     5 |       5 |
|     6 |      20 |
|     7 |      20 |
|     8 |      10 |
|     9 |      25 |
|    10 |      10 |
+-------+---------+
```

13.4. Подсчет отсутствующих значений

Задача

Набор наблюдений не полон. Вы хотите понять – насколько.

Решение

Сосчитайте количество значений `NULL` в множестве.

Обсуждение

Значения могут отсутствовать в множестве по ряду причин: может быть, результаты еще не обработаны, может быть, что-то в ходе эксперимента пошло не так, и весь тест признается недействительным, и т. д. Вы можете представить подобные наблюдения значениями `NULL`, чтобы показать, что они или отсутствуют, или недействительны, а затем применить суммарный запрос для характеристики полноты множества данных.

Если таблица `t` содержит значения, которые нужно суммировать в одном измерении, то все отсутствующие значения можно охватить одним простым запросом. Предположим, что `t` выглядит так:

```
mysql> SELECT subject, score FROM t ORDER BY subject;
+---------+-------+
| subject | score |
```

```
+---------+-------+
|       1 |    38 |
|       2 |  NULL |
|       3 |    47 |
|       4 |  NULL |
|       5 |    37 |
|       6 |    45 |
|       7 |    54 |
|       8 |  NULL |
|       9 |    40 |
|      10 |    49 |
+---------+-------+
```

`COUNT(*)` вычисляет общее количество строк, а `COUNT(score)` – только количество действительных результатов. Их разность характеризует количество отсутствующих результатов, и отношение этой разности к общему количеству значений дает нам процент отсутствующих результатов. Вычисления будут такими:

```
mysql> SELECT COUNT(*) AS 'n (total)',
    -> COUNT(score) AS 'n (non-missing)',
    -> COUNT(*) - COUNT(score) AS 'n (missing)',
    -> ((COUNT(*) - COUNT(score)) * 100) / COUNT(*) AS '% missing'
    -> FROM t;
+-----------+-----------------+-------------+-----------+
| n (total) | n (non-missing) | n (missing) | % missing |
+-----------+-----------------+-------------+-----------+
|        10 |               7 |           3 |     30.00 |
+-----------+-----------------+-------------+-----------+
```

В качестве альтернативы вычислению количества значений `NULL` как разности двух счетчиков можно получить его напрямую, используя `SUM(ISNULL(score))`. Функция `ISNULL()` возвращает 1, если ее аргумент – `NULL`, и ноль в противном случае:

```
mysql> SELECT COUNT(*) AS 'n (total)',
    -> COUNT(score) AS 'n (non-missing)',
    -> SUM(ISNULL(score)) AS 'n (missing)',
    -> (SUM(ISNULL(score)) * 100) / COUNT(*) AS '% missing'
    -> FROM t;
+-----------+-----------------+-------------+-----------+
| n (total) | n (non-missing) | n (missing) | % missing |
+-----------+-----------------+-------------+-----------+
|        10 |               7 |           3 |     30.00 |
+-----------+-----------------+-------------+-----------+
```

Если значения разбиты на группы, то вхождения значений `NULL` могут оцениваться на уровне группы. Предположим, что `t` содержит баллы по предметам, распределенным по двум показателям, каждый из которых имеет два уровня:

```
mysql> SELECT subject, A, B, score FROM t ORDER BY subject;
```

```
+---------+------+------+-------+
| subject | A    | B    | score |
+---------+------+------+-------+
|       1 |    1 |    1 |    18 |
|       2 |    1 |    1 |  NULL |
|       3 |    1 |    1 |    23 |
|       4 |    1 |    1 |    24 |
|       5 |    1 |    2 |    17 |
|       6 |    1 |    2 |    23 |
|       7 |    1 |    2 |    29 |
|       8 |    1 |    2 |    32 |
|       9 |    2 |    1 |    17 |
|      10 |    2 |    1 |  NULL |
|      11 |    2 |    1 |  NULL |
|      12 |    2 |    1 |    25 |
|      13 |    2 |    2 |  NULL |
|      14 |    2 |    2 |    33 |
|      15 |    2 |    2 |    34 |
|      16 |    2 |    2 |    37 |
+---------+------+------+-------+
```

В данном случае запрос использует инструкцию `GROUP BY` для формирования итогов для каждой комбинации условий:

```
mysql> SELECT A, B, COUNT(*) AS 'n (total)',
    -> COUNT(score) AS 'n (non-missing)',
    -> COUNT(*) - COUNT(score) AS 'n (missing)',
    -> ((COUNT(*) - COUNT(score)) * 100) / COUNT(*) AS '% missing'
    -> FROM t
    -> GROUP BY A, B;
+------+------+-----------+-----------------+-------------+-----------+
| A    | B    | n (total) | n (non-missing) | n (missing) | % missing |
+------+------+-----------+-----------------+-------------+-----------+
|    1 |    1 |         4 |               3 |           1 |     25.00 |
|    1 |    2 |         4 |               4 |           0 |      0.00 |
|    2 |    1 |         4 |               2 |           2 |     50.00 |
|    2 |    2 |         4 |               3 |           1 |     25.00 |
+------+------+-----------+-----------------+-------------+-----------+
```

13.5. Вычисление линейной регрессии и коэффициентов корреляции

Задача

Вы хотите вычислить для двух переменных регрессию методом наименьших квадратов или коэффициент корреляции, который выражает отношение между ними.

Решение

Используйте агрегирующие функции.

Обсуждение

Если значения данных для двух переменных X и Y хранятся в базе данных, для них можно без труда получить регрессию методом наименьших квадратов, используя агрегирующие функции. Это же касается и коэффициента корреляции. На самом деле эти два вычисления во многом похожи, и для их получения требуется предварительный расчет одних и тех же величин.

Предположим, вы хотите вычислить регрессию методом наименьших квадратов, используя значения возраста и количество баллов, полученных при тестировании, из таблицы `testscore`:

```
mysql> SELECT age, score FROM testscore;
+-----+-------+
| age | score |
+-----+-------+
|   5 |     5 |
|   5 |     4 |
|   5 |     6 |
|   5 |     7 |
|   6 |     8 |
|   6 |     9 |
|   6 |     4 |
|   6 |     6 |
|   7 |     8 |
|   7 |     6 |
|   7 |     9 |
|   7 |     7 |
|   8 |     9 |
|   8 |     6 |
|   8 |     7 |
|   8 |    10 |
|   9 |     9 |
|   9 |     7 |
|   9 |    10 |
|   9 |     9 |
+-----+-------+
```

Линия регрессии задается уравнением, где `a` – отрезок, отсекаемый линией регрессии на оси Y, а `b` задает угол наклона линии:

```
Y = bX + a
```

Предположим, что возраст – это X, а результаты теста – Y, и начнем вычислять величины, необходимые для уравнения корреляции: количество тестов, средние значения, суммы и суммы квадратов для обеих переменных, а также сумму произведений для каждой переменной:[1]

```
mysql> SELECT
    -> @n := COUNT(score) AS N,
```

[1] Обратитесь к любому стандартному руководству по статистике, чтобы узнать, откуда и почему берутся все эти величины.

```
    -> @meanX := AVG(age) AS "X mean",
    -> @sumX := SUM(age) AS "X sum",
    -> @sumXX := SUM(age*age) "X sum of squares",
    -> @meanY := AVG(score) AS "Y mean",
    -> @sumY := SUM(score) AS "Y sum",
    -> @sumYY := SUM(score*score) "Y sum of square",
    -> @sumXY := SUM(age*score) AS "X*Y sum"
    -> FROM testscore\G
*************************** 1. row ***************************
              N: 20
         X mean: 7.0000
          X sum: 140
X sum of squares: 1020
         Y mean: 7.3000
          Y sum: 146
 Y sum of square: 1130
        X*Y sum: 1053
```

Теперь угловые коэффициенты корреляции можно получить так:

```
mysql> SELECT
    -> @b := (@n*@sumXY - @sumX*@sumY) / (@n*@sumXX - @sumX*@sumX)
    -> AS slope;
+-------+
| slope |
+-------+
| 0.775 |
+-------+
mysql> SELECT @a :=
    -> (@meanY - @b*@meanX)
    -> AS intercept;
+-----------+
| intercept |
+-----------+
|     1.875 |
+-----------+
```

А само уравнение корреляции будет таким:

```
mysql> SELECT CONCAT('Y = ',@b,'X + ',@a) AS 'least-squares regression';
+--------------------------+
| least-squares regression |
+--------------------------+
| Y = 0.775X + 1.875       |
+--------------------------+
```

Чтобы вычислить коэффициент корреляции, используем многие из уже полученных величин:

```
mysql> SELECT
    -> (@n*@sumXY - @sumX*@sumY)
    -> / SQRT((@n*@sumXX - @sumX*@sumX) * (@n*@sumYY - @sumY*@sumY))
    -> AS correlation;
```

```
+------------------+
| correlation      |
+------------------+
| 0.61173620442199 |
+------------------+
```

13.6. Генерация случайных чисел

Задача

Вам нужен источник случайных чисел.

Решение

Вызовите функцию MySQL `RAND()`.

Обсуждение

В MySQL есть функция `RAND()`, которая генерирует случайные числа в диапазоне от 0 до 1:

```
mysql> SELECT RAND(), RAND(), RAND();
+------------------+------------------+------------------+
| RAND()           | RAND()           | RAND()           |
+------------------+------------------+------------------+
| 0.31466114177803 | 0.89354679723601 | 0.52375059157959 |
+------------------+------------------+------------------+
```

Если указать функции целый аргумент, то `RAND()` использует заданное значение как начало последовательности случайных чисел. При каждой установке определенного значения в качестве начального `RAND()` будет генерировать повторяющиеся серии чисел:

```
mysql> SELECT RAND(1), RAND(), RAND();
+------------------+------------------+------------------+
| RAND(1)          | RAND()           | RAND()           |
+------------------+------------------+------------------+
| 0.18109050223705 | 0.75023211143001 | 0.20788908117254 |
+------------------+------------------+------------------+
mysql> SELECT RAND(20000000), RAND(), RAND();
+------------------+-------------------+------------------+
| RAND(20000000)   | RAND()            | RAND()           |
+------------------+-------------------+------------------+
| 0.24628307879556 | 0.020315642487552 | 0.36272900678472 |
+------------------+-------------------+------------------+
mysql> SELECT RAND(1), RAND(), RAND();
+------------------+------------------+------------------+
| RAND(1)          | RAND()           | RAND()           |
+------------------+------------------+------------------+
| 0.18109050223705 | 0.75023211143001 | 0.20788908117254 |
+------------------+------------------+------------------+
mysql> SELECT RAND(20000000), RAND(), RAND();
```

```
+------------------+-------------------+------------------+
| RAND(20000000)   | RAND()            | RAND()           |
+------------------+-------------------+------------------+
| 0.24628307879556 | 0.020315642487552 | 0.36272900678472 |
+------------------+-------------------+------------------+
```

Если вы хотите задавать начало последовательности случайных чисел случайным образом, выбирайте начальное значение RAND(), используя источник энтропии. Возможными источниками являются текущая временная метка и идентификатор соединения, по отдельности или в сочетании:

```
mysql> SELECT RAND(UNIX_TIMESTAMP()) AS rand1,
    -> RAND(CONNECTION_ID()) AS rand2,
    -> RAND(UNIX_TIMESTAMP()+CONNECTION_ID()) AS rand3;
+------------------+------------------+------------------+
| rand1            | rand2            | rand3            |
+------------------+------------------+------------------+
| 0.50452774158169 | 0.18113064782799 | 0.50456789089792 |
+------------------+------------------+------------------+
```

Однако если у вас есть другие источники начальных значений, лучше использовать их. Например, если в вашей системе имеется устройство */dev/random* или */dev/urandom*, то вы можете читать это устройство и использовать его для того, чтобы сгенерировать начальное значение для RAND().

13.7. Рандомизация набора строк

Задача

Вы хотите рандомизировать набор строк или значений.

Решение

Используйте инструкцию ORDER BY RAND().

Обсуждение

Функцию RAND() MySQL можно использовать для внесения элемента случайности в порядок возвращаемых запросом строк результирующего множества. Как это ни парадоксально, расположение в случайном порядке обеспечивается путем добавления в запрос инструкции ORDER BY. Метод имеет сходство с методом, применяемым в табличной процедуре рандомизации. Предположим, что в электронной таблице есть такой набор значений:

```
Patrick
Penelope
Pertinax
Polly
```

Чтобы расположить их в произвольном порядке, сначала добавим еще один столбец, содержащий произвольно выбранные числа:

```
Patrick   .73
Penelope  .37
Pertinax  .16
Polly     .48
```

Затем отсортируем строки по значениям случайных чисел:

```
Pertinax  .16
Penelope  .37
Polly     .48
Patrick   .73
```

Теперь исходные значения расположены в случайном порядке, так как результатом сортировки случайных чисел является рандомизация сопоставленных им значений. Чтобы повторно рандомизировать множество, выберите другой набор случайных чисел и еще раз отсортируйте строки.

В MySQL аналогичный эффект достигается путем сопоставления набора случайных чисел результатам запроса и сортировки результата по этим числам. Начиная с версии MySQL 3.23.2 операцию выполняет инструкция `ORDER BY RAND()`:

```
mysql> SELECT name FROM t ORDER BY RAND();
+----------+
| name     |
+----------+
| Pertinax |
| Penelope |
| Patrick  |
| Polly    |
+----------+
mysql> SELECT name FROM t ORDER BY RAND();
+----------+
| name     |
+----------+
| Patrick  |
| Pertinax |
| Penelope |
| Polly    |
+----------+
```

Для MySQL версий более ранних, чем 3.23.2, инструкция `ORDER BY` не может ссылаться на выражения, поэтому в ней нельзя использовать `RAND()` (см. рецепт 6.3). Чтобы разрешить проблему, добавьте в список столбцов вывода столбец случайных чисел, присвойте ему псевдоним и ссылайтесь на псевдоним при сортировке:

```
mysql> SELECT name, name*0+RAND() AS rand_num FROM t ORDER BY rand_num;
+----------+------------------+
| name     | rand_num         |
+----------+------------------+
| Penelope | 0.372227413926485 |
| Patrick  | 0.431537678867148 |
```

```
| Pertinax | 0.566524063764628 |
| Polly    | 0.715938107777329 |
+----------+-------------------+
```

Обратите внимание, что выражение для столбца случайных чисел выглядит как `name*0+RAND()`, а не просто как `RAND()`. Если попытаться использовать вто-

Насколько случайна функция RAND()?

Действительно ли функция `RAND()` генерирует числа, распределенные случайным образом? Проверьте это самостоятельно, используя предлагаемый сценарий на Python, *rand_test.py*, из каталога *stats* дистрибутива `recipes`. Он использует `RAND()` для генерирования случайных чисел и строит их частотное распределение для категорий размера 0.1. Тем самым оценивается, насколько произвольно распределены значения.

```
#! /usr/bin/python
# rand_test.py - создание частотного распределения значений RAND().
# Проверка на случайность RAND().

# Метод извлекает случайные числа в диапазоне от 0 до 1.0 и подсчитывает,
# сколько из них попадает в интервалы размера 0.1 (от 0 до 0.1,
# от 0.1 до 0.2, ..., от 0.9 *вплоть до* 1.0).

import sys
sys.path.insert (0, "/usr/local/apache/lib/python")
import MySQLdb
import Cookbook

npicks = 1000       # количество извлечений чисел

bucket = [0] * 10

conn = Cookbook.connect ()
cursor = conn.cursor ()

for i in range (0, npicks):
    cursor.execute ("SELECT RAND()")
    (val,) = cursor.fetchone ()
    slot = int (val * 10)
    if slot > 9:
        slot = 9        # поместить 1.0 в последний интервал
    bucket[slot] = bucket[slot] + 1

cursor.close ()
conn.close ()

# Вывести результирующее частотное распределение

for slot in range (0, 9):
    print "%2d  %d" % (slot+1, bucket[slot])

sys.exit (0)
```

В каталоге *stats* содержатся аналогичные сценарии на других языках.

рое, то оптимизатор версий MySQL до 3.23 заметит, что столбец содержит только функцию, предположит, что функция для каждой строки возвращает постоянное значение и с целью оптимизации исключит инструкцию ORDER BY. В результате сортировка не будет выполняться. Необходимо ввести оптимизатор в заблуждение, добавив в выражение дополнительные члены, не изменяющие его значение, но делающие столбец похожим на непостоянный. В представленном выше запросе приводится простой способ сделать это: берем имя любого столбца, умножаем на ноль и прибавляем результат к RAND(). Конечно, использование name в арифметическом выражении может показаться странным, так как значения этого столбца не числовые. Но это неважно; MySQL видит оператор умножения * и выполняет преобразование строки в число для значений name непосредственно перед умножением. Важно то, что результатом этого умножения является ноль, то есть выражение name*0+RAND() имеет то же значение, что и RAND().

Рандомизацию набора строк можно применять в любых сценариях, использующих выбор без возвращения в множество (выбор каждого элемента множества до тех пор, пока не останется ни одного элемента). Рассмотрим несколько примеров:

- Определение начального порядка участников некоторого мероприятия. Перечислите участников в таблице и выберите в случайном порядке.
- Назначение беговых дорожек или входов на скаковом поле для участников забега. Приведите список дорожек в таблице и выберите в случайном порядке.
- Выбор порядка представления вопросов викторины.
- Перемешивание колоды карт. Представьте каждую карту строкой таблицы и тасуйте колоду, выбирайте строки в случайном порядке. Сдавайте их по одной до тех пор, пока не закончится колода.

Чтобы использовать последний пример в качестве иллюстрации, давайте реализуем алгоритм перемешивания колоды карт. Перемешивание и сдача карт – это рандомизация и выбор без возвращения в множество: каждая карта выбирается единожды, прежде чем какая-то будет выбрана повторно; когда колода заканчивается, она заново перемешивается с целью повторной рандомизации для нового порядка сдачи. В программе задачу можно решить с помощью таблицы deck, содержащей 52 строки (набор карт 4 мастей, по 13 карт каждой масти):

- Выберите всю таблицу и сохраните ее в массиве.
- Каждый раз, когда нужна карта, берите следующий элемент массива.
- Когда массив исчерпан, все карты розданы. Заново «перемешайте» таблицу для генерирования нового порядка чисел.

Создание таблицы deck было бы весьма утомительным, если бы пришлось вставлять 52 записи о картах, написав все предложения INSERT вручную. Более простым способом получения содержимого deck будет комбинаторный – генерируем пары масть-достоинство карты. Приведем код PHP, который создает таблицу deck со столбцами face и suit, а затем заполняет ее данными

при помощи вложенных циклов, формирующих пары для предложений INSERT:

```
mysql_query ("
        CREATE TABLE deck
        (
            face    ENUM('A', 'K', 'Q', 'J', '10', '9', '8',
                         '7', '6', '5', '4', '3', '2') NOT NULL,
            suit    ENUM('hearts', 'diamonds', 'clubs', 'spades') NOT NULL
        )", $conn_id)
    or die ("Cannot issue CREATE TABLE statement\n");

$face_array = array ("A", "K", "Q", "J", "10", "9", "8",
                     "7", "6", "5", "4", "3", "2");
$suit_array = array ("hearts", "diamonds", "clubs", "spades");

# вставить "карту" в колоду для каждой комбинации масти и достоинства

reset ($face_array);
while (list ($index, $face) = each ($face_array))
{
    reset ($suit_array);
    while (list ($index2, $suit) = each ($suit_array))
    {
        mysql_query ("INSERT INTO deck (face,suit) VALUES('$face','$suit')",
                     $conn_id)
            or die ("Cannot insert card into deck\n");
    }
}
```

Для перемешивания карт создаем такое предложение:

```
SELECT face, suit FROM deck ORDER BY RAND();
```

Чтобы выполнить его и сохранить результаты в массиве, напишем функцию shuffle_deck(), которая запускает запрос и возвращает результирующие значения в массив (опять-таки на PHP):

```
function shuffle_deck ($conn_id)
{
    $query = "SELECT face, suit FROM deck ORDER BY RAND()";
    $result_id = mysql_query ($query, $conn_id)
        or die ("Cannot retrieve cards from deck\n");
    $card = array ();
    while ($obj = mysql_fetch_object ($result_id))
        $card[] = $obj;     # добавить запись о карте в конец массива $card
    mysql_free_result ($result_id);
    return ($card);
}
```

Раздаем карты, храня счетчик, принимающий значения от 0 до 51 и указывающий, какая карта выбирается. Когда счетчик достигает 52, это означает, что колода исчерпана, и необходимо перемешать ее заново.

13.8. Случайный выбор из набора строк

Задача

Вы хотите выбрать случайным образом элемент или элементы из набора значений.

Решение

Рандомизируйте значения, затем выберите первое из них (или несколько первых).

Обсуждение

Если набор значений хранится в MySQL, то вы можете произвольно извлечь одно из них так:

- Выбираем элементы в произвольном порядке при помощи `ORDER BY RAND()` (см. рецепт 13.7).
- Добавляем в запрос `LIMIT 1` для извлечения первого элемента.

Например, простой симулятор бросания костей можно реализовать путем создания таблицы `die`, строки которой содержат значения от 1 до 6, соответствующие шести граням кубика, и случайного выбора строк из нее:

```
mysql> SELECT n FROM die ORDER BY RAND() LIMIT 1;
+------+
| n    |
+------+
|    6 |
+------+
mysql> SELECT n FROM die ORDER BY RAND() LIMIT 1;
+------+
| n    |
+------+
|    4 |
+------+
mysql> SELECT n FROM die ORDER BY RAND() LIMIT 1;
+------+
| n    |
+------+
|    5 |
+------+
mysql> SELECT n FROM die ORDER BY RAND() LIMIT 1;
+------+
| n    |
+------+
|    4 |
+------+
```

По мере повторения операции вы выбираете случайную последовательность элементов множества. Это выбор с возвращением: элемент выбирается из со-

вокупности элементов, а затем возвращается в нее для следующей выборки. Так как элементы возвращаются на место, есть шанс выбрать один и тот же элемент несколько раз последовательно. Можно привести и другие примеры выбора с возвращением:

- Выбор баннера для показа на веб-странице.
- Выбор строки для «цитаты дня».
- Фокусы типа «Выберите карту, любую карту», каждый раз начинающиеся с полной колоды.

Если вы хотите выбрать не один, а несколько элементов, то измените аргумент `LIMIT`. Например, чтобы произвольным образом извлечь пять выигрышных строк из таблицы `drawing`, содержащей конкурсные записи, используйте `RAND()` в сочетании с `LIMIT`:

```
SELECT * FROM drawing ORDER BY RAND() LIMIT 5;
```

Особым случаем является извлечение одной строки из таблицы, про которую известно, что она содержит столбец со значениями в диапазоне от 1 до *n* в непрерывной последовательности. В данной ситуации можно избежать выполнения операции `ORDER BY` для всей таблицы, выбирая случайное число в данном диапазоне и выбирая соответствующую строку:

```
SET @id = FLOOR(RAND()*n)+1;
SELECT ... FROM имя_таблицы WHERE id = @id;
```

При увеличении размера таблицы второй вариант работает гораздо быстрее, чем `ORDER BY RAND() LIMIT 1`.

13.9. Присваивание рангов

Задача

Вы хотите присвоить ранги набору значений.

Решение

Выберите метод ранжирования, расположите элементы в нужном порядке и примените к ним выбранный метод.

Обсуждение

Для некоторых типов статистических проверок требуется присваивание рангов. Я опишу три метода ранжирования и покажу, как можно реализовать каждый из них при помощи переменных SQL. В примерах предполагается, что таблица `t` содержит следующие результаты, которые должны быть ранжированы по убыванию значений:

```
mysql> SELECT score FROM t ORDER BY score DESC;
```

```
+-------+
| score |
+-------+
|     5 |
|     4 |
|     4 |
|     3 |
|     2 |
|     2 |
|     2 |
|     1 |
+-------+
```

Один из способов ранжирования заключается просто в присвоении каждому значению номера соответствующей строки упорядоченного множества значений. Для получения такой классификации будем отслеживать номер строки и использовать его для текущего ранга:

```
mysql> SET @rownum := 0;
mysql> SELECT @rownum := @rownum + 1 AS rank, score
    -> FROM t ORDER BY score DESC;
+------+-------+
| rank | score |
+------+-------+
|    1 |     5 |
|    2 |     4 |
|    3 |     4 |
|    4 |     3 |
|    5 |     2 |
|    6 |     2 |
|    7 |     2 |
|    8 |     1 |
+------+-------+
```

Такая классификация не принимает в расчет возможность «ничейного счета» – вхождения одинаковых значений. Второй метод учитывает такую возможность, увеличивая ранг только при изменении значения:

```
mysql> SET @rank = 0, @prev_val = NULL;
mysql> SELECT @rank := IF(@prev_val=score,@rank,@rank+1) AS rank,
    -> @prev_val := score AS score
    -> FROM t ORDER BY score DESC;
+------+-------+
| rank | score |
+------+-------+
|    1 |     5 |
|    2 |     4 |
|    2 |     4 |
|    3 |     3 |
|    4 |     2 |
|    4 |     2 |
|    4 |     2 |
|    5 |     1 |
+------+-------+
```

Третий метод ранжирования – это комбинация первых двух. Ранги присваиваются по номеру строки, за исключением случаев с повторениями. Все одинаковые значения получают одинаковый ранг, равный номеру строки первого из значений. Для реализации этого метода будем отслеживать номер строки и предыдущее значение, увеличивая ранг до текущего номера строки при изменении значения:

```
mysql> SET @rownum = 0, @rank = 0, @prev_val = NULL;
mysql> SELECT @rownum := @rownum + 1 AS row,
    -> @rank := IF(@prev_val!=score,@rownum,@rank) AS rank,
    -> @prev_val := score AS score
    -> FROM t ORDER BY score DESC;
+------+------+-------+
| row  | rank | score |
+------+------+-------+
|    1 |    1 |     5 |
|    2 |    2 |     4 |
|    3 |    2 |     4 |
|    4 |    4 |     3 |
|    5 |    5 |     2 |
|    6 |    5 |     2 |
|    7 |    5 |     2 |
|    8 |    8 |     1 |
+------+------+-------+
```

Ранги так же легко присваивать и в программах. Например, следующий фрагмент кода PHP ранжирует результаты из таблицы `t`, используя третий метод:

```
$result_id = mysql_query ("SELECT score FROM t ORDER BY score DESC", $conn_id)
                or die ("Cannot select scores\n");
$rownum = 0;
$rank = 0;
unset ($prev_score);
print ("Row\tRank\tScore\n");
while (list ($score) = mysql_fetch_row ($result_id))
{
    ++$rownum;
    if ($rownum == 1 || $prev_score != $score)
        $rank = $rownum;
    print ("$rownum\t$rank\t$score\n");
    $prev_score = $score;
}
mysql_free_result ($result_id);
```

Третий способ ранжирования широко распространен вне области статистических расчетов. Вспомните, как в рецепте 3.18 мы использовали таблицу `al_winner`, содержащую информацию о 15 лучших подающих 2001 года из Американской Лиги :

```
mysql> SELECT name, wins FROM al_winner ORDER BY wins DESC, name;
```

```
+----------------+------+
| name           | wins |
+----------------+------+
| Mulder, Mark   |   21 |
| Clemens, Roger |   20 |
| Moyer, Jamie   |   20 |
| Garcia, Freddy |   18 |
| Hudson, Tim    |   18 |
| Abbott, Paul   |   17 |
| Mays, Joe      |   17 |
| Mussina, Mike  |   17 |
| Sabathia, C.C. |   17 |
| Zito, Barry    |   17 |
| Buehrle, Mark  |   16 |
| Milton, Eric   |   15 |
| Pettitte, Andy |   15 |
| Radke, Brad    |   15 |
| Sele, Aaron    |   15 |
+----------------+------+
```

С помощью третьего метода этим игрокам можно присвоить ранги следующим образом:

```
mysql> SET @rownum = 0, @rank = 0, @prev_val = NULL;
mysql> SELECT @rownum := @rownum + 1 AS row,
    -> @rank := IF(@prev_val!=wins,@rownum,@rank) AS rank,
    -> name,
    -> @prev_val := wins AS wins
    -> FROM al_winner ORDER BY wins DESC;
+------+------+----------------+------+
| row  | rank | name           | wins |
+------+------+----------------+------+
|    1 |    1 | Mulder, Mark   |   21 |
|    2 |    2 | Clemens, Roger |   20 |
|    3 |    2 | Moyer, Jamie   |   20 |
|    4 |    4 | Garcia, Freddy |   18 |
|    5 |    4 | Hudson, Tim    |   18 |
|    6 |    6 | Abbott, Paul   |   17 |
|    7 |    6 | Mays, Joe      |   17 |
|    8 |    6 | Mussina, Mike  |   17 |
|    9 |    6 | Sabathia, C.C. |   17 |
|   10 |    6 | Zito, Barry    |   17 |
|   11 |   11 | Buehrle, Mark  |   16 |
|   12 |   12 | Milton, Eric   |   15 |
|   13 |   12 | Pettitte, Andy |   15 |
|   14 |   12 | Radke, Brad    |   15 |
|   15 |   12 | Sele, Aaron    |   15 |
+------+------+----------------+------+
```

14

Обработка повторяющихся записей

14.0. Введение

Иногда в таблицах или в результирующих множествах встречаются повторяющиеся записи (дубликаты). В некоторых случаях это допустимо. Например, если вы проводите интернет-голосование, и вместе с его результатами записываете даты и IP-адреса проголосовавших, то дубликаты допустимы, так как большое количество голосов может быть подано с одного IP-номера интернет-провайдера, который пропускает трафик своих клиентов через один прокси-сервер. В других случаях дубликаты записей недопустимы, и вы хотели бы предпринять соответствующие меры. Операции, связанные с обработкой повторяющихся записей, могут быть такими:

- Подсчет дубликатов для определения того, присутствуют ли они и в каком количестве.
- Выявление дублирующихся значений (или содержащих их записей), чтобы посмотреть, что это за значения и где они появляются.
- Устранение дубликатов для обеспечения уникальности каждой записи. Может потребоваться удалить строки из таблицы, чтобы оставить в ней только уникальные записи. А можно выбрать результирующее множество так, чтобы в него не попали дубликаты. (Например, чтобы вывести список штатов, в которых у вас есть клиенты, вы вряд ли захотите выбирать весь длинный список названий штатов из всех клиентских записей. Будет достаточно перечня, в котором каждый штат появится лишь единожды, к тому же это удобнее для восприятия.)
- Предотвращение создания дубликатов в таблице. Если запись представляет единую сущность (например, человека, элемент каталога или определенное наблюдение в ходе эксперимента), то наличие дубликатов создает определенные трудности при использовании записей. Из-за повторений на некоторые записи таблицы невозможно сослаться однозначно, поэтому лучше сразу обеспечить их отсутствие.

Существует ряд инструментов для работы с дубликатами. Выбор завит от решаемой задачи:

- Создание таблицы с уникальным индексом предотвращает вставку дубликатов в таблицу. MySQL использует индекс для ввода в действие такого требования: каждая запись таблицы должна содержать уникальный ключ в индексированном столбце или столбцах.
- Предложения `INSERT IGNORE` и `REPLACE` в сочетании с уникальным индексом позволяют изящно обрабатывать вставку дубликатов без генерирования ошибки. При пакетной загрузке данных те же возможности обеспечивают модификаторы `IGNORE` и `REPLACE` предложения `LOAD DATA`.
- Если вы хотите определить, содержит ли таблица дубликаты, используйте `GROUP BY` для разбиения строк на группы и `COUNT()` для вывода количества строк в группе. Мы говорили о таком приеме в главе 7 в контексте формирования итогов, но он не менее полезен и для подсчета и выявления повторений. Как-никак, вычисление итогов – это не что иное, как операция распределения значений по категориям для определения того, как часто встречается каждое из них.
- `SELECT DISTINCT` полезно применять для устранения дубликатов из результирующего множества, чтобы оставить только уникальные записи. Добавление в таблицу уникального индекса может удалить имеющиеся дубликаты. Если вы определили, что в таблице *n* идентичных записей, то можете выполнить `DELETE ... LIMIT` для удаления *n*–1 экземпляров.

В этой главе описано применение всех перечисленных приемов выявления и устранения дубликатов. Прежде чем перейти к ним, давайте уточним, что мы будем понимать под «дубликатом». Говоря «дубликат» (повторение, повторяющаяся запись), люди подразумевают разные вещи. В данной главе одна запись считается дубликатом второй, если две строки содержат одинаковые значения в столбцах, которые должны их отличать. Рассмотрим такую таблицу:

```
mysql> SELECT * FROM person;
+------+-----------+------------+---------------+------+
| id   | last_name | first_name | address       | age  |
+------+-----------+------------+---------------+------+
|    1 | Smith     | Jim        | 428 Mill Road |   36 |
|    2 | Smith     | Joan       | 428 Mill Road |   36 |
|    3 | Smith     | Junior     | 428 Mill Road |   12 |
+------+-----------+------------+---------------+------+
```

Ни одна из этих записей не является дубликатом никакой другой, если сравнивать строки по всем столбцам, так как столбцы `id` и `first_name` этих строк содержат только уникальные значения. Однако если смотреть только на столбцы `last_name` и `address`, все столбцы содержат повторяющиеся значения. Между этими двумя крайностями находится результирующее множество на основе столбца `age`, в котором смешаны уникальные и повторяющиеся значения.

Сценарии примеров главы можно найти в каталоге *dups* дистрибутива `recipes`, а сценарии создания таблиц – в каталоге *tables*.

14.1. Предотвращение появления дубликатов в таблице

Задача

Вы хотите не допустить появления в таблице дубликатов, чтобы впоследствии не пришлось устранять их.

Решение

Используйте индекс PRIMARY KEY или UNIQUE.

Обсуждение

Чтобы обеспечить уникальность записей в таблице, необходимо потребовать, чтобы какой-то столбец или столбцы содержали уникальные значения в каждой строке. Если это условие выполнено, вы можете ссылаться на любую запись таблицы по ее уникальному идентификатору. Чтобы снабдить таблицу таким свойством, добавьте при создании таблицы в ее структуру индекс PRIMARY KEY или UNIQUE. Следующая таблица не содержит такого индекса, поэтому вставка дубликатов разрешена:

```
CREATE TABLE person
(
    last_name   CHAR(20),
    first_name  CHAR(20),
    address     CHAR(40)
);
```

Чтобы предотвратить создание в этой таблице записей с одинаковыми значениями имени и фамилии, добавьте в ее определение PRIMARY KEY. После этого необходимо еще объявить индексированные столбцы как NOT NULL, так как PRIMARY KEY не разрешает использовать значения NULL:

```
CREATE TABLE person
(
    last_name   CHAR(20) NOT NULL,
    first_name  CHAR(20) NOT NULL,
    address     CHAR(40),
    PRIMARY KEY (last_name, first_name)
);
```

Присутствие в таблице уникального индекса обычно приводит к ошибке при попытке вставки записи, дублирующей существующую в столбце или столбцах, определяющих индекс. В рецепте 14.2 рассказано о том, как поступать с такими ошибками и как изменить поведение MySQL в отношении обработки дубликатов.

Для обеспечения уникальности можно добавить в таблицу индекс UNIQUE, а не PRIMARY KEY. Эти два типа индекса идентичны во всем, кроме одного: UNIQUE может создаваться для столбцов, допускающих использование значений NULL.

В таблице person вы, вероятно, будете требовать заполнения столбцов как имени, так и фамилии. Тогда столбцы можно объявить как NOT NULL, и следующее объявление таблицы будет эквивалентно предыдущему:

```
CREATE TABLE person
(
    last_name   CHAR(20) NOT NULL,
    first_name  CHAR(20) NOT NULL,
    address     CHAR(40),
    UNIQUE (last_name, first_name)
);
```

Если индекс UNIQUE разрешает значения NULL, то NULL отличается от всех остальных значений тем, что может встречаться множество раз. Основная причина этого в том, что невозможно определить, совпадает одно неизвестное с другим или нет, поэтому разрешается использовать несколько таких значений.

Конечно, может случиться так, что таблица person, являющаяся отражением реального мира, в котором некоторых людей зовут одинаково, будет содержать повторяющиеся значения. Тогда вы не сможете ввести уникальный индекс для столбцов имен, так как необходимо разрешить дубликаты. Вместо этого каждому человеку присваиваем некоторый уникальный идентификатор, который становится значением, отличающим одну запись от другой. В MySQL в таких случаях обычно используется столбец AUTO_INCREMENT:

```
CREATE TABLE person
(
    id          INT UNSIGNED NOT NULL AUTO_INCREMENT,
    last_name   CHAR(20),
    first_name  CHAR(20),
    address     CHAR(40),
    PRIMARY KEY (id)
);
```

При создании записи, id которой содержит NULL, MySQL автоматически присваивает этому столбцу уникальный идентификатор. Есть и другая возможность – присваивать идентификаторы внешним образом и использовать эти идентификаторы как уникальные ключи. Например, граждане какой-то страны могут иметь идентификационные номера налогоплательщиков. Тогда можно применять такие номера как уникальные идентификаторы:

```
CREATE TABLE person
(
    tax_id      INT UNSIGNED NOT NULL,
    last_name   CHAR(20),
    first_name  CHAR(20),
    address     CHAR(40),
    PRIMARY KEY (tax_id)
);
```

См. также

Столбцы AUTO_INCREMENT подробно рассматриваются в главе 11.

14.2. Обработка дубликатов на этапе создания записи

Задача

Вы создали таблицу с уникальным индексом для предотвращения появления дубликатов значений в индексированном столбце или столбцах. Но теперь при попытке вставки записи с повторяющимся значением генерируется ошибка, а вы хотели бы избежать обработки таких ошибок.

Решение

Один из способов – просто игнорировать ошибку, другой – использовать предложение `INSERT IGNORE` или `REPLACE`, каждое из которых изменяет поведение MySQL в отношении обработки повторений. Для операции пакетной загрузки предложение `LOAD DATA` имеет модификаторы, позволяющие указать способ обработки дубликатов.

Обсуждение

По умолчанию MySQL генерирует ошибку при вставке записи, дублирующей существующий уникальный ключ. Например, если таблица `person` содержит уникальный индекс для столбцов `last_name` и `first_name`, то вы увидите следующее:

```
mysql> INSERT INTO person (last_name, first_name)
    -> VALUES('X1','Y1');
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO person (last_name, first_name)
    -> VALUES('X1','Y1');
ERROR 1062 at line 1: Duplicate entry 'X1-Y1' for key 1
```

Если вы интерактивно запускаете предложения из программы *mysql*, то можете просто сказать: «Понял, не сработало», игнорировать ошибку и продолжать работу. Но если вы пишете программу, вставляющую записи, то ошибка может привести к завершению ее работы. Один из способов избежать этого – изменить поведение программы за счет отлавливания ошибки и ее игнорирования. Сведения о приемах обработки ошибок приведены в рецепте 2.2.

Если вы хотите предотвратить появление ошибки, то, вероятно, подумываете о решении задачи обработки дубликатов с помощью двух запросов: запустите `SELECT`, чтобы определить, есть ли уже такая запись, а затем – `INSERT`, если записи еще нет. Но на самом деле ничего не получится. Другое клиентское приложение может вставить такую же запись в промежуток между вашими `SELECT` и `INSERT`, и тогда опять-таки сгенерируется ошибка. Чтобы это не произошло, можно заключить два предложения в транзакцию или заблокировать таблицы, но тогда вместо двух предложений у вас появится четыре. MySQL предлагает два решения задачи обработки дубликатов, каждое из которых состоит из единственного предложения:

- Используйте предложение INSERT IGNORE вместо INSERT. Если запись не дублирует существующую, то MySQL вставляет ее как обычно. Если же запись – это дубликат, то ключевое слово IGNORE указывает MySQL, что следует молча отбросить ее, не генерируя ошибку:

```
mysql> INSERT IGNORE INTO person (last_name, first_name)
    -> VALUES('X2','Y2');
Query OK, 1 row affected (0.00 sec)
mysql> INSERT IGNORE INTO person (last_name, first_name)
    -> VALUES('X2','Y2');
Query OK, 0 rows affected (0.00 sec)
```

  Значение счетчика строк показывает, была запись вставлена или проигнорирована. В программе вы можете получить это значение, используя функцию подсчета обработанных строк, имеющуюся в вашем API (см. рецепты 2.4 и 9.1).

- Используйте предложение REPLACE вместо INSERT. Если запись новая, она вставляется так, как если бы выполнялось предложение INSERT. Если же это дубликат, то новая запись замещает старую:

```
mysql> REPLACE INTO person (last_name, first_name)
    -> VALUES('X3','Y3');
Query OK, 1 row affected (0.00 sec)
mysql> REPLACE INTO person (last_name, first_name)
    -> VALUES('X3','Y3');
Query OK, 2 rows affected (0.00 sec)
```

  Значение количества обработанных строк во втором случае равно 2, так как исходная запись удалена, а на ее место вставлена новая запись.

Выбор INSERT IGNORE или REPLACE зависит от того, какое поведение для вас предпочтительно. INSERT IGNORE хранит первую из множества повторяющихся записей и удаляет остальные. REPLACE хранит последний из дубликатов и удаляет все остальные. Предложение INSERT IGNORE эффективнее, чем REPLACE, так как дубликаты не вставляются в таблицу. То есть его лучше применять, когда вы просто хотите убедиться в том, что копия указанной записи содержится в таблице. С другой стороны, REPLACE больше подходит для таблиц, в которых может потребоваться обновление других столбцов, не входящих в ключ. Предположим, что у вас есть таблица passtbl, используемая в веб-приложении для хранения адресов электронной почты и паролей, в которой адрес является ключом:

```
CREATE TABLE passtbl
(
    email       CHAR(60) NOT NULL,
    password    CHAR(20) BINARY NOT NULL,
    PRIMARY KEY (email)
);
```

Как создавать записи для новых пользователей и изменять пароли для существующих? Без REPLACE создание нового пользователя и изменение пароля

существующего обрабатывались бы по-разному. Стандартный алгоритм мог бы быть таким:

- Запустить SELECT, чтобы проверить, существует ли уже запись с указанным значением email.
- Если такой записи нет, добавить новую при помощи INSERT.
- Если запись существует, обновить ее при помощи UPDATE.

Все это можно выполнить внутри транзакции или заблокировав таблицы, чтобы запретить другим пользователям изменять таблицы в течение того времени, пока вы с ними работаете. Применив REPLACE, вы можете свести оба случая к одному предложению:

```
REPLACE INTO passtbl (email,password) VALUES(адрес,пароль);
```

Если запись с указанным адресом электронной почты не существует, то MySQL создает новую. Если запись существует, MySQL заменяет ее; в результате обновляется столбец password записи, содержащей данный адрес.

Преимущество INSERT IGNORE и REPLACE в том, что они не требуют дополнительных расходов, которые вызвала бы транзакция. Но за это преимущество приходится платить переносимостью, так как оба предложения являются специфичными для MySQL. Если для вас важна переносимость, то предпочтительнее использовать транзакцию.

Для операций пакетной загрузки, в которых предложение LOAD DATA используется для загрузки набора записей из файла в таблицу, обработку дубликатов можно регулировать модификаторами IGNORE и REPLACE предложения. Они обеспечивают поведение, аналогичное использованию предложений INSERT IGNORE и REPLACE (см. рецепт 10.7).

14.3. Подсчет и выявление дубликатов

Задача

Вы хотите узнать, содержит ли таблица дубликаты, и насколько их много. Или хотите увидеть записи, содержащие повторяющиеся значения.

Решение

Используйте счетчик (суммарный запрос), который ищет и выводит дубликаты. Чтобы увидеть записи, в которых встречаются дубликаты, объедините результат с исходной таблицей для вывода совпадающих записей.

Обсуждение

Предположим, что ваш веб-сайт содержит страницу регистрации, позволяющую пользователям добавить себя в список рассылки для получения периодических сообщений с каталогами товаров. Но вы забыли при создании таблицы включить в нее уникальный индекс, и теперь вы полагаете, что некоторые пользователи подписались на рассылку по несколько раз. Может

быть, они забыли, что уже присутствуют в списке, а может быть кто-то решил добавить своих друзей, которые уже подписались сами. В любом случае благодаря наличию повторяющихся записей вы дважды отправите каталоги. Это дополнительные расходы для вас и неудобство для получателей. В этом разделе рассказано о том, как определить, есть ли в таблице дубликаты, много ли их и как вывести повторяющиеся записи. (В рецепте 14.6 описано, как удалить дубликаты из таблицы, которая их содержит.)

Чтобы определить, есть ли в таблице дубликаты, используйте функции получения итоговой информации, описанные в главе 7. Методы получения итогов могут оказаться полезными при выявлении и подсчете дубликатов: группируем записи с помощью `GROUP BY` и подсчитываем строки в каждой группе с помощью `COUNT()`. Например, предположим, что получатели каталога перечислены в таблице `cat_mailing`:

```
mysql> SELECT * FROM cat_mailing;
+-----------+-------------+--------------------------+
| last_name | first_name  | street                   |
+-----------+-------------+--------------------------+
| Isaacson  | Jim         | 515 Fordam St., Apt. 917 |
| Baxter    | Wallace     | 57 3rd Ave.              |
| McTavish  | Taylor      | 432 River Run            |
| Pinter    | Marlene     | 9 Sunset Trail           |
| BAXTER    | WALLACE     | 57 3rd Ave.              |
| Brown     | Bartholomew | 432 River Run            |
| Pinter    | Marlene     | 9 Sunset Trail           |
| Baxter    | Wallace     | 57 3rd Ave., Apt 102     |
+-----------+-------------+--------------------------+
```

Предположим, что вы хотите выявить дубликаты для столбцов `last_name` и `first_name`, то есть получатели с одинаковыми именами принимаются за одного и того же человека (естественно, это упрощенная картина). Приведем запросы, часто используемые для описания таблицы и оценки существования и количества дубликатов:

- Общее количество строк таблицы:

```
mysql> SELECT COUNT(*) AS rows FROM cat_mailing;
+------+
| rows |
+------+
|    8 |
+------+
```

- Количество различных имен:

```
mysql> SELECT COUNT(DISTINCT last_name, first_name) AS 'distinct names'
    -> FROM cat_mailing;
+----------------+
| distinct names |
+----------------+
|              5 |
+----------------+
```

- Количество строк, содержащих дубликаты:

```
mysql> SELECT COUNT(*) - COUNT(DISTINCT last_name, first_name)
    -> AS 'duplicate names'
    -> FROM cat_mailing;
+-----------------+
| duplicate names |
+-----------------+
|               3 |
+-----------------+
```

- Доля записей с уникальными и неуникальными значениями:

```
mysql> SELECT COUNT(DISTINCT last_name, first_name) / COUNT(*)
    -> AS 'unique',
    -> 1 - (COUNT(DISTINCT last_name, first_name) / COUNT(*))
    -> AS 'non-unique'
    -> FROM cat_mailing;
+--------+------------+
| unique | non-unique |
+--------+------------+
|   0.62 |       0.38 |
+--------+------------+
```

Эти запросы характеризуют меру распространения дубликатов в множестве, но не показывают, какие именно значения повторяются. Чтобы увидеть повторяющиеся записи таблицы `cat_mailing`, используйте суммарный запрос, выводящий неуникальные записи вместе с соответствующим счетчиком:

```
mysql> SELECT COUNT(*) AS repetitions, last_name, first_name
    -> FROM cat_mailing
    -> GROUP BY last_name, first_name
    -> HAVING repetitions > 1;
+-------------+-----------+------------+
| repetitions | last_name | first_name |
+-------------+-----------+------------+
|           3 | Baxter    | Wallace    |
|           2 | Pinter    | Marlene    |
+-------------+-----------+------------+
```

Запрос содержит инструкцию `HAVING`, ограничивающую вывод только теми именами, которые встречаются несколько раз. (Если опустить эту инструкцию, то будут выведены и уникальные имена, которые нам не нужны, так как мы интересуемся только дубликатами.) Вообще, чтобы идентифицировать множество повторяющихся значений, сделайте следующее:

- Определите, какие столбцы содержат значения, которые могут повторяться.
- Перечислите эти столбцы в списке выбора, добавив `COUNT(*)`.
- Перечислите эти столбцы и в инструкции `GROUP BY`.
- Добавьте инструкцию `HAVING`, удаляющую уникальные значения, требуя, чтобы возвращенные групповые счетчики были больше единицы.

Построенные по этому плану запросы имеют такую форму:

```
SELECT COUNT(*), список_столбцов
FROM имя_таблицы
GROUP BY список_столбцов
HAVING COUNT(*) > 1
```

Можно без труда генерировать подобные запросы поиска дубликатов в программе для указанного имени таблицы и непустого множества имен столбцов. Рассмотрим, например, функцию на Perl `make_dup_count_query()`, которая формирует запрос для нахождения и подсчета дубликатов в указанных столбцах:

```
sub make_dup_count_query
{
my ($tbl_name, @col_name) = @_;

    return (
        "SELECT COUNT(*)," . join (",", @col_name)
        . "\nFROM $tbl_name"
        . "\nGROUP BY " . join (",", @col_name)
        . "\nHAVING COUNT(*) > 1"
    );
}
```

Функция `make_dup_count_query()` возвращает запрос в виде строки. Если вызвать ее так:

```
$str = make_dup_count_query ("cat_mailing", "last_name", "first_name");
```

то результирующим значением `$str` будет:

```
SELECT COUNT(*),last_name,first_name
FROM cat_mailing
GROUP BY last_name,first_name
HAVING COUNT(*) > 1
```

Теперь, когда строка запроса перед вами, можно выполнить запрос из создавшего строку сценария, передать другой программе или записать в файл для последующего выполнения. Каталог *dups* дистрибутива `recipes` содержит сценарий *dup_count.pl*, который можно использовать для опробования функции (там же можно найти и варианты на других языках). Далее в этой главе в рецепте 14.6 будет реализовано удаление дубликатов с использованием функции `make_dup_count_query()`.

Методы получения итогов позволяют оценить наличие дубликатов, их количество и вывести повторяющиеся значения. Но сам по себе суммарный запрос не может вывести полное содержимое записей, содержащих повторяющиеся значения. (Например, приведенные ранее суммарные запросы отображают счетчики повторяющихся имен из таблицы `cat_mailing` или сами имена, но не адреса, соответствующие этим именам.) Чтобы увидеть исходные записи, содержащие дубликаты, соедините итоговую информацию с исходной таблицей. Следующий пример показывает, как вывести записи таб-

лицы cat_mailing, содержащие повторяющиеся имена. Итоговая информация записывается во временную таблицу, которая затем объединяется с таблицей cat_mailing для вывода записей, соответствующих этим именам:

```
mysql> CREATE TABLE tmp
    -> SELECT COUNT(*) AS count, last_name, first_name
    -> FROM cat_mailing GROUP BY last_name, first_name HAVING count > 1;
mysql> SELECT cat_mailing.*
    -> FROM tmp, cat_mailing
    -> WHERE tmp.last_name = cat_mailing.last_name
    -> AND tmp.first_name = cat_mailing.first_name
    -> ORDER BY last_name, first_name;
+-----------+------------+----------------------+
| last_name | first_name | street               |
+-----------+------------+----------------------+
| Baxter    | Wallace    | 57 3rd Ave.          |
| BAXTER    | WALLACE    | 57 3rd Ave.          |
| Baxter    | Wallace    | 57 3rd Ave., Apt 102 |
| Pinter    | Marlene    | 9 Sunset Trail       |
| Pinter    | Marlene    | 9 Sunset Trail       |
+-----------+------------+----------------------+
```

Выявление дубликатов и чувствительность строк к регистру

Символьные строки, отличающиеся регистром, при сравнении считаются одинаковыми. Чтобы интерпретировать их как разные, используйте ключевое слово BINARY, которое сделает их чувствительными к регистру.

14.4. Устранение дубликатов из результата запроса

Задача

Вы хотите выбрать строки в результирующее множество так, чтобы оно не содержало дубликатов.

Решение

Используйте SELECT DISTINCT.

Обсуждение

Строки результирующего множества могут содержать дубликаты. Обычно так бывает при выборе только подмножества столбцов таблицы, так как уменьшается количество информации, которая могла бы отличать одну строку от другой. Чтобы результат содержал только уникальные строки, удалите

дубликаты, добавив ключевое слово DISTINCT, которое указывает MySQL на необходимость возврата одного экземпляра для каждого набора значений столбцов. Например, если вы выбираете названия столбцов из таблицы cat_mailing, не используя DISTINCT, то выводится несколько дубликатов:

```
mysql> SELECT last_name, first_name
    -> FROM cat_mailing ORDER BY last_name, first_name;
+-----------+-------------+
| last_name | first_name  |
+-----------+-------------+
| Baxter    | Wallace     |
| BAXTER    | WALLACE     |
| Baxter    | Wallace     |
| Brown     | Bartholomew |
| Isaacson  | Jim         |
| McTavish  | Taylor      |
| Pinter    | Marlene     |
| Pinter    | Marlene     |
+-----------+-------------+
```

Если добавить DISTINCT, дубликаты исчезают:

```
mysql> SELECT DISTINCT last_name, first_name
    -> FROM cat_mailing ORDER BY last_name;
+-----------+-------------+
| last_name | first_name  |
+-----------+-------------+
| Baxter    | Wallace     |
| Brown     | Bartholomew |
| Isaacson  | Jim         |
| McTavish  | Taylor      |
| Pinter    | Marlene     |
+-----------+-------------+
```

Альтернативой инструкции DISTINCT является инструкция GROUP BY, содержащая имена выбираемых столбцов. В результате удаляются дубликаты и выводятся только уникальные комбинации значений указанных столбцов:

```
mysql> SELECT last_name, first_name FROM cat_mailing
    -> GROUP BY last_name, first_name;
+-----------+-------------+
| last_name | first_name  |
+-----------+-------------+
| Baxter    | Wallace     |
| Brown     | Bartholomew |
| Isaacson  | Jim         |
| McTavish  | Taylor      |
| Pinter    | Marlene     |
+-----------+-------------+
```

См. также

Предложение SELECT DISTINCT подробно рассматривается в рецепте 7.4.

14.5. Устранение дубликатов из результата самообъединения

Задача

Самообъединения часто выводят строки, близкие к дубликатам, то есть строки, которые содержат одинаковые значения, но в другом порядке. В этом случае запрос SELECT DISTINCT не удалит дубликаты.

Решение

Выберите значения полей записи в определенном порядке, чтобы сделать идентичными строки с повторяющимся набором значений. Тогда можно будет использовать SELECT DISTINCT для устранения дубликатов. Или можно извлечь строки так, что близкие к дубликатам строки просто не будут выбраны.

Обсуждение

Самообъединения могут формировать строки, которые являются повторяющимися (в том смысле, что содержат одни и те же значения), хотя и не идентичными. Рассмотрим запрос, который применяет самообъединение для поиска всех пар штатов, вступивших в Союз в один и тот же год:

```
mysql> SELECT YEAR(s2.statehood) AS year, s1.name, s2.name
    -> FROM states AS s1, states AS s2
    -> WHERE YEAR(s1.statehood) = YEAR(s2.statehood)
    -> AND s1.name != s2.name
    -> ORDER BY year, s1.name, s2.name;
+------+----------------+----------------+
| year | name           | name           |
+------+----------------+----------------+
| 1787 | Delaware       | New Jersey     |
| 1787 | Delaware       | Pennsylvania   |
| 1787 | New Jersey     | Delaware       |
| 1787 | New Jersey     | Pennsylvania   |
| 1787 | Pennsylvania   | Delaware       |
| 1787 | Pennsylvania   | New Jersey     |
...
| 1912 | Arizona        | New Mexico     |
| 1912 | New Mexico     | Arizona        |
| 1959 | Alaska         | Hawaii         |
| 1959 | Hawaii         | Alaska         |
+------+----------------+----------------+
```

Условие инструкции WHERE, требующее, чтобы названия штатов в паре не были идентичными, устраняет тривиальные совпадения, показывающие, что каждый штат вступил в Союз в том же году, что и он сам. Но каждая из оставшихся пар штатов опять-таки выводится дважды. Например, одна строка содержит Delaware и New Jersey, а другая – New Jersey и Delaware. Каждая такая пара строк может считаться дубликатами, так как они содержат одинаковые значения. Однако из-за того, что значения приведены в строках

в разном порядке, строки не идентичны, и от таких повторений невозможно избавиться при помощи добавления в запрос ключевого слова DISTINCT.

Одним из путей решения проблемы может быть задание определенного постоянного порядка следования столбцов в строке. Для этого выбирайте названия с помощью двух выражений, которые располагают первыми в списке вывода меньшие значения:

```
IF(val1<val2,val1,val2) AS lesser_value,
IF(val1<val2,val2,val1) AS greater_value
```

Применение этого приема к запросу о парах штатов выводит следующий результат (выражения обеспечивают отображение названий штатов в лексическом порядке внутри каждой строки):

```
mysql> SELECT YEAR(s2.statehood) AS year,
    -> IF(s1.name<s2.name,s1.name,s2.name) AS state1,
    -> IF(s1.name<s2.name,s2.name,s1.name) AS state2
    -> FROM states AS s1, states AS s2
    -> WHERE YEAR(s1.statehood) = YEAR(s2.statehood)
    -> AND s1.name != s2.name
    -> ORDER BY year, state1, state2;
+------+---------------+----------------+
| year | state1        | state2         |
+------+---------------+----------------+
| 1787 | Delaware      | New Jersey     |
| 1787 | Delaware      | New Jersey     |
| 1787 | Delaware      | Pennsylvania   |
| 1787 | Delaware      | Pennsylvania   |
| 1787 | New Jersey    | Pennsylvania   |
| 1787 | New Jersey    | Pennsylvania   |
...
| 1912 | Arizona       | New Mexico     |
| 1912 | Arizona       | New Mexico     |
| 1959 | Alaska        | Hawaii         |
| 1959 | Alaska        | Hawaii         |
+------+---------------+----------------+
```

В выводе все еще присутствуют дубликаты, но теперь повторяющиеся строки идентичны, и ненужные копии можно удалить, добавив в запрос DISTINCT:

```
mysql> SELECT DISTINCT YEAR(s2.statehood) AS year,
    -> IF(s1.name<s2.name,s1.name,s2.name) AS state1,
    -> IF(s1.name<s2.name,s2.name,s1.name) AS state2
    -> FROM states AS s1, states AS s2
    -> WHERE YEAR(s1.statehood) = YEAR(s2.statehood)
    -> AND s1.name != s2.name
    -> ORDER BY year, state1, state2;
+------+---------------+----------------+
| year | state1        | state2         |
+------+---------------+----------------+
| 1787 | Delaware      | New Jersey     |
| 1787 | Delaware      | Pennsylvania   |
```

```
| 1787 | New Jersey     | Pennsylvania   |
...
| 1912 | Arizona        | New Mexico     |
| 1959 | Alaska         | Hawaii         |
+------+----------------+----------------+
```

Альтернативный подход к удалению неидентичных дубликатов заключается не в их выявлении и удалении, а в выборе строк таким образом, чтобы в результате запроса появлялась только одна строка из каждой пары. Это избавляет от необходимости переупорядочивания значений в строках и использования ключевого слова DISTINCT. Для запроса о парах штатов, выбор только тех строк, где название первого штата лексически меньше второго, автоматически удаляет строки, в которых имена присутствуют в обратном порядке:[1]

```
mysql> SELECT YEAR(s2.statehood) AS year, s1.name, s2.name
    -> FROM states AS s1, states AS s2
    -> WHERE YEAR(s1.statehood) = YEAR(s2.statehood)
    -> AND s1.name < s2.name
    -> ORDER BY year, s1.name, s2.name;
+------+----------------+----------------+
| year | name           | name           |
+------+----------------+----------------+
| 1787 | Delaware       | New Jersey     |
| 1787 | Delaware       | Pennsylvania   |
| 1787 | New Jersey     | Pennsylvania   |
...
| 1912 | Arizona        | New Mexico     |
| 1959 | Alaska         | Hawaii         |
+------+----------------+----------------+
```

14.6. Удаление дубликатов из таблицы

Задача

Вы хотите удалить дубликаты из таблицы, чтобы она содержала только уникальные записи.

Решение

Выберите уникальные строки из таблицы в другую таблицу и замените ею исходную. Или добавьте в таблицу уникальный индекс, используя ALTER TABLE, в результате чего дубликаты исчезнут. Или примените DELETE ... LIMIT *n* для удаления всех, кроме одного, экземпляров повторяющихся строк.

Обсуждение

Если при создании таблицы вы забыли определить уникальный индекс, предотвращающий наличие дубликатов в таблице, то в дальнейшем вы можете

[1] Это же ограничение удаляет и строки, в которых названия штатов совпадают.

столкнуться с необходимостью как-то избавиться от дубликатов. Таблица cat_mailing из примеров предыдущих разделов как раз относится к подобным таблицам, так как содержит по несколько записей об одних и тех же людях:

```
mysql> SELECT * FROM cat_mailing ORDER BY last_name, first_name;
+-----------+-------------+--------------------------+
| last_name | first_name  | street                   |
+-----------+-------------+--------------------------+
| Baxter    | Wallace     | 57 3rd Ave.              |
| BAXTER    | WALLACE     | 57 3rd Ave.              |
| Baxter    | Wallace     | 57 3rd Ave., Apt 102     |
| Brown     | Bartholomew | 432 River Run            |
| Isaacson  | Jim         | 515 Fordam St., Apt. 917 |
| McTavish  | Taylor      | 432 River Run            |
| Pinter    | Marlene     | 9 Sunset Trail           |
| Pinter    | Marlene     | 9 Sunset Trail           |
+-----------+-------------+--------------------------+
```

Таблица содержит избыточные данные, и неплохо было бы удалить их, чтобы избежать повторной рассылки и снизить почтовые издержки. Есть несколько способов сделать это:

- Выберите уникальные строки таблицы в другую таблицу, затем используйте новую таблицу для замены исходной. В результате дубликаты будут удалены. Такой способ подходит, когда под дубликатами понимаются полностью совпадающие строки.
- Добавьте в таблицу уникальный индекс при помощи ALTER TABLE. Эта операция устранит повторяющиеся строки, основываясь на содержимом индексированных столбцов.
- Дубликаты из указанного набора повторяющихся строк можно удалить с помощью предложения DELETE ... LIMIT *n*, удалив все строки кроме одной.

В этом разделе подробно рассмотрены все перечисленные способы. Размышляя о том, какой из них выбрать в определенных условиях, помните, что применимость метода к конкретной задаче часто определяется ответами на два вопроса:

- Требуется ли, чтобы таблица имела уникальный индекс?
- Если столбец, в котором встречаются дубликаты, допускает использование NULL, удалит ли метод повторяющиеся значения NULL?

Удаление дубликатов путем замены таблицы

Одним из способов удаления дубликатов из таблицы является выбор ее уникальных записей в новую таблицу, имеющую такую же структуру. Затем исходная таблица заменяется новой. Если строка считается дубликатом другой строки только в случае их полного совпадения, то можно использовать для выбора уникальных строк SELECT DISTINCT:

```
mysql> CREATE TABLE tmp SELECT DISTINCT * FROM cat_mailing;
mysql> SELECT * FROM tmp ORDER BY last_name, first_name;
```

```
+-----------+-------------+--------------------------+
| last_name | first_name  | street                   |
+-----------+-------------+--------------------------+
| Baxter    | Wallace     | 57 3rd Ave.              |
| Baxter    | Wallace     | 57 3rd Ave., Apt 102     |
| Brown     | Bartholomew | 432 River Run            |
| Isaacson  | Jim         | 515 Fordam St., Apt. 917 |
| McTavish  | Taylor      | 432 River Run            |
| Pinter    | Marlene     | 9 Sunset Trail           |
+-----------+-------------+--------------------------+
```

Этот способ работает в отсутствие индекса (хотя для больших таблиц он может быть медленным), а из таблиц, содержащих повторяющиеся значения NULL, он такие дубликаты удаляет. Обратите внимание на то, что в данном случае немного отличающиеся значениями street строки для Wallace Baxter считаются различными.

Если дубликаты определяются только относительно подмножества столбцов таблицы, создайте новую таблицу, имеющую уникальный индекс, а затем выберите в нее строки, используя INSERT IGNORE.

```
mysql> CREATE TABLE tmp (
    -> last_name CHAR(40) NOT NULL,
    -> first_name CHAR(40) NOT NULL,
    -> street CHAR(40) NOT NULL,
    -> PRIMARY KEY (last_name, first_name));
mysql> INSERT IGNORE INTO tmp SELECT * FROM cat_mailing;
mysql> SELECT * FROM tmp ORDER BY last_name, first_name;
+-----------+-------------+--------------------------+
| last_name | first_name  | street                   |
+-----------+-------------+--------------------------+
| Baxter    | Wallace     | 57 3rd Ave.              |
| Brown     | Bartholomew | 432 River Run            |
| Isaacson  | Jim         | 515 Fordam St., Apt. 917 |
| McTavish  | Taylor      | 432 River Run            |
| Pinter    | Marlene     | 9 Sunset Trail           |
+-----------+-------------+--------------------------+
```

Индекс не допускает вставки записей с дублирующимися значениями ключа в tmp, а IGNORE указывает MySQL на то, что не следует завершаться с ошибкой в случае обнаружения дубликата. Недостатком метода является то, что если индексированные столбцы могут содержать значения NULL, то необходимо использовать индекс UNIQUE вместо PRIMARY KEY, и тогда не будут удаляться повторяющиеся ключи со значениями NULL (индекс UNIQUE допускает несколько значений NULL).

После создания новой таблицы tmp, содержащей только уникальные строки, используйте ее для замены исходной таблицы cat_mailing. В результате cat_mailing больше не будет содержать дубликатов:

```
mysql> DROP TABLE cat_mailing;
mysql> ALTER TABLE tmp RENAME TO cat_mailing;
```

Удаление дубликатов путем добавления индекса

Чтобы удалить дубликаты из таблицы непосредственно «на месте», добавьте в таблицу уникальный индекс при помощи ALTER TABLE, используя ключевое слово IGNORE для того, чтобы указать на необходимость удаления записей с дублирующимися значениями ключа в процессе построения индекса. Исходная таблица cat_mailing без индекса выглядит так:

```
mysql> SELECT * FROM cat_mailing ORDER BY last_name, first_name;
+-----------+-------------+--------------------------+
| last_name | first_name  | street                   |
+-----------+-------------+--------------------------+
| Baxter    | Wallace     | 57 3rd Ave.              |
| BAXTER    | WALLACE     | 57 3rd Ave.              |
| Baxter    | Wallace     | 57 3rd Ave., Apt 102     |
| Brown     | Bartholomew | 432 River Run            |
| Isaacson  | Jim         | 515 Fordam St., Apt. 917 |
| McTavish  | Taylor      | 432 River Run            |
| Pinter    | Marlene     | 9 Sunset Trail           |
| Pinter    | Marlene     | 9 Sunset Trail           |
+-----------+-------------+--------------------------+
```

Добавьте уникальный индекс и посмотрите, как это подействует на содержимое таблицы:

```
mysql> ALTER IGNORE TABLE cat_mailing
    -> ADD PRIMARY KEY (last_name, first_name);
mysql> SELECT * FROM cat_mailing ORDER BY last_name, first_name;
+-----------+-------------+--------------------------+
| last_name | first_name  | street                   |
+-----------+-------------+--------------------------+
| Baxter    | Wallace     | 57 3rd Ave.              |
| Brown     | Bartholomew | 432 River Run            |
| Isaacson  | Jim         | 515 Fordam St., Apt. 917 |
| McTavish  | Taylor      | 432 River Run            |
| Pinter    | Marlene     | 9 Sunset Trail           |
+-----------+-------------+--------------------------+
```

Если индексированные столбцы могут содержать значения NULL, то вместо индекса PRIMARY KEY нужно использовать UNIQUE. Тогда индекс не будет удалять повторяющиеся значения NULL.

Удаление дубликатов определенной строки

Начиная с версии MySQL 3.22.7, вы можете использовать инструкцию LIMIT для ограничения действия предложения DELETE на подмножество строк, которое оно в противном случае удалило бы. В такой форме предложение можно применять для удаления повторяющихся записей. Предположим, что у вас есть таблица t с таким содержимым:

```
+-------+
| color |
+-------+
```

```
| blue  |
| green |
| blue  |
| blue  |
| red   |
| green |
| red   |
+-------+
```

В таблице трижды присутствует `blue` (голубой) и дважды — `green` (зеленый) и `red` (красный). Чтобы удалить дополнительные экземпляры каждого цвета, выполните:

```
mysql> DELETE FROM t WHERE color = 'blue' LIMIT 2;
mysql> DELETE FROM t WHERE color = 'green' LIMIT 1;
mysql> DELETE FROM t WHERE color = 'red' LIMIT 1;
mysql> SELECT * FROM t;
+-------+
| color |
+-------+
| blue  |
| green |
| red   |
+-------+
```

Прием работает в отсутствие уникального индекса и удаляет повторяющиеся значения `NULL`. Его удобно использовать, если вам нужно удалить дубликаты только для определенного множества строк таблицы. Однако если следует удалить много различных наборов дубликатов, подобную процедуру вряд ли захочется проводить вручную. Процесс можно автоматизировать, используя методы, предложенные ранее в рецепте 14.3 для выявления дубликатов. В том рецепте мы создали функцию `make_dup_count_query()` для генерирования запроса, подсчитывающего количество повторяющихся значений в указанном множестве столбцов таблицы:

```
sub make_dup_count_query
{
my ($tbl_name, @col_name) = @_;

    return (
        "SELECT COUNT(*)," . join (",", @col_name)
        . "\nFROM $tbl_name"
        . "\nGROUP BY " . join (",", @col_name)
        . "\nHAVING COUNT(*) > 1"
    );
}
```

Можно написать еще одну функцию `delete_dups()`, использующую `make_dup_count_query()` для определения того, какие значения таблицы повторяются и как часто. Из этой информации можно понять, сколько дубликатов следует удалить при помощи `DELETE … LIMIT` *n*, чтобы в таблице остался только один экземпляр записи. Функция `delete_dups()` выглядит так:

```
sub delete_dups
{
my ($dbh, $tbl_name, @col_name) = @_;

    # Создать и выполнить запрос, который находит дубликаты

    my $dup_info = $dbh->selectall_arrayref (
                    make_dup_count_query ($tbl_name, @col_name)
                );
    return unless defined ($dup_info);

    # Для каждого повторяющегося множества значений удалить все вхождения строк,
    # содержащих эти значения, кроме одного

    foreach my $row_ref (@{$dup_info})
    {
        my ($count, @col_val) = @{$row_ref};
        next unless $count > 1;
        # Построить строку условия для сравнения значений, не забывая про IS NULL
        my $str;
        for (my $i = 0; $i < @col_name; $i++)
        {
            $str .= " AND " if $str;
            $str .= defined ($col_val[$i])
                        ? "$col_name[$i] = " . $dbh->quote ($col_val[$i])
                        : "$col_name[$i] IS NULL";
        }
        $str = "DELETE FROM $tbl_name WHERE $str LIMIT " . ($count - 1);
        $dbh->do ($str);
    }
}
```

Предположим, что у нас есть таблица `employee` **со следующими записями:**

```
mysql> SELECT * FROM employee;
+----------+------------+
| name     | department |
+----------+------------+
| Fred     | accounting |
| Fred     | accounting |
| Fred     | accounting |
| Fred     | accounting |
| Bob      | shipping   |
| Mary Ann | shipping   |
| Mary Ann | shipping   |
| Mary Ann | sales      |
| Mary Ann | sales      |
| Mary Ann | sales      |
| Mary Ann | sales      |
| Mary Ann | sales      |
| Mary Ann | sales      |
| Boris    | NULL       |
| Boris    | NULL       |
+----------+------------+
```

Для того чтобы использовать функцию `delete_dups()` для удаления дубликатов в столбцах `name` и `department` таблицы `employee`, вызовите ее так:

```
delete_dups ($dbh, "employee", "name", "department");
```

Функция `delete_dups()` вызывает функцию `make_dup_count_query()` и выполняет формируемый ею запрос `SELECT`. Для таблицы `employee` этот запрос выводит такой результат:

```
+----------+----------+------------+
| COUNT(*) | name     | department |
+----------+----------+------------+
|        2 | Boris    | NULL       |
|        4 | Fred     | accounting |
|        6 | Mary Ann | sales      |
|        2 | Mary Ann | shipping   |
+----------+----------+------------+
```

Функция `delete_dups()` использует данную информацию для формирования следующих предложений `DELETE`:

```
DELETE FROM employee
WHERE name = 'Boris' AND department IS NULL LIMIT 1
DELETE FROM employee
WHERE name = 'Fred' AND department = 'accounting' LIMIT 3
DELETE FROM employee
WHERE name = 'Mary Ann' AND department = 'sales' LIMIT 5
DELETE FROM employee
WHERE name = 'Mary Ann' AND department = 'shipping' LIMIT 1
```

В целом, способ с использованием `DELETE ... LIMIT n`, вероятно, работает медленнее, чем удаление дубликатов при помощи создания второй таблицы или добавления уникального индекса. Эти методы хранят данные на сервере и позволяют ему выполнить всю работу. `DELETE ... LIMIT n` требует большого объема взаимодействия клиента с сервером, так как использует запрос `SELECT` для извлечения информации о дубликатах, а затем ряд предложений `DELETE` для удаления копий дублирующихся строк.

Когда вы запускаете предложения `DELETE ... LIMIT n` из программы, убедитесь в том, что они выполняются только для положительных значений *n*. Это не только разумно (зачем нужны дополнительные издержки на запросы, которые ничего не удаляют?), но и необходимо, чтобы избежать ошибки, имеющейся в некоторых версиях MySQL. Естественно предполагать, что предложение вида `DELETE ... LIMIT 0` не должно удалять ни одной записи, и в текущих версиях MySQL дело обстоит именно так. Но в версиях до 3.23.40 имелась ошибка: `LIMIT 0` интерпретировалось как отсутствие инструкции `LIMIT`, в результате предложение `DELETE` удаляло *все* выбранные строки! (В тех же версиях MySQL эта проблема возникает и для `UPDATE ... LIMIT 0`.)

15

Выполнение транзакций

15.0. Введение

Благодаря своей многопоточности, сервер MySQL способен обслуживать множество клиентов одновременно. Исключение конкуренции между пользователями достигается блокировкой соответствующих ресурсов, не позволяющей двум клиентам одновременно модифицировать одни и те же данные. Тем не менее, в процессе выполнения запросов нельзя исключить такой ситуации, когда в последовательность запросов некоторого пользователя «вклинятся» запросы и от других клиентов. Если несколько запросов зависят один от другого, тот факт, что другие клиенты могут изменить данные в промежутке между ними, может создать проблемы. Ошибки выполнения запросов также создают сложности, если операция, состоящая из нескольких запросов, не завершилась полностью. Предположим, у вас есть таблица `flight` с информацией о расписании полетов, и вы хотите изменить запись о рейсе 578, выбрав нового пилота из списка свободных. Это можно сделать такими тремя предложениями:

```
SELECT @p_val := pilot_id FROM pilot WHERE available = 'yes' LIMIT 1;
UPDATE pilot SET available = 'no' WHERE pilot_id = @p_val;
UPDATE flight SET pilot_id = @p_val WHERE flight_id = 578;
```

Первое предложение выбирает одного из свободных пилотов, второе отмечает его как занятого, а третье назначает пилота на указанный рейс. На практике это довольно просто, но с точки зрения теории здесь есть пара подводных камней:

Проблема конкуренции. Сервер MySQL может обслуживать нескольких клиентов одновременно. Если два пользователя собираются изменять расписание, возможна ситуация, когда оба они выполнят первое предложение `SELECT` и получат один и тот же идентификатор пилота, прежде чем кто-либо из них успеет пометить его как недоступный. Если это произойдет, пилот будет назначен сразу на два рейса.

Проблема целостности. Все три предложения должны выполниться успешно как единое целое. Например, если запросы `SELECT` и первый `UPDATE`

завершились успешно, а второй UPDATE — нет, то пилот получит статус «занят», но не будет назначен на рейс. База данных останется в противоречивом состоянии.

Возникающие в таких ситуациях проблемы конкуренции и целостности решаются с помощью транзакций. Транзакция объединяет набор предложений в группу и гарантирует выполнение следующих условий:

- Никакой другой пользователь не может изменить данные, задействованные в транзакции, до ее завершения, сервер как бы предоставляется вам в единоличное пользование. Например, пока вы назначаете пилота на рейс, другие пользователи не смогут изменить записи о пилоте и рейсе. Таким образом, запрещая другим клиентам влиять на выполняемые вами операции, транзакции решают проблему конкуренции, связанную с многопользовательским характером работы сервера MySQL. По сути, транзакции обеспечивают поочередный доступ к разделяемым ресурсам для операций, состоящих из нескольких предложений.
- Сгруппированные в транзакции предложения фиксируются (вступают в силу) как единое целое, но только в том случае, если все они выполнились успешно. Если возникает ошибка, все выполненные операции откатываются, оставляя соответствующие таблицы неизмененными, как будто ни одно из предложений и не выполнялось. Это не дает информации базы данных стать противоречивой. Например, если не удается обновить таблицу flights, то откат отменяет изменение таблицы pilots, и пилот остается свободным. Откат освобождает вас от необходимости самостоятельно отменять частично выполненную операцию.

В этой главе рассказывается том, как определить, поддерживает ли ваш сервер MySQL транзакции. Приводится синтаксис предложений SQL, начинающих и завершающих транзакцию. Описывается реализация транзакций в программах с использованием механизма выявления ошибок для определения того, следует ли зафиксировать или же откатить операции. В последнем разделе обсуждаются возможные обходные пути для тех, чей сервер MySQL не поддерживает транзакции.

Сценарии примеров главы хранятся в каталоге *transactions* дистрибутива recipes.

15.1. Проверка поддержки транзакций

Задача

Вы хотите использовать транзакции, но не знаете, поддерживает ли их ваш сервер MySQL.

Решение

Проверьте версию сервера, чтобы убедиться в том, что она достаточно свежая, и определите, какие типы таблиц поддерживаются. Или можно попы-

таться создать таблицу транзакционного типа и посмотреть, действительно ли MySQL использует такой тип для определения таблицы.

Обсуждение

Для того чтобы использовать транзакции в MySQL, необходима достаточно свежая версия сервера, которая поддерживала бы соответствующие табличные дескрипторы, а приложения должны использовать таблицы транзакционного типа. Используйте следующий запрос для проверки версии вашего сервера:

```
mysql> SELECT VERSION();
+----------------+
| VERSION()      |
+----------------+
| 4.0.4-beta-log |
+----------------+
```

Поддержка транзакций была введена в версии MySQL 3.23.17 с включением таблиц BDB (Berkeley DB) транзакционного типа. С того времени появился еще тип InnoDB, и, начиная с MySQL 3.23.29, можно работать с обоими типами таблиц. Я бы рекомендовал использовать последнюю из доступных версий MySQL. Поддержка транзакций (как и сам сервер) была значительно усовершенствована с момента выхода версии 3.23.29.

Даже если сервер достаточно современен для того, чтобы обеспечивать поддержку транзакций, может оказаться, что на самом деле он не обладает такими возможностями. Обработчики соответствующих типов таблиц могли быть не сконфигурированы при компиляции сервера. Может оказаться, что обработчики присутствуют, но отключены – если сервер был запущен с опцией *--skip-bdb* или *--skip-innodb*. Чтобы проверить доступность и статус обработчиков транзакционных таблиц, используйте предложение `SHOW VARIABLES`:

```
mysql> SHOW VARIABLES LIKE 'have_bdb';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| have_bdb      | YES   |
+---------------+-------+
mysql> SHOW VARIABLES LIKE 'have_innodb';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| have_innodb   | YES   |
+---------------+-------+
```

Приведенный результат запроса показывает, что таблицы BDB и InnoDB могут быть использованы. Если бы какой-то из запросов не возвращал значения или выводил в столбце `Value` что-то отличное от `YES` (`NO` или `DISABLED`), то соответствующий тип таблиц был бы недоступен.

Программные методы проверки версии сервера и поддерживаемых им типов таблиц рассматриваются в рецептах 9.13 и 9.17.

Еще один способ проверки доступности определенного типа таблиц – попытаться создать таблицу такого типа. Создайте предложение SHOW CREATE TABLE, чтобы посмотреть, какой тип MySQL использует фактически. Например, попробуем создать t как таблицу InnoDB, выполнив следующие предложения:

```
mysql> CREATE TABLE t (i INT) TYPE = InnoDB;
mysql> SHOW CREATE TABLE t\G
*************************** 1. row ***************************
       Table: t
Create Table: CREATE TABLE `t` (
  `i` int(11) default NULL
) TYPE=InnoDB
```

Если тип InnoDB доступен, то в последней части выражения SHOW будет содержаться TYPE=InnoDB. В противном случае MySQL создаст таблицу, используя тип по умолчанию – MyISAM, тогда последняя часть предложения будет выглядеть как TYPE=MyISAM. (Для проверки типа таблицы также можно применить SHOW TABLE STATUS.)

Если ваш сервер MySQL не содержит обработчиков транзакционных таблиц, необходимо заменить его. При установке MySQL из дистрибутива в исходных текстах инструкции инсталлятора подскажут вам, какие конфигурационные флаги следует использовать для разрешения соответствующих обработчиков. Если вы предпочитаете двоичный дистрибутив, проверьте, что устанавливаете тот из них, в который включены обработчики BDB или InnoDB.

После того как вы убедитесь в том, что сервер поддерживает соответствующие типы транзакционных таблиц, можно начинать использовать их в приложениях:

- Если вы пишете новое приложение, то можете сразу создать таблицы как таблицы транзакционного типа. Для этого нужно только добавить TYPE = *тип_таблицы* в конец предложения CREATE TABLE:

  ```
  CREATE TABLE t1 (i INT) TYPE = BDB;
  CREATE TABLE t2 (i INT) TYPE = INNODB;
  ```

- Если вы изменяете существующее приложение так, что появляется необходимость в применении транзакций для уже имеющихся таблиц, которые изначально не предназначались для транзакций, можно изменить тип таких таблиц. Например, типы ISAM и MyISAM относятся к нетранзакционным. Попытка использовать их для транзакций приведет к выводу некорректных результатов, так как таблицы не поддерживают операцию отката. Можно применить предложение ALTER TABLE для преобразования таблицы к транзакционному типу. Предположим, что t – это таблица MyISAM. Чтобы преобразовать ее к типу InnoDB, выполните:

  ```
  ALTER TABLE t TYPE = INNODB;
  ```

  Имейте в виду, что изменение типа таблицы для поддержки транзакций может изменить и другие аспекты ее поведения. Например, таблицы MyISAM обеспечивают более гибкую обработку столбцов AUTO_INCREMENT, чем другие типы. Если вы используете возможности последовательнос-

тей, доступные только для MyISAM, то изменение типа таблицы может создать проблемы (это обсуждалось в главе 11).

Если сервер не поддерживает транзакции и у вас нет возможности заменить его другим, можно достичь аналогичных результатов без использования транзакций. В некоторых случаях можно заблокировать таблицу на период выполнения нескольких предложений при помощи предложений `LOCK` и `UNLOCK`. Тогда другие клиентские приложения не смогут вмешаться в вашу работу, однако откатить какое-то неудавшееся предложение будет невозможно. Есть и другой путь – изменить запросы так, чтобы они не требовали использования транзакций. Оба способа подробно рассмотрены в рецепте 15.8.

15.2. Выполнение транзакций средствами SQL

Задача

Вам требуется создать ряд запросов, которые должны успешно выполниться или же не выполниться все вместе как единое целое.

Решение

Настройте режим автофиксации MySQL так, чтобы были разрешены транзакции, включающие несколько предложений, затем фиксируйте или откатывайте предложения в зависимости от успешности их выполнения.

Обсуждение

Этот раздел описывает предложения SQL, управляющие поведением транзакций в MySQL. В последующих разделах рассказано о программной реализации транзакций. Некоторые API требуют запуска соответствующих предложений SQL, другие предлагают собственный механизм, обеспечивающий работу с транзакциями без непосредственного участия SQL. Однако даже в последнем случае механизм API сопоставляет программным операциям транзакционные предложения SQL, так что прочтите этот раздел, чтобы лучше представлять себе то, что API делает для вас.

Обычно MySQL работает в режиме автофиксации, то есть результат каждого предложения фиксируется сразу после его выполнения. (Фактически, каждое предложение – это транзакция.) Чтобы выполнить транзакцию, состоящую из нескольких предложений, необходимо отключить режим автофиксации, выдать предложения, составляющие транзакцию, затем или принять или откатить сделанные изменения. В MySQL есть два способа сделать это:

- Создайте предложение `BEGIN` (или `BEGIN WORK`) для временного отключения режима автофиксации, затем запустите запросы, входящие в транзакцию, запишите результат в базу данных и завершите транзакцию, выполнив предложение `COMMIT`:

```
mysql> CREATE TABLE t (i INT) TYPE = InnoDB;
mysql> BEGIN;
mysql> INSERT INTO t (i) VALUES(1);
```

```
mysql> INSERT INTO t (i) VALUES(2);
mysql> COMMIT;
mysql> SELECT * FROM t;
+------+
| i    |
+------+
|    1 |
|    2 |
+------+
```

В случае ошибки не используйте COMMIT. Отменить транзакцию можно, выполнив предложение ROLLBACK. В следующем примере таблица t остается пустой после выполнения транзакции, так как предложения INSERT откатываются:

```
mysql> CREATE TABLE t (i INT) TYPE = InnoDB;
mysql> BEGIN;
mysql> INSERT INTO t (i) VALUES(1);
mysql> INSERT INTO t (x) VALUES(2);
ERROR 1054 at line 5: Unknown column 'x' in 'field list'
mysql> ROLLBACK;
mysql> SELECT * FROM t;
Empty set (0.00 sec)
```

- Другой способ группировки предложений – явное отключение режима автофиксации. Тогда каждое выдаваемое предложение становится частью текущей транзакции. Чтобы завершить транзакцию и начать следующую, выполните предложение COMMIT или ROLLBACK:

```
mysql> CREATE TABLE t (i INT) TYPE = InnoDB;
mysql> SET AUTOCOMMIT = 0;
mysql> INSERT INTO t (i) VALUES(1);
mysql> INSERT INTO t (i) VALUES(2);
mysql> COMMIT;
mysql> SELECT * FROM t;
+------+
| i    |
+------+
|    1 |
|    2 |
+------+
```

Для возврата в режим автофиксации выполните такую команду:

```
mysql> SET AUTOCOMMIT = 1;
```

Отменить можно не все

Транзакции действуют в определенных пределах, так как не все предложения могут включаться в них. Например, если вы выполняете предложение DROP DATABASE, то не надейтесь получить базу данных обратно, выполнив ROLLBACK.

15.3. Выполнение транзакций в программах

Задача

Вы пишете программу, в которой необходимо использовать транзакции.

Решение

Используйте абстракцию транзакции в API для вашего языка программирования, если такая существует. Если нет, используйте обычный механизм выполнения запроса в API для непосредственного запуска транзакционных предложений SQL.

Обсуждение

При интерактивном запуске запросов из *mysql* (как в примерах предыдущего раздела) вы можете увидеть, успешно ли выполнено предложение, и на основе этой информации определить, следует зафиксировать или откатить изменения. Если же вы выполняете запросы в неинтерактивном сценарии SQL, хранящемся в файле, то все не так просто. У вас нет возможности выполнять фиксацию и откат в зависимости от результата выполнения предложения, так как в MySQL нет инструкции `IF/THEN/ELSE`, которая управляла бы логикой сценария. (Есть функция `IF()`, но это не то же самое.) Поэтому обработка транзакций обычно производится в программе, где можно применить средства базового языка для выявления ошибок и выполнения соответствующего действия. В этом разделе представлены основные сведения о том, как это делать. Следующие разделы посвящены особенностям реализации транзакций в программных интерфейсах Perl, PHP, Python и Java.

Все API поддерживают транзакции хотя бы в том смысле, что вы можете явно создавать транзакционные предложения SQL, такие как `BEGIN` и `COMMIT`. Однако некоторые API предлагают абстракцию транзакции, которая позволяет управлять поведением транзакций без непосредственного обращения к SQL. Этот подход скрывает детали и обеспечивает переносимость на другие базы данных, поддерживающие транзакции, синтаксис SQL которых может несколько отличаться от исходного. MySQL-интерфейсы Perl, Python и Java предоставляют такую абстракцию. В PHP вам придется создавать предложения SQL самостоятельно.

В следующих разделах для иллюстрации программной реализации транзакций рассматривается один и тот же пример. В них используется таблица `t`, исходное содержимое которой показывает, сколько денег было у каждого из двух человек:

```
+------+------+
| name | amt  |
+------+------+
| Eve  |   10 |
| Ida  |    0 |
+------+------+
```

Примером транзакции будет простая финансовая операция перевода денег, использующая два предложения UPDATE, чтобы передать Иде 6 долларов Евы:

```
UPDATE money SET amt = amt - 6 WHERE name = 'Eve';
UPDATE money SET amt = amt + 6 WHERE name = 'Ida';
```

В результате таблица станет такой:

```
+------+------+
| name | amt  |
+------+------+
| Eve  |    4 |
| Ida  |    6 |
+------+------+
```

Необходимо включить оба предложения в транзакцию, чтобы быть уверенным в том, что они вступают в силу одновременно. В отсутствие транзакции деньги Евы просто пропадут, если не выполнится второе предложение. Если же использовать транзакцию, то в случае невыполнения предложения таблица останется неизмененной.

Программы примеров для каждого языка хранятся в каталоге *transactions* дистрибутива recipes. Если сравнить их, то станет видно, что все они используют одну и ту же схему обработки транзакций:

- Предложения транзакции группируются в управляющую структуру вместе с операцией фиксации.
- Если статус управляющей структуры указывает на то, что она не была успешно завершена, транзакция откатывается.

Эту логику можно выразить следующим образом (block – это управляющая структура, используемая для группировки предложений):

```
block:
    statement 1
    statement 2
    ...
    statement n
    commit
if the block failed:
    roll back
```

В Perl управляющая структура – это блок eval, который или выполняется успешно, или завершается с ошибкой, возвращая ее код. Python и Java используют блок try, который выполняется до конца, если транзакция была успешной. В случае ошибки генерируется исключение, которое инициирует выполнение соответствующего блока обработки ошибок для отката транзакции. В PHP нет подобных конструкций, но вы можете получить тот же результат, выполняя предложения транзакции и ее фиксацию в функции. Если функция завершается с ошибкой, выполняется откат.

Преимущество такого структурирования кода в том, что минимизируется количество проверок, необходимых для определения необходимости отката. Альтернативный вариант – проверка результата каждого предложения в

транзакции и откат при ошибках отдельных предложений, быстро сделает ваш код беспорядочным и трудночитаемым.

При выполнении отката в языке, генерирующем исключения, необходимо помнить о том, что сам откат может не удаться, что вызовет новое исключение. Если вы не хотите иметь с этим дело, выполняйте откат в отдельном блоке с пустым обработчиком исключения. Именно так поступают программы на Perl, Python и Java.

Сопоставление абстракции транзакции API предложениям SQL

Если в API имеется абстракция транзакции, то можно увидеть, как программный интерфейс отображается на стандартный SQL, включив протоколирование на сервере MySQL и исследуя лог запросов, чтобы посмотреть, какие предложения API выполняет, когда вы запускаете программу с транзакциями.

15.4. Использование транзакций в программах на Perl

Задача

Вы хотите выполнить транзакцию в DBI-сценарии.

Решение

Используйте стандартный механизм поддержки транзакций DBI.

Обсуждение

Механизм реализации транзакции в DBI базируется на явном управлении режимом автофиксации. Процедура выглядит так:

1. Если это еще не сделано, включите атрибут `RaiseError` и выключите `PrintError`. Вы хотите, чтобы ошибки порождали исключения и ничего не выводили; если же оставить `PrintError` включенным, в некоторых случаях это может помешать обнаружению ошибки.
2. Отключите атрибут `AutoCommit`, чтобы фиксация происходила только по вашему указанию.
3. Выполните предложения транзакции в блоке `eval` так, чтобы ошибки вызывали исключение и завершали блок. Последним в блоке должен быть вызов `commit()`, который фиксировал бы транзакцию в случае успешного выполнения всех ее предложений.
4. После выполнения `eval` проверьте переменную `$@`. Если она содержит пустую строку, транзакция выполнена успешно. В противном случае это

означает, что произошла какая-то ошибка, и `$@` будет содержать сообщение об ошибке. Вызовите `rollback()` для отмены транзакции. Если вы хотите отобразить для пользователя сообщение об ошибке, выведите `$@` перед вызовом `rollback()`.

Следующий код показывает, как реализовать процедуру выполнения нашей транзакции. Текущие значения атрибутов обработки ошибок и автофиксации сохраняются перед выполнением транзакции, а после ее выполнения возвращаются в исходное состояние. Для ваших приложений это может оказаться избыточным. Например, если вы знаете, что `RaiseError` и `PrintError` уже установлены так, как надо, нет необходимости в их сохранении и восстановлении.

```
# сохраняем атрибуты обработки ошибок и автофиксации,
# затем убеждаемся в том, что они установлены корректно.
$save_re = $dbh->{RaiseError};
$save_pe = $dbh->{PrintError};
$save_ac = $dbh->{AutoCommit};
$dbh->{RaiseError} = 1; # исключение в случае ошибки
$dbh->{PrintError} = 0; # не выводить сообщение об ошибке
$dbh->{AutoCommit} = 0; # отключить автофиксацию

eval
{
    # передать немного денег от одного человека другому
    $dbh->do ("UPDATE money SET amt = amt - 6 WHERE name = 'Eve'");
    $dbh->do ("UPDATE money SET amt = amt + 6 WHERE name = 'Ida'");
    # все предложения выполнены успешно, зафиксировать транзакцию
    $dbh->commit ();
};

if ($@) # произошла ошибка
{
    print "Transaction failed, rolling back. Error was:\n$@\n";
    # откат внутри eval, чтобы ошибка отката не привела к завершению работы сценария
    eval { $dbh->rollback (); };
}

# восстановить исходное состояние атрибутов
$dbh->{AutoCommit} = $save_ac;
$dbh->{PrintError} = $save_pe;
$dbh->{RaiseError} = $save_re;
```

Как видите, приходится проделать большую работу для того, чтобы выдать пару предложений. Чтобы упростить обработку транзакции, можно создать несколько функций, которые займутся обработкой до и после выполнения `eval`:

```
sub transact_init
{
my $dbh = shift;
my $attr_ref = {};  # создать хеш для хранения атрибутов
```

```
    $attr_ref->{RaiseError} = $dbh->{RaiseError};
    $attr_ref->{PrintError} = $dbh->{PrintError};
    $attr_ref->{AutoCommit} = $dbh->{AutoCommit};
    $dbh->{RaiseError} = 1; # исключение в случае ошибки
    $dbh->{PrintError} = 0; # не выводить сообщение об ошибке
    $dbh->{AutoCommit} = 0; # отключить автофиксацию
    return ($attr_ref);     # вернуть атрибуты в вызывающую программу
}

sub transact_finish
{
my ($dbh, $attr_ref, $error) = @_;

    if ($error) # произошла ошибка
    {
        print "Transaction failed, rolling back. Error was:\n$error\n";
        # откат внутри eval, чтобы ошибка отката не привела
        # к завершению работы сценария
        eval { $dbh->rollback (); };
    }
    # восстановить исходное состояние атрибутов обработки ошибок и автофиксации
    $dbh->{AutoCommit} = $attr_ref->{AutoCommit};
    $dbh->{PrintError} = $attr_ref->{PrintError};
    $dbh->{RaiseError} = $attr_ref->{RaiseError};
}
```

Если использовать две эти функции, наша транзакция значительно упростится:

```
$ref = transact_init ($dbh);
eval
{
    # передать деньги от одного человека другому
    $dbh->do ("UPDATE money SET amt = amt - 6 WHERE name = 'Eve'");
    $dbh->do ("UPDATE money SET amt = amt + 6 WHERE name = 'Ida'");
    # все предложения выполнены успешно, зафиксировать транзакцию
    $dbh->commit ();
};
transact_finish ($dbh, $ref, $@);
```

Начиная с DBI 1.20 есть альтернатива ручному управлению атрибутом `AutoCommit` **– можно начинать транзакцию, вызывая** `begin_work()`**. Этот метод отключает** `AutoCommit` **и автоматически разрешает его при последующем вызове** `commit()` **или** `rollback()`**.**

Транзакции и старые версии DBD::mysql

Механизм поддержки транзакций DBI требует версии *DBD::mysql* 1.2216 или выше. В более ранних версиях установка атрибута `AutoCommit` **ни к чему не приведет, так что вам придется самостоятельно запускать соответствующие предложения SQL (**`BEGIN`**,** `COMMIT`**,** `ROLLBACK`**).**

15.5. Использование транзакций в программах на PHP

Задача

Вы хотите выполнить транзакцию в PHP-сценарии.

Решение

Запускайте предложения SQL, которые начинают и завершают транзакции.

Обсуждение

В PHP нет специального механизма транзакций, так что следует непосредственно создавать соответствующие предложения SQL. То есть вы можете или использовать `BEGIN` для начала транзакции, или самостоятельно включать и выключать режим автофиксации, используя `SET AUTOCOMMIT`. В следующем примере используется `BEGIN`. Предложения транзакции помещены в функцию, чтобы избежать ненужной проверки ошибок. Чтобы определить, нужно ли произвести откат, нужно просто проверить результат функции:

```
function do_queries ($conn_id)
{
    # передать деньги от одного человека другому
    if (!mysql_query ("BEGIN", $conn_id))
        return (0);
    if (!mysql_query ("UPDATE money SET amt = amt - 6 WHERE name = 'Eve'", $conn_id))
        return (0);
    if (!mysql_query ("UPDATE money SET amt = amt + 6 WHERE name = 'Ida'", $conn_id))
        return (0);
    if (!mysql_query ("COMMIT", $conn_id))
        return (0);
    return (1);
}

if (!do_queries ($conn_id))
{
    print ("Transaction failed, rolling back. Error was:\n"
           . mysql_error ($conn_id) . "\n");
    mysql_query ("ROLLBACK", $conn_id);
}
```

Функция `do_queries()` проверяет каждый метод и возвращает «ложь», если хотя бы один из них не выполнен. Такой вид проверки применяется в тех случаях, когда может потребоваться провести дополнительную обработку между предложениями или после выполнения предложений, но до возвращения успешного результата. В приведенном примере никакой дополнительной обработки не требуется, и можно переформулировать `do_queries()` как одно длинное предложение:

```
function do_queries ($conn_id)
```

```
{
    # передать деньги от одного человека другому
    return
    (
        mysql_query ("BEGIN", $conn_id)
        &&
        mysql_query ("UPDATE money SET amt = amt - 6 WHERE name = 'Eve'", $conn_id)
        &&
        mysql_query ("UPDATE money SET amt = amt + 6 WHERE name = 'Ida'", $conn_id)
        &&
        mysql_query ("COMMIT", $conn_id)
    );
}
```

15.6. Использование транзакций в программах на Python

Задача

Вы хотите использовать транзакцию в сценарии DB-API.

Решение

Используйте стандартный механизм поддержки транзакций DB-API.

Обсуждение

В DB-API имеется абстракция, обеспечивающая контроль обработки транзакций за счет методов объектов соединений с базой данных. Вызовите `begin()` для начала транзакции и `commit()` или `rollback()` для ее завершения. Вызовы `begin()` и `commit()` помещены в блок `try`, а `rollback()` — в соответствующий блок `except` для отмены транзакции в случае возникновения ошибки:

```
try:
    conn.begin ()
    cursor = conn.cursor ()
    # передать деньги от одного человека другому
    cursor.execute ("UPDATE money SET amt = amt - 6 WHERE name = 'Eve'")
    cursor.execute ("UPDATE money SET amt = amt + 6 WHERE name = 'Ida'")
    cursor.close ()
    conn.commit()
except MySQLdb.Error, e:
    print "Transaction failed, rolling back. Error was:"
    print e.args
    try:    # пустой обработчик исключения, если откат не удался
        conn.rollback ()
    except:
        pass
```

15.7. Использование транзакций в программах на Java

Задача

Вы хотите выполнить транзакцию в сценарии JDBC.

Решение

Используйте стандартный механизм поддержки транзакций JDBC.

Обсуждение

Для выполнения транзакции в Java используйте ваш объект `Connection` для отключения режима автофиксации. После запуска запросов используйте метод `commit()`объекта для фиксации транзакции или `rollback()` для ее отката. Обычно предложения транзакции выполняются в блоке `try`, в конце которого стоит `commit()`. Для обработки ошибок вызовите `rollback()` в соответствующем обработчике исключений:

```
try
{
    conn.setAutoCommit (false);
    Statement s = conn.createStatement ();
    // передать деньги от одного человека другому
    s.executeUpdate ("UPDATE money SET amt = amt - 6 WHERE name = 'Eve'");
    s.executeUpdate ("UPDATE money SET amt = amt + 6 WHERE name = 'Ida'");
    s.close ();
    conn.commit ();
    conn.setAutoCommit (true);
}
catch (SQLException e)
{
    System.err.println ("Transaction failed, rolling back.");
    Cookbook.printErrorMessage (e);
    // пустой обработчик исключений, если откат не удался
    try
    {
        conn.rollback ();
        conn.setAutoCommit (true);
    }
    catch (Exception e2) { }
}
```

15.8. Альтернативы транзакциям

Задача

Вам нужно выполнить транзакцию, но ваш сервер MySQL не поддерживает их.

Решение

Некоторые транзакционные операции можно заменить явной блокировкой таблицы. А в некоторых случаях транзакция вообще не нужна, можно переформулировать запрос так, что необходимость в транзакции полностью отпадет.

Обсуждение

Транзакции очень полезны, но в некоторых ситуациях их невозможно или не нужно использовать:

- Ваш сервер может вообще не поддерживать транзакции. (Он может быть слишком старым или у него могут быть не установлены соответствующие типы таблиц, см. рецепт 15.1). В этом случае остается только придумать какую-то замену транзакциям. Для предотвращения проблем с параллельными операциями можно использовать явную блокировку таблиц.
- Иногда приложения используют транзакции там, где в этом нет необходимости. Можно переформулировать предложения и избавиться от транзакций. В результате может даже улучшиться производительность приложения.

Группировка предложений с помощью блокировок

Если ваш сервер не поддерживает транзакции, а вам требуется выполнить группу предложений, не пересекаясь с другими клиентскими приложениями, используйте `LOCK TABLE` и `UNLOCK TABLE`:[1]

- Установите для всех используемых таблиц блокировку при помощи `LOCK TABLE`. (Применяйте блокировку записи для таблиц, которые вы будете изменять, и блокировку чтения для остальных таблиц.) Тогда другие клиенты не смогут изменять таблицы, пока вы будете их использовать.
- Запустите запросы, которые должны выполниться как группа.
- Снимите блокировку при помощи `UNLOCK TABLE`. Другие клиенты получат доступ к таблицам.

Блокировки, установленные `LOCK TABLE`, действуют, пока вы их не снимете, в том числе и на период выполнения нескольких предложений. То есть обеспечивается решение проблемы конкуренции, как и в транзакции. Однако нет отката на случай ошибки, поэтому блокировка таблиц подходит не для каждого приложения. Например, вы можете попытаться выполнить операцию передачи денег от Евы Иде так:

```
LOCK TABLE money WRITE;
UPDATE money SET amt = amt - 6 WHERE name = 'Eve';
UPDATE money SET amt = amt + 6 WHERE name = 'Ida';
UNLOCK TABLE;
```

[1] `LOCK TABLES` и `UNLOCK TABLES` – это синонимы `LOCK TABLE` и `UNLOCK TABLE`.

К сожалению, если второе обновление не выполнится, откатить первое будет невозможно. Но несмотря на некоторые ограничения, есть ряд ситуаций, для которых вполне подходят блокировки таблиц:

- Набор предложений, состоящий только из запросов SELECT. Если вы хотите выполнить несколько предложений SELECT и запретить другим клиентам изменять таблицы, пока вы к ним обращаетесь, используйте блокировку. Например, если вам нужно выполнить несколько суммарных запросов для множества таблиц, может оказаться, что итоги формируются для различных наборов данных, если между запросами другие клиенты будут иметь возможность изменять записи. Итоговые данные получатся противоречивыми. Чтобы не допустить этого, заблокируйте таблицы на время использования.
- Блокировку можно использовать для набора предложений, в котором только последнее является обновлением. Тогда предыдущие предложения не вносят никаких изменений в таблицы, и если обновление не выполнится, нет необходимости в откате более ранних предложений.

Переформулируем запросы, чтобы не использовать транзакции

Иногда приложения используют транзакции без необходимости. Предположим, что у вас есть таблица meeting, в которой содержится информация о собраниях и заседаниях (включая количество оставшихся билетов), и вы пишете приложение на Perl, содержащее функцию get_ticket(), которая распределяет билеты. Одним из способов реализации функции является проверка счетчика билетов, его уменьшение, если он положительный, и возвращение статуса, показывающего доступность билета. Чтобы воспрепятствовать попыткам нескольких клиентов одновременно завладеть последним билетиком, будем выдавать запросы в транзакции:[1]

```
sub get_ticket
{
my ($dbh, $meeting_id) = @_;

    my $ref = transact_init ($dbh);
    my $count = 0;
    eval
    {
        # проверяем текущий счетчик билетов
        $count = $dbh->selectrow_array (
                "SELECT tix_left FROM meeting
                WHERE meeting_id = ?", undef, $meeting_id);
        # если билеты остались, уменьшаем счетчик
        if ($count > 0)
        {
            $dbh->do (
```

[1] Функции transact_init() и transact_finish() обсуждаются в рецепте 15.4.

```
                "UPDATE meeting SET tix_left = tix_left-1
                WHERE meeting_id = ?", undef, $meeting_id);
        }
        $dbh->commit ();
    };
    $count = 0 if $@;   # если произошла ошибка, то билетов больше нет
    transact_finish ($dbh, $ref, $@);
    return ($count > 0)
}
```

Функция корректно распределяет билеты, но выполняет ненужную работу. Можно сделать то же самое, не прибегая к транзакции. Будем уменьшать счетчик билетов, только если он положительный, затем проверять, изменило ли предложение строку:

```
sub get_ticket
{
my ($dbh, $meeting_id) = @_;

    my $count = $dbh->do ("UPDATE meeting SET tix_left = tix_left-1
                WHERE meeting_id = ? AND tix_left > 0",
                undef, $meeting_id);
    return ($count > 0);
}
```

В MySQL счетчик строк, возвращаемый предложением `UPDATE`, указывает на количество измененных строк. То есть если билетов на мероприятие не осталось, то `UPDATE` не изменит строки, и счетчик будет равен нулю. Легче определить, доступен ли билет, при помощи одного запроса, чем выполнять несколько запросов, используя транзакционный подход. Мораль в том, что несмотря на всю важность и полезность транзакций, может оказаться, что в них нет необходимости, и, избавившись от них, вы повысите производительность своего приложения. (Решение, состоящее из одного запроса, – это пример того, что справочное руководство по MySQL называет «атомарной операцией». В руководстве такие операции рассматриваются как эффективная альтернатива транзакциям.)

16

Знакомство с MySQL для Web

16.0. Введение

Несколько следующих глав рассказывают о том, как использовать MySQL для создания хорошего веб-сайта. В целом, главное преимущество использования MySQL заключается в упрощении создания динамического содержимого. Статическое содержимое – это страницы дерева документов веб-сервера, которые предлагаются «как есть». Посетители могут получить доступ только к тем документами, которые вы разместили в дереве, и изменения происходят только при добавлении, редактировании или удалении таких документов. Динамическое же содержимое создается по требованию. Вместо того чтобы открывать файл и передавать его содержимое непосредственно клиенту, веб-сервер выполняет сценарий, который формирует страницу и отправляет результирующий вывод. В качестве простого примера можно привести сценарий, который ищет значение текущего счетчика посещаемости указанной веб-страницы в базе данных, обновляет счетчик и возвращает новое значение для отображения на странице. При каждом выполнении сценария выводится новое значение. Более сложным примером могут послужить сценарии, выводящие имена людей, чей день рождения выпадает на текущую дату, извлекающие и выводящие элементы из каталога товаров, предоставляющие информацию о текущем состоянии сервера. И это только начало; веб-сценарии могут использовать всю мощь того языка программирования, на котором написаны, так что спектр действий, выполняемых ими для генерирования страниц, может быть весьма широк. Например, веб-сценарии важны при обработке форм, при этом один сценарий может обеспечивать создание формы, отправку ее пользователю, обработку формы, заполненной и отправленной пользователем, и хранение содержимого в базе данных. Взаимодействуя с пользователями, веб-сценарии делают ваш сайт интерактивным.

В этой главе представлены начальные сведения о написании сценариев, применяющих MySQL в веб-среде. Часть вводного материала не является специфичной для MySQL, но я привожу его для того, чтобы заложить фундамент для использования вашей базы данных в контексте веб-программирования. Здесь рассмотрены следующие вопросы:

- Чем создание веб-сценариев отличается от написания статических документов HTML или сценариев, предназначенных для запуска из командной строки.
- Какие условия необходимы для запуска веб-сценариев. В частности, требуется установленный веб-сервер, настроенный на распознавание ваших сценариев как программ для исполнения, а не статических файлов, которые следует передать по сети неизменными.
- Как использовать API каждого из наших языков для создания небольшого веб-сценария, который запрашивает информацию у сервера MySQL и выводит результаты на веб-страницу.
- Как правильно кодировать вывод. Код HTML состоит из текста, который должен быть выведен, окруженного специальной разметкой. Но если текст содержит специальные символы, необходимо преобразовать их, чтобы избежать некорректного формирования веб-страниц. Каждый API предлагает соответствующий механизм.

Следующие главы подробно рассматривают такие темы, как отображение результатов запроса в различных форматах (абзацы, списки, таблицы и т. д.), работа с изображениями, обработка форм и отслеживание обращений пользователя к нескольким страницам в рамках одного сеанса.

В книге используется веб-сервер Apache для сценариев на Perl, PHP и Python и сервер Tomcat для сценариев на Java в нотации JSP (JavaServer Pages). Оба сервера доступны на сайте Apache Group:

http://httpd.apache.org/
http://jakarta.apache.org/

Поскольку сервер Apache наиболее популярен, я буду считать, что он уже установлен в вашей системе, и требуется только его сконфигурировать. В рецепте 16.2 показано, как настроить Apache для Perl, PHP и Python и как создать небольшой веб-сценарий на каждом из языков. Сервер Tomcat используется не так широко, как Apache, поэтому в приложении B приведена некоторая дополнительная информация о нем. В рецепте 16.3 обсуждается написание сценария JSP при помощи Tomcat. При желании вы можете использовать другие серверы, но тогда вам придется адаптировать для них приведенные в главе инструкции.

Сценарии примеров для Сети включены в дистрибутив `recipes` и находятся в каталогах с именами тех серверов, которые их выполняют. Ищите примеры для Perl, PHP и Python в каталоге *apache*; а примеры для JSP – в каталоге *tomcat*.

Предполагается, что читатель знаком с азами HTML. Для работы с Tomcat неплохо бы знать что-нибудь и об XML, так как конфигурационные файлы Tomcat записываются в виде документов XML, а страницы JSP содержат элементы, написанные в синтаксисе XML. Если вы совсем не знакомы с XML, просмотрите хотя бы примечание «XML и XHTML в двух словах». Вообще, веб-сценарии этой книги формируют вывод, который соответствует

не только формату HTML, но и XHTML – переходному между HTML и XML (еще одна причина познакомиться с XML.) Например, XHTML требует использования закрывающих тегов, поэтому абзацы записываются с помощью

XML и XHTML в двух словах

XML немного похож на HTML, и поскольку многие знают HTML, вероятно, легче всего описать его в контексте отличия от HTML:

- Регистр букв не имеет значения для имен атрибутов и тегов HTML; в XML имена чувствительны к регистру.
- В HTML атрибуты тегов можно указывать со значением, заключенным или не заключенным в кавычки, или даже вообще без значения. В XML каждый атрибут тега должен иметь значение, и это значение должно быть заключено в кавычки.
- Каждому открывающему тегу в XML должен соответствовать закрывающий. Это верно даже для тех случаев, когда тело тега отсутствует, хотя тогда можно использовать сокращенную форму тега. Например, в HTML можно написать `<br>`, но XML требует наличия закрывающего тега. Можно было бы написать `<br></br>`, но этот элемент не имеет тела, поэтому можно использовать сокращенную форму `<br/>`, которая является комбинацией открывающего и закрывающего тегов. Однако если вы пишете XML, который будет транслироваться в HTML, надежнее записать тег как `<br />`, поставив пробел перед символом слэша. Пробел помогает броузеру не сделать ошибки: можно было бы воспринять `br/` как название тега и проигнорировать его как нераспознанный.

XHTML – это переходный формат, используемый при миграции из HTML в XML. Он менее строг, чем XML, но строже, чем HTML. Например, теги и имена атрибутов XHTML должны записываться в нижнем регистре, значения атрибутов должны быть заключены в двойные кавычки.

В HTML можно записать переключатель (radio button) так:

```
<INPUT TYPE=RADIO NAME="my button" VALUE=3 CHECKED>
```

В XHTML название тега должно записываться в нижнем регистре, значения атрибутов должны заключаться в кавычки, атрибут `checked` должен иметь значение, кроме того, необходим закрывающий тег:

```
<input type="radio" name="my button" value="3" checked="checked"></input>
```

В данном случае элемент не имеет тела, поэтому можно использовать сокращенную форму:

```
<input type="radio" name="my button" value="3" checked="checked" />
```

Если вас интересует дополнительная информация об HTML, XHTML или XML, обратитесь к источникам, перечисленным в приложении C.

закрывающего </p>, который следует за телом абзаца. Применение такого стиля вывода в сценариях, создаваемых на языках типа PHP, в которых теги HTML включаются в сценарий как литералы, видно невооруженным глазом. Если же интерфейс генерирует HTML за вас, как модуль Perl CGI.pm, то соответствие XHTML зависит от того, формирует ли сам модуль XHTML. CGI.pm делает это, начиная с версии 2.69, а в более поздних версиях его соответствие XHTML совершенствуется.

16.1. Основы формирования веб-страницы

Задача

Вы хотите сформировать веб-страницу из сценария вместо того, чтобы писать ее вручную.

Решение

Напишите программу, которая при выполнении генерирует страницу. Тем самым вы обеспечите более полный контроль над тем, что отправляется клиенту, чем при создании статической страницы, хотя при этом может возникнуть необходимость более детального составления ответа. Например, может потребоваться написать заголовки, предшествующие телу страницы.

Обсуждение

HTML – это язык разметки (именно поэтому в его названии присутствуют буквы «ML» – Markup Language), состоящий из смеси обычного текста для вывода и специальных указателей разметки или конструкций, управляющих отображением текста. Вот простейшая HTML-страница, которая указывает название в заголовке страницы и тело с белым фоном, содержащее один абзац:

```
<html>
<head>
<title>Web Page Title</title>
</head>
<body bgcolor="white">
<p>Web page body.</p>
</body>
</html>
```

Можно написать сценарий, который формирует такую же страницу, но этот способ будет по нескольким параметрам отличаться от написания статической страницы. Во-первых, вы пишете сразу на двух языках. (Сценарий пишется на вашем языке программирования, а сам сценарий выводит HTML.) Еще одно отличие в том, что может потребоваться формирование более объемного ответа для отправки клиенту. Когда веб-сервер отправляет клиенту статическую страницу, на самом деле сначала отправляется одна или нескольких строк заголовка, несущих дополнительную информацию о странице. Например, документ HTML может начинаться с заголовка Content-Type:,

который информирует клиента о типе документа, и пустой строки, отделяющей заголовок от тела страницы:

```
Content-Type: text/html

<html>
<head>
<title>Web Page Title</title>
</head>
<body bgcolor="white">
<p>Web page body.</p>
</body>
</html>
```

Веб-сервер автоматически формирует заголовочную информацию для статических HTML-страниц. При написании веб-сценария вам, возможно, придется самостоятельно предоставлять такую информацию. Некоторые API (такие как PHP) могут автоматически отправлять заголовок типа содержимого, но разрешают вам заменить тип по умолчанию. Например, если ваш сценарий отправляет клиенту изображение в формате JPEG вместо HTML-страницы, вы можете захотеть изменить тип содержимого сценария с `text/html` на `image/jpeg`.

Создание веб-сценариев отличается и от написания сценариев командной строки, как на входе, так и на выходе. Входная информация поступает в веб-сценарий от веб-сервера, а не задается аргументами командной строки и не вводится с клавиатуры. То есть ваши сценарии получают ввод не за счет использования операторов чтения. Вместо этого веб-сервер помещает информацию в окружение сценария, а сценарий извлекает ее и обрабатывает.

Что касается вывода, сценарии командной строки обычно формируют вывод в виде простого текста, в то время как веб-сценарии генерируют HTML-страницы или другой тип содержимого, который следует отправить клиенту. Вывод, формируемый в веб-среде, обычно должен быть правильно структурированным, чтобы его могли распознать принимающие клиентские программы.

Любой API позволяет формировать вывод при помощи операторов печати, а некоторые также предлагают собственные средства подготовки веб-страниц, которые могут быть встроены непосредственно в API или реализованы в отдельных модулях:

- Для сценариев на Perl часто используется модуль CGI.pm, способный генерировать HTML-разметку, обрабатывать формы и т. д.
- Сценарии PHP пишутся как смесь HTML и вложенного кода на PHP. То есть вы пишете HTML непосредственно в сценарии, затем переходите в «программный режим» всякий раз, когда необходимо сформировать вывод, выполнив код. На результирующей странице, отправляемой клиенту, код заменяется на сгенерированный им вывод.
- Для Python имеются модули `cgi` и `urllib`, помогающие решить задачи веб-программирования.

- Для Java мы будем писать сценарии согласно спецификации JSP, что позволит встраивать директивы сценариев и код в веб-страницы. Аналогично работает PHP.

Доступны и другие пакеты, отличные от используемых в книге, причем некоторые из них могут влиять на то, как будет использоваться язык программирования. Например, Mason, embPerl, ePerl и AxKit позволяют интерпретировать Perl как встроенный язык, что отчасти напоминает работу с PHP. Модуль Apache *mod_snake* разрешает встраивать код Python в шаблоны HTML.

Прежде чем вы сможете запустить какой-либо сценарий в веб-окружении, необходимо правильно настроить ваш веб-сервер. Информация о конфигурировании Apache и Tomcat приводится в рецептах 16.2 и 16.3, пока же отметим, что обычно веб-сервер запускает сценарий одним из двух способов. Во-первых, веб-сервер может использовать для исполнения сценария внешнюю программу. Например, может вызвать экземпляр интерпретатора Python для запуска сценария Python. Во-вторых, если сервер обладает способностью обработки соответствующего языка, он может запустить сценарий самостоятельно. Использование внешней программы для исполнения сценариев не требует никаких специальных возможностей веб-сервера, но этот способ медленнее, так как необходимо запустить отдельный процесс, кроме того, некоторая дополнительная нагрузка возникает за счет передачи информации запросов в сценарий и считывания оттуда результатов. Если интерпретатор встраивается в веб-сервер, он может выполнять сценарии напрямую, в результате чего производительность значительно повышается.

Как и большая часть веб-серверов, Apache умеет запускать внешние сценарии. Кроме того, он поддерживает модули расширения, которые становятся частью самого процесса Apache (их можно встроить при компиляции или динамически подгружать во время исполнения). Эту возможность обычно используют для встраивания в сервер языкового процессора, что ускоряет исполнение сценария. Сценарии Perl, PHP и Python могут выполняться любым из способов. Аналогично сценариям оболочки, веб-сценарии, исполняемые как внешние программы, записываются как исполняемые файлы, начинающиеся со строки `#!` с указанием пути к исполняемому файлу соответствующего языкового интерпретатора. Apache использует этот путь, чтобы определить, какой интерпретатор запускает сценарий. Вы также можете расширить возможности Apache, используя такие модули, как *mod_perl* для Perl, *mod_php* для PHP и *mod_python* или *mod_snake* для Python. Тогда Apache получит возможность непосредственно выполнять сценарии, написанные на этих языках.

Что касается JSP-сценариев Java, они компилируются в сервлеты Java и запускаются внутри процесса, называемого контейнером сервлета. Наблюдается некоторое сходство с встраиванием языкового процессора: сценарии исполняются управляющим ими серверным процессом вместо запуска внешнего процесса для каждого сценария. При первом запросе JSP-страницы клиентом контейнер компилирует ее в сервлет в виде исполнимого байт-кода Java, затем загружает и исполняет. Контейнер кэширует байт-код, так

что при последующих обращениях сценарий запускается сразу же, без компиляции. Если вы изменяете сценарий, контейнер замечает это при получении следующего запроса, тогда сценарий перекомпилируется в новый сервлет и перезагружается. JSP-подход имеет значительное преимущество над созданием сервлетов напрямую, поскольку вам не приходится самим компилировать код и заниматься загрузкой и выгрузкой сервлетов. Tomcat может выполнять функции как контейнера сервлетов и как веб-сервера, взаимодействующего с контейнером.

Если на одном хосте работает несколько серверов, они должны прослушивать разные порты. В стандартной конфигурации Apache прослушивает порт HTTP по умолчанию (80), а Tomcat – 8080. Примеры главы будут использовать такие имена хостов, как *apache.snake.net* и *tomcat.snake.net* для представления URL в сценариях, обрабатываемых Apache и Tomcat. В зависимости от ваших настроек DNS они могут быть или не быть сопоставлены одной и той же физической машине, поэтому в примерах для Tomcat будет использоваться порт с отличным номером – 8080. Обычно в книге будут встречаться URL в такой форме:

http://apache.snake.net/cgi-bin/my_perl_script.pl
http://apache.snake.net/cgi-bin/my_python_script.py
http://apache.snake.net/mcb/my_php_script.php
http://tomcat.snake.net:8080/mcb/my_jsp_script.jsp

Вам нужно будет заменить имя хоста и номер порта на соответствующие значения для ваших собственных серверов.

16.2. Запуск веб-сценариев на сервере Apache

Задача

Вы хотите выполнить программы на Perl, PHP или Python в веб-окружении.

Решение

Запустите их при помощи сервера Apache.

Обсуждение

В этом разделе рассказано о том, как настроить Apache для запуска сценариев Perl, PHP и Python, и показаны примеры веб-сценариев на каждом из этих языков.

Корневой каталог Apache (будем считать, что это каталог */usr/local/apache*) включает следующие каталоги:

bin

Содержит *httpd*, то есть сам Apache, а также другие исполняемые программы.

conf

Конфигурационные файлы, в том числе *httpd.conf*, основной файл, используемый Apache.

htdocs

Корень дерева документов.

logs

Лог-файлы.

Чтобы настроить Apache для выполнения сценариев, отредактируйте файл *httpd.conf* в каталоге *conf*. Обычно исполняемые сценарии характеризуются либо расположением, либо расширением имени файла. Расположение файла может зависеть или не зависеть от языка.

Конфигурации Apache часто содержат внутри корневого каталога сервера каталог *cgi-bin*, где устанавливаются сценарии, которые должны запускаться как внешние программы. Он задается при помощи директивы `ScriptAlias`:

```
ScriptAlias /cgi-bin/ /usr/local/apache/cgi-bin/
```

Второй аргумент – это фактическое местоположение каталога сценариев в вашей файловой системе, а первый – виртуальное имя, которое соответствует этому каталогу. То есть приведенная директива сопоставляет сценарии, расположенные в каталоге */usr/local/apache/cgi-bin* адресам URL, в которых за именем хоста следует *cgi-bin*. Например, если вы инсталлируете сценарий *myscript.py* в каталог */usr/local/apache/cgi-bin* хоста *apache.snake.net*, то сможете запустить его, используя такой URL:

http://apache.snake.net/cgi-bin/myscript.py

В такой конфигурации каталог *cgi-bin* может содержать сценарии, написанные на любом языке. Поэтому каталог нейтрален по отношению к языку, и серверу Apache необходимо как-то понять, какой интерпретатор использовать для выполнения каждого из находящихся в каталоге сценариев. Для обеспечения такой информация первая строка сценария должна начинаться с символов `#!` и пути к программе, которая выполняет сценарий, а также, возможно, каких-то опций. Например, сценарий, который начинается со строки, приведенной ниже, будет запущен при помощи Perl, а опция *-w* указывает Perl на необходимость выдачи предупреждений по поводу сомнительных языковых конструкций:

```
#! /usr/bin/perl -w
```

В UNIX для корректной работы сценария необходимо сделать сценарий исполняемым (посредством команды *chmod +x*). Приведенная строка `#!` подходит для системы, в которой Perl установлен как */usr/bin/perl*. Если интерпретатор Perl установлен в каком-то другом каталоге, измените строку соответствующим образом. Если вы работаете в Windows, и Perl установлен как *D:\Perl\bin\perl.exe*, то строка `#!` должна выглядеть так:

```
#! D:\Perl\bin\perl -w
```

У пользователей Windows есть и другая возможность – задать соответствие расширения имени файла *.pl* интерпретатору Perl. Тогда сценарии могут начинаться так:

```
#! perl -w
```

Директива `ScriptAlias` определяет каталог, который может использоваться для сценариев, написанных на любых языках. Есть также возможность сопоставить каталогу какой-то определенный языковой процессор, тогда любой сценарий каталога будет считаться написанным на данном языке. Например, чтобы назначить каталог */usr/local/apache/cgi-perl* каталогом *mod_perl*, можно сконфигурировать Apache так:

```
<IfModule mod_perl.c>
    Alias /cgi-perl/ /usr/local/apache/cgi-perl/
    <Location /cgi-perl>
        SetHandler perl-script
        PerlHandler Apache::Registry
        PerlSendHeader on
        Options +ExecCGI
    </Location>
</IfModule>
```

Тогда можно вызывать расположенные в каталоге сценарии Perl так:

http://apache.snake.net/cgi-perl/myscript.pl

Использование *mod_perl* выходит за рамки нашего обсуждения, так что не будем о нем больше говорить. Некоторые источники полезной информации о *mod_perl* приведены в приложении С.

Каталоги, используемые исключительно для хранения сценариев, обычно располагаются вне дерева документов Apache. Альтернативой использования специальных каталогов для сценариев может служить идентификация сценариев по расширению имени файла: имена с определенным расширением сопоставляются указанному языковому процессору. Тогда такие файлы можно размещать в дереве документов где угодно. Обычно именно так работают с PHP. Например, если ваш сервер Apache сконфигурирован со встроенной поддержкой PHP посредством модуля *mod_php*, вы можете указать ему, что сценарии, имена которых заканчиваются на *.php*, должны интерпретироваться как сценарии PHP. Для этого добавим в *httpd.conf* такую строку:

```
AddType application/x-httpd-php .php
```

Если теперь установить сценарий PHP *myscript.php* в *htdocs* (корневой каталог документов Apache), то URL для вызова сценария будет таким:

http://apache.snake.net/myscript.php

Если PHP запускается как внешняя автономная программа, необходимо сообщить Apache, где ее следует искать. Например, если вы работаете в Windows, а PHP инсталлирован как *D:\Php\php.exe*, поместите в *httpd.conf* сле-

дующие строки (обратите внимание на использование в путях символов прямого слэша, а не обратного):

```
ScriptAlias /php/ "D:/Php/"
AddType application/x-httpd-php .php
Action application/x-httpd-php /php/php.exe
```

Приводя URL в примерах, я буду считать, что сценарии Perl и Python находятся в каталоге *cgi-bin*, а сценарии PHP – в каталоге *mcb* вашего дерева документов, и имеют расширение *.php*. То есть URL сценариев на этих языках будут выглядеть так:

http://apache.snake.net/cgi-bin/myscript.pl
http://apache.snake.net/cgi-bin/myscript.py
http://apache.snake.net/mcb/myscript.php

Если вы планируете использовать подобные настройки, убедитесь в том, что в вашем корневом каталоге Apache есть каталог *cgi-bin*, а в корневом каталоге документов Apache – каталог *mcb*. Затем для работы с веб-сценариями Perl или Python следует скопировать их в *cgi-bin*, а сценарии PHP – в *mcb*.

Замечание о безопасности в Web

В UNIX сценарии запускаются с определенными идентификаторами пользователя и группы. Сценарии, исполняемые из командной строки, запускаются с вашим идентификаторам пользователя (UID) и группы (GID) и имеют права доступа к файловой системе, определяемые вашей учетной записью. Однако сценарии, исполняемые веб-сервером, скорее всего не будут иметь ваших прав доступа и не будут связаны с вашими идентификаторами. Вместо этого они получат UID, GID и права доступа той учетной записи, под которой запускается веб-сервер. (Чтобы определить эту учетную запись, найдите в конфигурационном файле *httpd.conf* директивы `User` и `Group`.) Это означает, что если вы рассчитываете на то, что веб-сценарии смогут читать и записывать файлы, то эти файлы должны быть доступны учетной записи, под которой работает веб-сервер. Например, если сервер работает под учетной записью `nobody`, и вы хотите, чтобы сценарий мог сохранять загруженные файлы изображений в каталоге *uploads* дерева документов, то необходимо сделать этот каталог доступным на чтение и запись для пользователя `nobody`.

Это означает, что если другие пользователи могут писать сценарии для вашего веб-сервера, то их сценарии будут также выполняться под пользователем `nobody`, и они смогут читать и записывать те же файлы, что и ваши собственные сценарии. Проблема решается использованием механизма suEXEC в Apache 1.x или использованием Apache 2.x, который позволяет определить, под какими UID и GID должен выполняться заданный набор сценариев.

Если вы обращаетесь к веб-сценарию, а в ответ получаете страницу с ошибкой, обратитесь к логу ошибок Apache, из которого можно получить важную информацию о том, почему сценарий не работает. Обычно лог ошибок хранится в файле *error_log* каталога *logs*. Если вы не нашли такой файл, посмотрите в конфигурационном файле *httpd.conf*, как установлена директива `ErrorLog`.

Сконфигурировав Apache для выполнения сценариев, вы можете приступить к их написанию для генерации веб-страниц. В оставшейся части данного раздела рассказывается, как это сделать на Perl, PHP и Python. В примерах для каждого из языков устанавливается соединение с сервером MySQL, выполняется запрос `SHOW TABLES` и результат отображается на веб-странице. В приведенных здесь сценариях указаны дополнительные модули и библиотеки, которые должны быть в них включены. (В дальнейшем будем предполагать, что соответствующие модули уже подключены, и показывать только фрагменты сценариев.)

Perl

Ниже приведен наш первый веб-сценарий на Perl, *show_tables.pl*. Он создает заголовок в соответствии с указанным `Content-Type:`, пустую строку, отделяющую заголовок от тела страницы, и начальную часть страницы. Затем сценарий получает и отображает список таблиц базы данных `cookbook`. После списка таблиц выводятся завершающие HTML-теги, закрывающие страницу:

```
#! /usr/bin/perl -w
# show_tables.pl - Выполнить запрос SHOW TABLES и показать результат,
# генерируя непосредственно HTML

use strict;
use lib qw(/usr/local/apache/lib/perl);
use Cookbook;

# Вывести заголовок, пустую строку и начало страницы

print <<EOF;
Content-Type: text/html

<html>
<head>
<title>Tables in cookbook Database</title>
</head>
<body bgcolor="white">

<p>Tables in cookbook database:</p>
EOF

# Соединиться с БД, показать список таблиц, отключиться

my $dbh = Cookbook::connect ();
my $sth = $dbh->prepare ("SHOW TABLES");
$sth->execute ();
while (my @row = $sth->fetchrow_array ())
{
    print "$row[0]<br />\n";
```

```
}
$dbh->disconnect ();

# Вывести завершающие теги

print <<EOF;
</body>
</html>
EOF

exit (0);
```

Для проверки сценария поместите его в каталог *cgi-bin* и вызовите из броузера так:

http://apache.snake.net/cgi-bin/show_tables.pl

В сценарии *show_tables.pl* HTML генерируется включением в операторы вывода литералов, представляющих теги. Другой подход к генерации веб-страниц реализован в модуле CGI.pm, который упрощает создание веб-сценариев, не требуя буквального написания тегов. Модуль CGI.pm предлагает как объектно-ориентированный, так и функциональный интерфейс, позволяя создавать веб-страницы любым из этих методов. Сценарий *show_tables_oo.pl* использует объектно-ориентированный интерфейс для создания того же отчета, что и *show_tables.pl*:

```
#! /usr/bin/perl -w
# show_tables_oo.pl - Выполнить запрос SHOW TABLES и показать результат,
# используя объектно-ориентированный интерфейс CGI.pm

use strict;
use lib qw(/usr/local/apache/lib/perl);
use CGI;
use Cookbook;

# Создать CGI объект для доступа к методам CGI.pm

my $cgi = new CGI;

# Вывести заголовок, пустую строку и начало страницы

print $cgi->header ();
print $cgi->start_html (-title => "Tables in cookbook Database", -bgcolor => "white");

print $cgi->p ("Tables in cookbook database:");

# Соединиться с БД, показать список таблиц, отключиться

my $dbh = Cookbook::connect ();
my $sth = $dbh->prepare ("SHOW TABLES");
$sth->execute ();
while (my @row = $sth->fetchrow_array ())
{
    print $row[0] . $cgi->br ();
}
$dbh->disconnect ();
```

```
# Вывести завершающие теги

print $cgi->end_html ();

exit (0);
```

Сценарий включает модуль CGI.pm, используя директиву `use CGI`, затем создает `CGI`-объект `$cgi`, с помощью которого обращается к различным методам генерации HTML. Метод `header()` создает заголовок `Content-Type:`, а метод `start_html()` формирует начальную часть страницы вплоть до открывающего тега `<body>`. После создания первой части страницы сценарий получает информацию с сервера базы данных и отображает ее. Имя каждой таблицы сопровождается тегом `<br />`, формируемым вызовом метода `br()`. Метод `end_html()` выводит завершающие теги `</body>` и `</html>`. Установив сценарий в каталог *cgi-bin* и вызвав его из броузера, вы увидите, что он генерирует такую же страницу, как и *show_tables.pl*.

Функции CGI.pm могут принимать множество параметров, часть которых необязательны. Чтобы позволить вам указывать только необходимые параметры, CGI.pm разрешает использовать в списке параметров нотацию *-имя => значение*. Например, в вызове `start_html()` параметр `title` назначает заголовок страницы, а `bgcolor` — цвет фона. Нотация *-имя => значение* позволяет указывать параметры в любом порядке, так что два следующих предложения эквиваленты:

```
print $cgi->start_html (-title => "My Page Title", -bgcolor => "white");
print $cgi->start_html (-bgcolor => "white", -title => "My Page Title");
```

Чтобы использовать функциональный, а не объектно-ориентированный интерфейс CGI.pm, следует немного изменить сценарии. Строка `use`, ссылающаяся на CGI.pm, импортирует названия методов в пространство имен сценария, и вы можете вызывать их как функции, не создавая объект CGI. Например, чтобы импортировать наиболее распространенные методы, сценарий должен содержать такое предложение:

```
use CGI qw(:standard);
```

Сценарий *show_tables_fc.pl* — это эквивалент только что рассмотренного *show_tables_oo.pl*, но написанный с использованием вызовов функций. Применяются те же методы CGI.pm, но вызываются они как автономные функции, а не через объект `$cgi`:

```
#! /usr/bin/perl -w
# show_tables_fc.pl - Выполнить запрос SHOW TABLES и показать результат,
# используя функциональный интерфейс CGI.pm

use strict;
use lib qw(/usr/local/apache/lib/perl);
use CGI qw(:standard);  # импортировать имена стандартных методов
                        # в пространство имен сценария
use Cookbook;

# Вывести заголовок, пустую строку и начало страницы

print header ();
```

```
print start_html (-title => "Tables in cookbook Database", -bgcolor => "white");

print p ("Tables in cookbook database:");

# Соединиться с БД, показать список таблиц, отключиться

my $dbh = Cookbook::connect ();
my $sth = $dbh->prepare ("SHOW TABLES");
$sth->execute ();
while (my @row = $sth->fetchrow_array ())
{
    print $row[0] . br ();
}
$dbh->disconnect ();

# Вывести завершающие теги

print end_html ();

exit (0);
```

Поместите сценарий *show_tables_fc.pl* в каталог *cgi-bin* и убедитесь в том, что он формирует такой же вывод, как и *show_tables_oo.pl*.

Далее в книге для веб-сценариев Perl используется функциональный интерфейс CGI.pm. Если вас интересует дополнительная информация о CGI.pm, вы можете выполнить из командной строки следующие команды, которые обеспечат доступ к имеющейся документации:

```
% perldoc CGI
% perldoc CGI::Carp
```

Другие источники информации об этом модуле, как печатные, так и доступные в Интернете, приведены в приложении C.

PHP

Как это ни удивительно для веб-ориентированного языка, PHP не предоставляет возможностей сокращенного задания тегов. Но поскольку PHP является встраиваемым языком, вы можете просто буквально писать HTML в сценарии, не пользуясь предложением `print`. Рассмотрим сценарий *show_tables.php*, который переключается между режимами HTML и PHP:

```
<?php
# show_tables.php - Выполнить запрос SHOW TABLES и показать результат

include "Cookbook.php";

?>

<html>
<head>
<title>Tables in cookbook Database</title>
</head>
<body bgcolor="white">

<p>Tables in cookbook database:</p>
```

```
<?php

# Соединиться с БД, показать список таблиц, отключиться
$conn_id = cookbook_connect ();
$result_id = mysql_query ("SHOW TABLES", $conn_id);
while (list ($tbl_name) = mysql_fetch_row ($result_id))
    print ("$tbl_name<br />\n");
mysql_free_result ($result_id);
mysql_close ($conn_id);

?>

</body>
</html>
```

Чтобы протестировать сценарий, поместим его в каталог *mcb* дерева документов веб-сервера и вызовем его так:

http://apache.snake.net/mcb/show_tables.php

В отличие от Perl-версий сценариев вывода списка таблиц MySQL, сценарий PHP не содержит кода для вывода заголовка `Content-Type:`, так как PHP формирует его автоматически. (Если вас не устраивает такое положение дел, и вы хотите выводить свои заголовки, обратитесь к описанию функции `header()` в руководстве по PHP.)

Если не считать тегов переноса строки, все содержимое HTML в сценарии *show_tables.php* размещается за пределами тегов `<?php` и `?>`, так что интерпретатор PHP просто выводит его без интерпретирования. Приведем версию сценария, которая формирует весь HTML при помощи предложений `print`:

```
<?php
# show_tables_print.php - Выполнить запрос SHOW TABLES и показать результат,
# используя print() для формирования всего HTML

include "Cookbook.php";

print ("<html>\n");
print ("<head>\n");
print ("<title>Tables in cookbook Database</title>\n");
print ("</head>\n");
print ("<body bgcolor=\"white\">\n");
print ("<p>Tables in cookbook database:</p>\n");

# Соединиться с БД, показать список таблиц, отключиться
$conn_id = cookbook_connect ();
$result_id = mysql_query ("SHOW TABLES", $conn_id);
while (list ($tbl_name) = mysql_fetch_row ($result_id))
    print ("$tbl_name<br />\n");
mysql_free_result ($result_id);
mysql_close ($conn_id);

print ("</body>\n");
print ("</html>\n");
?>
```

Иногда разумно применить один подход, иногда – другой, а иногда даже оба сразу в одном и том же сценарии. Если раздел HTML не ссылается ни на какие значения переменных и выражений, то, вероятно, предпочтительнее записать его в режиме HTML. В противном случае лучше использовать предложение `print` или `echo`, чтобы избежать частого переключения между режимами HTML и PHP.

Python

Стандартная инсталляция Python включает полезные для веб-программирования модули `cgi` и `urllib`. Однако на самом деле они нам пока не нужны, поскольку единственное, что будет делать наш первый веб-сценарий на Python, – генерация простого HTML-кода. Напишем на Python версию сценария, выводящего список таблиц MySQL:

```
#! /usr/bin/python
# show_tables.py - Выполнить запрос SHOW TABLES и показать результат

import sys
sys.path.insert (0, "/usr/local/apache/lib/python")
import MySQLdb
import Cookbook

# Вывести заголовок, пустую строку и начало страницы

print """Content-Type: text/html

<html>
<head>
<title>Tables in cookbook Database</title>
</head>
<body bgcolor="white">

<p>Tables in cookbook database:</p>
"""

# Соединиться с БД, показать список таблиц, отключиться

conn = Cookbook.connect ()
cursor = conn.cursor ()
cursor.execute ("SHOW TABLES")
for (tbl_name, ) in cursor.fetchall ():
    print tbl_name + "<br />"
cursor.close ()
conn.close ()

# Вывести завершающие теги
print """
</body>
</html>
"""
```

Поместите сценарий в каталог Apache *cgi-bin* и вызовите его так:

http://apache.snake.net/cgi-bin/show_tables.py

16.3. Запуск веб-сценариев на сервере Tomcat

Задача

Вы хотите запустить программу на Java в веб-среде.

Решение

Пишите программы, используя нотацию JSP, и выполняйте их, используя контейнер сервлетов.

Обсуждение

В рецепте 16.2 говорилось о том, что сервер Apache можно использовать для запуска сценариев Perl, PHP и Python. Для Java необходим другой подход, так как Apache не может запускать страницы JSP. Поэтому будем работать с сервером Tomcat, предназначенным для обработки Java в веб-среде. Apache и Tomcat – очень разные серверы, но они связаны семейными узами – Tomcat входит в проект Jakarta, которым занимается Apache Software Foundation.

В этом разделе приводится обзор JSP-программирования с использованием Tomcat, причем я буду исходить из следующих предположений:

- У вас есть некоторое представление об основах JavaServer Pages, например, вы знаете, что такое контейнер сервлета, контекст приложения и каковы базовые элементы сценариев JSP.
- Сервер Tomcat установлен так, что вы можете исполнять страницы JSP и знаете, как его запустить и остановить.
- Вы знакомы с каталогом Tomcat *webapps* и знаете, какова структура приложения Tomcat. В частности, вы понимаете, что представляют собой каталог *WEB-INF* и файл *web.xml*.
- Вы знаете, что такое библиотека тегов и как ее использовать.

Признаю, что сделанные допущения достаточно серьезны, поскольку JSP и Tomcat используются в мире MySQL не так широко, как другие языки в сочетании с Apache. Если вы не знакомы с JSP или вам необходима помощь в установке Tomcat, обратитесь за информацией к приложению B.

После того как Tomcat установлен, следует инсталлировать перечисленные ниже компоненты, чтобы иметь возможность работать с JSP-примерами:

- Приложение `mcb` из каталога *tomcat* дистрибутива `recipes`.
- Драйвер JDBC для MySQL. Может быть, вы уже устанавливали его для сценариев из предыдущих глав, но для Tomcat необходим собственный экземпляр.
- Библиотека стандартных тегов JSP (JSTL – JSP Standard Tag Library), которая содержит теги условных конструкций, итеративных операций и работы с базами данных для страниц JSP.

Давайте поговорим о том, как установить эти компоненты, затем рассмотрим некоторые теги JSTL и, наконец, напишем JSP-эквивалент сценария вывода списка таблиц MySQL, реализованный в рецепте 16.2 на языках Perl, PHP и Python.

Установка приложения mcb

Веб-приложения для Tomcat обычно упаковываются в файлы WAR (веб-архив) и устанавливаются в каталоге *webapps*, который напоминает корневой каталог дерева документов Apache *htdocs*. Дистрибутив `recipes` содержит пример приложения `mcb`, которое можно использовать для опробования приведенных примеров JSP. Скопируйте файл *mcb.war* из каталога *tomcat* дистрибутива в каталог Tomcat *webapps*.

Считаем, что дистрибутив `recipes` и Tomcat расположены в каталогах */u/paul/recipes* и */usr/local/jakarta-tomcat*, тогда команда установки *mcb.war* для UNIX будет такой:

```
% cp /u/paul/recipes/tomcat/mcb.war /usr/local/jakarta-tomcat/webapps
```

В Windows, если соответствующие каталоги – это *D:\recipes* и *D:\jakarta-tomcat*, то команда будет такой:

```
C:\> copy D:\recipes\tomcat\mcb.war D:\jakarta-tomcat\webapps
```

После того как файл *mcb.war* скопирован в каталог *webapps*, перезапустите Tomcat. Обычно изначально Tomcat по умолчанию настроен так, чтобы при запуске искать файлы WAR в каталоге *webapps* и автоматически распаковывать все, что еще не распаковано. То есть копирование *mcb.war* в каталог *webapps* и перезапуск Tomcat должны привести к распаковке приложения `mcb`. Когда загрузка Tomcat завершится, вы должны обнаружить в каталоге *webapps* новый каталог *mcb*, включающий все файлы из *mcb.war*. (Если Tomcat автоматически не распаковывает *mcb.war*, обратитесь к примечанию «Как распаковать файл WAR вручную».) Давайте посмотрим на каталог *mcb*. Он должен содержать файлы, которые могут понадобиться клиентам, использующим броузер. Кроме того, должен присутствовать подкаталог *WEB-INF*, предназначенный для информации с ограниченным доступом, то есть доступный для использования сценариям каталога *mcb*, но клиенты непосредственного доступа к нему не имеют.

Теперь проверим, может ли Tomcat обслуживать страницы контекста приложения `mcb`, запросив какие-нибудь страницы из броузера. Следующие URL запрашивают статическую HTML-страницу, сервлет и простую страницу JSP соответственно:

http://tomcat.snake.net:8080/mcb/test.html
http://tomcat.snake.net:8080/mcb/servlet/SimpleServlet
http://tomcat.snake.net:8080/mcb/simple.jsp

Измените значения имени хоста и номера порта в URL на значения вашей системы.

Как распаковать файл WAR вручную

Файлы WAR – это архивы в формате ZIP, которые могут быть распакованы при помощи *jar*, *WinZip* или любого другого средства, распознающего файлы ZIP. Однако при ручной распаковке файла WAR вам нужно будет сначала создать его каталог верхнего уровня. Приведем последовательность шагов, представляющую один из способов выполнения операции. Используем утилиту *jar* для распаковки файла WAR с именем *mcb.war*, расположенного в каталоге Tomcat *webapps*. Если вы работаете в UNIX, перейдите в каталог *webapps*, затем выполните такие команды:

```
% mkdir mcb
% cd mcb
% jar xf ../mcb.war
```

В Windows команды почти такие же:

```
C:\> mkdir mcb
C:\> cd mcb
C:\> jar xf ..\mcb.war
```

Распаковка файла WAR в каталоге *webapps* создает новый контекст приложения, так что вам нужно будет перезапустить Tomcat, чтобы он заметил появление нового приложения.

Установка драйвера JDBC

Для соединения с базой данных `cookbook` страницам JSP приложения `mcb` нужен драйвер JDBC. Рассмотрим процедуру установки драйвера MySQL Connector/J, для других драйверов все будет аналогично.

Чтобы установить MySQL Connector/J для использования приложениями Tomcat, скопируйте его в дерево каталогов Tomcat. Если считать, что драйвер заархивирован в JAR-файл (в случае MySQL Connector/J это именно так), в корневом каталоге Tomcat есть три места, где его можно разместить, в зависимости от того, насколько доступным вы хотели бы его сделать:

- Чтобы драйвер был доступен только приложению `mcb`, поместите его в подкаталог *mcb/WEB-INF/lib* каталога Tomcat *webapps*.
- Чтобы драйвер был доступен всем приложениям Tomcat, но не самому Tomcat, поместите его в подкаталог *lib* корневого каталога Tomcat.
- Чтобы сделать драйвер доступным как приложениям, так и Tomcat, поместите его в подкаталог *common/lib* корневого каталога Tomcat.

Я рекомендовал бы скопировать драйвер в каталог *common/lib*. Тогда он будет наиболее доступен (и приложениям, и Tomcat), и процедуру инсталляции потребуется произвести всего однажды. Если разрешить использование драйвера только приложению `mcb`, поместив его в *mcb/WEB-INF/lib*, а затем создавать другие приложения, использующие MySQL, то придется копировать

драйверы в каталоги этих приложений или перемещать его в более доступное место.

Разумно сделать драйвер наиболее доступным и если вы предполагаете, что однажды решите использовать управление сеансами или аутентификацию области на основе JDBC. Такие операции выполняются самим сервером Tomcat поверх приложений, поэтому необходимо будет обеспечить доступ Tomcat к драйверу.

Приведем пример процедуры инсталляции для UNIX, предполагая, что драйвер MySQL Connector/J и Tomcat расположены в */src/Java/mysql-connector-java-bin.jar* и */usr/local/jakarta-tomcat*. Команда копирования драйвера будет такой:

```
% cp /src/Java/mysql-connector-java-bin.jar /usr/local/jakarta-tomcat/common/lib
```

Для Windows, если компоненты расположены в *D:\mysql-connector-java-bin.jar* и *D:\jakarta-tomcat*, команда выглядит так:

```
C:\> copy D:\mysql-connector-java-bin.jar D:\jakarta-tomcat\common\lib
```

После установки драйвера перезапустите Tomcat, а затем запросите страницу приложения `mcb`, чтобы убедиться в том, что Tomcat находит драйвер JDBC:

http://tomcat.snake.net:8080/mcb/jdbc_test.jsp

Может быть, придется предварительно отредактировать файл *jdbc_test.jsp*, чтобы изменить параметры соединения.

Установка JSTL

Большинство сценариев приложения `mcb` используют библиотеку JSTL, поэтому необходимо установить ее, или сценарии не будут работать. Чтобы поместить библиотеку тегов в контекст приложения, скопируйте файлы библиотеки в каталог приложения *WEB-INF*. Обычно необходимо установить хотя бы один JAR-файл и файл дескриптора библиотеки тегов (TLD – tag library descriptor) и добавить информацию о библиотеке в файл приложения *web.xml*. На самом деле JSTL состоит из нескольких наборов тегов, так что есть несколько файлов JAR и TLD. Процедура установки JSTL для использования с приложением `mcb` такова:

- Убедитесь в том, что файл *mcb.war* был распакован, и иерархия каталогов приложения `mcb` создана внутри каталога Tomcat *webapps* (см. раздел «Установка приложения mcb»). Это необходимо, поскольку файлы JSTL должны размещаться в каталоге *mcb/WEB-INF*, который не будет существовать до тех пор, пока не распакован *mcb.war*.
- Скачайте дистрибутив JSTL с сайта Jakarta Project. Зайдите на страницу проектов Jakarta Taglibs по адресу:

  http://jakarta.apache.org/taglibs/

  Следуйте по ссылке Standard Taglib для получения информационной страницы JSTL, содержащей раздел «Downloads», откуда можно получить двоичный дистрибутив JSTL.

- Распакуйте дистрибутив JSTL в какой-нибудь каталог, желательно вне иерархии Tomcat. Для этого выполните команды, подобные использованным для распаковки самого Tomcat (см. раздел «Установка сервера Tomcat» в приложении В). Например, для распаковки дистрибутива в формате ZIP выполните следующую команду, при необходимости заменив имя файла:

```
% jar xf jakarta-taglibs-standard.zip
```

- Распаковка дистрибутива приведет к созданию каталога, содержащего несколько файлов. Скопируйте файлы JAR (*jstl.jar*, *standard.jar* и т. д.) в каталог *mcb/WEB-INF/lib*. Эти файлы содержат библиотеки классов, которые реализуют действия тегов JSTL. Скопируйте файлы дескрипторов библиотеки тегов (*c.tld*, *sql.tld* и т. д.) в каталог *mcb/WEB-INF*. Эти файлы определяют интерфейс для действий, реализуемых классами в файлах JAR.
- Каталог *mcb/WEB-INF* содержит файл *web.xml*, являющийся файлом дескриптора инсталляции веб-приложения (необычное название для «файла конфигурации»). Измените *web.xml*, добавив записи `<taglib>` для каждого из файлов JSTL TLD:

```
<taglib>
    <taglib-uri>http://java.sun.com/jstl/core</taglib-uri>
    <taglib-location>/WEB-INF/c.tld</taglib-location>
</taglib>
<taglib>
    <taglib-uri>http://java.sun.com/jstl/sql</taglib-uri>
    <taglib-location>/WEB-INF/sql.tld</taglib-location>
</taglib>
```

  Каждый экземпляр `<taglib>` содержит элемент `<taglib-uri>`, указывающий символическое имя, по которому JSP-страницы `mcb` будут ссылаться на соответствующий файл TLD, и элемент `<taglib-location>`, указывающий местоположение файла TLD внутри каталога приложения. (Файл *web.xml* из дистрибутива уже содержит эти записи. Однако необходимо просмотреть их, чтобы убедиться в том, что они соответствуют названиям файлов TLD, установленных на предыдущем этапе.)
- Каталог *mcb/WEB-INF* содержит и файл с именем *jstl-mcb-setup.inc*. Данный файл не входит в состав JSTL, но содержит тег JSTL `<sql:setDataSource>`, который используется многими JSP-страницами `mcb` для указания источника данных при соединении с базой данных `cookbook`. Этот файл выглядит так:

```
<sql:setDataSource var="conn"
    driver="com.mysql.jdbc.Driver" url="jdbc:mysql://localhost/cookbook"
    user="cbuser" password="cbpass" />
```

  При необходимости вы можете отредактировать атрибуты тегов `driver`, `url`, `user` и `password` для изменения параметров соединения на те, которые вы используете для обращения к базе данных `cookbook`. Но не меняйте атрибут `var`.

- Дистрибутив JSTL также включает файлы WAR, содержащие документацию и примеры (*standard-doc.war* и *standard-examples.war*). Если вы хотите установить их, то скопируйте в каталог Tomcat *webapps*. (Вероятно, следует установить документацию, чтобы можно было получить к ней доступ локально с собственного сервера. Полезно установить и примеры, так как они являются наглядной иллюстрацией использования тегов JSTL на страницах JSP.)
- Перезапустите Tomcat, чтобы он заметил изменения, сделанные в приложении `mcb`, и распаковал файлы, содержащие документацию и примеры по JSTL. Если Tomcat автоматически не распаковывает файлы WAR, обратитесь к примечанию «Как распаковать файл WAR вручную».

После установки JSTL перезапустите Tomcat и запросите следующую страницу приложения `mcb` с тем, чтобы проверить, может ли Tomcat корректно определить расположение тегов JSTL:

http://tomcat.snake.net:8080/mcb/jstl_test.jsp

Создание страниц JSP с помощью JSTL

В разделе обсуждается синтаксис некоторых тегов JSTL, наиболее часто используемых JSP-страницами `mcb`. Описания будут очень краткими, а многие из тегов имеют дополнительные атрибуты, позволяющие использовать их не так, как описано в разделе. Подробную информацию о тегах вы сможете найти в спецификации JSTL (см. приложение C).

JSP-страница, использующая JSTL, должна содержать директиву `taglib` для каждого набора тегов, используемого страницей. Примеры книги используют базовые теги и теги работы с базой данных, определяемые следующими директивами `taglib`:

```
<%@ taglib uri="http://java.sun.com/jstl/sql" prefix="sql" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
```

Значения `uri` должны соответствовать символическим значениям, которые перечислены в записях *web.xml* `<taglib>` (см. раздел «Установка JSTL»). Значение `prefix` задает начальную строку, используемую в названиях тегов для идентификации принадлежности тега к определенной библиотеке тегов.

Теги JSTL записываются в формате XML с использованием специального синтаксиса для включения выражений в атрибуты тегов. Внутри атрибутов тегов текст интерпретируется буквально, если только он не заключен в `${...}` – тогда текст воспринимается как выражение, которое должно быть вычислено.

Перечислим теги, входящие в базовый набор JSTL:

`<c:out>`

Тег вычисляет свой атрибут `value` и заменяется результатом. Одним из применений этого тега является предоставление содержимого страницы вывода. Следующий тег выводит значение 3:

```
<c:out value="${1+2}" />
```

<c:set>

Тег присваивает значение переменной. Например, чтобы присвоить символьную строку переменной с именем title, а затем использовать переменную в элементе <title> страницы вывода, сделайте следующее:

```
<c:set var="title" value="JSTL Example Page" />

<html>
<head>
<title><c:out value="${title}" /></title>
</head>
...
```

Этот пример иллюстрирует правило, обычно справедливое для тегов JSTL: при указании переменной, в которой будет храниться значение, не используйте для ее имени нотацию ${...}. Для последующих ссылок на значение этой переменной используйте ее внутри ${...}, чтобы она интерпретировалась как выражение для вычисления.

<c:if>

Тег оценивает условную конструкцию, приведенную в его атрибуте test. Если результат – истина, то вычисляется тело тега, которое и становится его выводом; в противном случае тело тега игнорируется:

```
<c:if test="${1 != 0}">
1 is not equal to 0
</c:if>
```

К операторам сравнения относятся ==, !=, <, >, <= и >=. Альтернативные операторы eq, ne, lt, gt, le и ge позволяют избежать использования специальных символов HTML в выражениях. Арифметические операторы включают +, -, *, / (или div) и % (или mod). К логическим операторам относятся && (and), || (or) и ! (not). Специальный пустой оператор empty истинен, если его значение пусто или равно null:

```
<c:set var="x" value="" />
<c:if test="${empty x}">
x is empty
</c:if>
<c:set var="y" value="hello" />
<c:if test="${!empty y}">
y is not empty
</c:if>
```

<c:choose>

Это еще один условный тег, но он разрешает проверку нескольких условий. Добавьте тег <c:when> для каждого условия, которое вы хотите проверить явно, и <c: otherwise> – если речь идет о случае по умолчанию:

```
<c:choose>
    <c:when test="${count == 0}">
        Please choose an item
    </c:when>
```

```
        <c:when test="${count > 1}">
            Please choose only one item
        </c:when>
        <c:otherwise>
            Thank you for choosing an item
        </c:otherwise>
    </c:choose>
```

`<c:forEach>`

Данный тег работает как итератор, обеспечивая циклический просмотр множества значений. Следующий пример использует тег `<c:forEach>` для обхода набора строк результирующего множества запроса (представленного переменной `rs`):

```
<c:forEach var="row" items="${rs.rows}">
    id = <c:out value="${row.id}" />,
    name = <c:out value="${row.name}" />
    <br />
</c:forEach>
```

Каждый шаг цикла присваивает текущую строку переменной `row`. Если предположить, что результат запроса содержит столбцы `id` и `name`, то для доступа к значениям столбцов применяйте `row.id` и `row.name`.

JSTL-теги для работы с базой данных используются для запуска запросов и обращения к их результатам:

`<sql:setDataSource>`

Этот тег задает параметры соединения для взаимодействия JSTL с сервером базы данных. Например, чтобы указать параметры, используемые драйвером MySQL Connector/J для доступа к базе данных `cookbook`, тег должен выглядеть так:

```
<sql:setDataSource var="conn"
    driver="com.mysql.jdbc.Driver" url="jdbc:mysql://localhost/cookbook"
    user="cbuser" password="cbpass" />
```

Атрибуты `driver`, `url`, `user` и `password` определяют параметры соединения, а атрибут `var` задает переменную, сопоставленную соединению. Условимся, что JSP-страницы в этой книге будут использовать переменную `conn`, так что встречающиеся далее теги, которым необходим источник данных, могут ссылаться на соединение при помощи выражения `${conn}`.

Чтобы не перечислять параметры соединения на каждой странице JSP, использующей MySQL, поместим тег `<sql: setDataSource>`, характеризующий соединение с базой данных `cookbook`, во включаемый файл *WEB-INF/jstl-mcb-setup.inc*. Для установки соединения JSP-страница может обращаться к этому файлу так:

```
<%@ include file="/WEB-INF/jstl-mcb-setup.inc" %>
```

Для того чтобы изменить параметры соединения, используемые страницами `mcb`, просто отредактируйте *jstl-mcb-setup.inc*.

`<sql:update>`

Для запуска предложения, не возвращающего строк, такого как `UPDATE`, `DELETE` или `INSERT`, используйте тег `<sql:update>`. Атрибут тега `dataSource` определяет источник данных; счетчик строк, обработанных запросом, возвращается в переменную, указанную в атрибуте `var`, а само предложение необходимо указать в теле тега:

```
<sql:update var="count" dataSource="${conn}">
    DELETE FROM profile WHERE id > 100
</sql:update>
Number of rows deleted: <c:out value="${count}" />
```

`<sql:query>`

Для обработки запросов, возвращающих результирующее множество, используйте тег `<sql:query>`. Как и в случае с `<sql:update>`, текст запроса приводится в теле тега, а атрибут `dataSource` указывает источник данных. Тег `<sql:query>` также принимает атрибут `var`, в котором указывается переменная, сопоставляемая результирующему множеству:

```
<sql:query var="rs" dataSource="${conn}">
    SELECT id, name FROM profile ORDER BY id
</sql:query>
```

JSP-страницы `mcb` используют для результирующего множества переменную `rs`. О способах доступа к содержимому результирующего множества будет рассказано далее.

`<sql:param>`

Вы можете вставлять значения данных в строку запроса как литералы, но JSTL позволяет использовать и заполнители, что удобно при работе со значениями, содержащие специальные символы SQL. Используйте символ `?` для каждого заполнителя в строке запроса, а значения для связывания с заполнителями указывайте в тегах `<sql:param>` внутри тела тега запроса. Значения данных можно задавать в теле тега `<sql:param>` или в его атрибуте `value`:

```
<sql:update var="count" dataSource="${conn}">
    DELETE FROM profile WHERE id > ?
    <sql:param>100</sql:param>
</sql:update>

<sql:query var="rs" dataSource="${conn}">
    SELECT id, name FROM profile WHERE cats = ? AND color = ?
    <sql:param value="1" />
    <sql:param value="green" />
</sql:query>
```

К содержимому результирующего множества, возвращенного `<sql:query>`, можно обратиться несколькими способами. Если результирующему множеству сопоставлена переменная `rs`, то к строке *i* можно обратиться так: `rs.rows[`*i*`]` или `rs.rowsByIndex[`*i*`]`, где значения номеров строк начинаются с 0.

Первый вариант выводит строку, столбцы которой доступны по имени. Второй вариант выводит строку, столбцы которой доступны по своему номеру (начиная с 0). Например, если результирующее множество содержит столбцы `id` и `name`, то вы можете обратиться к значениям третьей строки, используя такие имена столбцов:

```
<c:out value="${rs.rows[2].id}" />
<c:out value="${rs.rows[2].name}" />
```

Теперь используем номера столбцов:

```
<c:out value="${rs.rowsByIndex[2][0]}" />
<c:out value="${rs.rowsByIndex[2][1]}" />
```

Можно применить `<c:forEach>` для организации цикла по строкам результирующего множества. Если вы хотите обращаться к столбцам по имени, используйте цикл по `rs.rows`:

```
<c:forEach var="row" items="${rs.rows}">
    id = <c:out value="${row.id}" />,
    name = <c:out value="${row.name}" />
    <br />
</c:forEach>
```

Или используйте `rs.rowsByIndex`, если хотите обращаться к столбцам по номеру:

```
<c:forEach var="row" items="${rs.rowsByIndex}">
    id = <c:out value="${row[0]}" />,
    name = <c:out value="${row[1]}" />
    <br />
</c:forEach>
```

Счетчик столбцов доступен как `rs.rowCount`:

```
Number of rows selected: <c:out value="${rs.rowCount}" />
```

Имена столбцов результирующего множества можно получить с помощью `rs.columnNames`:

```
<c:forEach var="name" items="${rs.columnNames}">
    <c:out value="${name}" />
    <br />
</c:forEach>
```

Создание сценария MySQL с помощью JSP и JSTL

В рецепте 16.2 было показано, как написать на Perl, PHP и Python сценарий, выводящий имена таблиц базы данных `cookbook`. Используя теги JSTL, мы можем создать страницу JSP, отображающую ту же информацию, так:

```
<%-- show_tables.jsp - Выполнить запрос SHOW TABLES и показать результат --%>

<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jstl/sql" prefix="sql" %>
```

```
<%@ include file="/WEB-INF/jstl-mcb-setup.inc" %>

<html>
<head>
<title>Tables in cookbook Database</title>
</head>
<body bgcolor="white">

<p>Tables in cookbook database:</p>

<sql:query var="rs" dataSource="${conn}">
    SHOW TABLES
</sql:query>

<c:forEach var="row" items="${rs.rowsByIndex}">
    <c:out value="${row[0]}" /><br />
</c:forEach>

</body>
</html>
```

Директивы `taglib` указывают, какие библиотеки тегов необходимы сценарию, а директива `include` включает код, задающий источник данных для доступа к базе данных `cookbook`. Оставшаяся часть сценария генерирует содержимое страницы.

Сценарий следует поместить в подкаталог *mcb* каталога *webapps* сервера Tomcat и вызывать его так:

http://tomcat.snake.net:8080/mcb/show_tables.jsp

Как и сценарий PHP, приведенный в рецепте 16.2, сценарий JSP явно не выводит заголовок `Content-Type:`. JSP автоматически формирует заголовок по умолчанию с типом содержимого `text/html`.

16.4. Кодирование специальных символов для Web

Задача

Некоторые символы являются специальными для HTML и должны кодироваться, если вы хотите отображать их буквально. Поскольку базы данных часто содержат такие символы, то сценарии, включающие результаты запросов в веб-страницы, должны кодировать эти результаты, чтобы не допустить некорректной интерпретации броузером.

Решение

Используйте методы, предлагаемые используемым API для выполнения кодирования HTML и URL.

Обсуждение

HTML – это язык разметки, он использует определенные символы как маркеры, имеющие специальное значение. Для того чтобы включить в страницу литеральные экземпляры таких символов, необходимо закодировать их, чтобы они не интерпретировались как специальные. Например, < нужно представить как < для того, чтобы броузер не интерпретировал его как начало тега. В зависимости от того, в каком контексте используется символ, возможны два способа кодирования. Один из них используется для общего текста HTML, а второй – для текста, входящего в состав URL в гиперссылке.

Сценарии вывода списка таблиц MySQL, приведенные в рецептах 16.2 и 16.3, представляют собой простые примеры формирования веб-страниц при помощи программ. Но (с одним исключением) у всех этих сценариев есть слабое место – они не занимаются кодированием специальных символов, встречающихся в данных, извлекаемых с сервера MySQL. (Исключением является версия на JSP; использованный в ней тег <c:out> автоматически выполняет кодирование, о чем мы еще вскорости поговорим.)

Я сознательно выбрал для отображения такую информацию, которая вряд ли может содержать специальные символы, так что сценарии будут работать корректно и без кодирования. Однако в общем случае не стоит надеяться на то, что результат запроса не будет содержать специальных символов, поэтому будьте готовы их кодировать. В противном случае сценарии могут формировать некорректно отображаемые HTML-страницы.

В этом разделе показано, как обрабатывать специальные символы. Начнем с общих принципов, а затем поговорим о том, как кодирование реализовано в каждом из API. Примеры для конкретных API показывают, как обрабатывать информацию, извлеченную из таблицы базы данных, но их можно адаптировать для работы с любым содержимым, включаемым в веб-страницу, вне зависимости от его происхождения.

Общие принципы кодирования

Один из видов кодирования применяется для символов, используемых при написании конструкций HTML, а второй – для текста, включаемого в URL. Важно понимать это отличие и применять соответствующий способ. Также имейте в виду, что кодирование текста для включения в веб-страницу не имеет ничего общего с кодированием специальных символов в значениях данных, включаемых в предложения SQL (см. рецепт 2.7).

Кодирование специальных символов в HTML

Разметка HTML использует символы < и > для начала и завершения тега, & – для начала наименований сущностей (таких как   для обозначения неразрывного пробела) и " для заключения в кавычки значений атрибутов в тегах (например, <p align="left">). Поэтому для буквального отображения таких символов необходимо закодировать их в сущности HTML, с тем чтобы броузеры и другие клиенты поняли ваше намерение. Для этого преобразуем

<, >, & и " в соответствующие сущности HTML < (меньше, чем), > (больше, чем), & (амперсанд) и " (кавычка).

Предположим, что вы хотите буквально вывести в веб-странице следующую строку:

```
Paragraphs begin and end with <p> & </p> tags.
```

Если отправить этот текст броузеру в том виде, в котором он записан сейчас, броузер не сможет интерпретировать его корректно. (Теги и будут восприняты как указатели абзаца, а & – вероятно, как начало сущности HTML.) Чтобы вывести строку именно так, как хотелось бы, необходимо преобразовать специальные символы в сущности <, > и &:

```
Paragraphs begin and end with &lt;p&gt; &amp; &lt;/p&gt; tags.
```

Этот же принцип кодирования текста применяется и внутри тегов. Например, значения атрибутов тегов HTML часто заключаются в двойные кавычки, поэтому необходимо выполнить HTML-кодирование значений атрибутов. Предположим, что вы хотите включить в форму поле ввода текста, при этом указав его начальное значение – Rich "Goose" Gossage. Нельзя просто написать это значение в теге:

```
<input type="text" name="player_name" value="Rich "Goose" Gossage" />
```

Получается, что атрибут value, заключенный в двойные кавычки, содержит внутренние двойные кавычки, в результате чего тег <input> оказывается некорректно сформированным. Необходимо закодировать двойные кавычки:

```
<input type="text" name="player_name" value="Rich &quot;Goose&quot; Gossage" />
```

Когда броузер получит такой текст, он преобразует " обратно в символы " и сможет правильно интерпретировать значение атрибута value.

Кодирование специальных символов в URL

URL гиперссылок, встречающиеся в HTML-страницах, имеют собственный синтаксис и собственный способ кодирования, который применяется к атрибутам нескольких тегов:

```
<a href="URL">
<img src="URL">
<form action="URL">
<frame src="URL">
```

Многие символы имеют специальное значение в URL (:, /, ?, =, & и ;). Следующий URL содержит некоторые из них:

http://apache.snake.net/myscript.php?id=428&name=Gandalf

Символы : и / разбивают URL на составляющие, символ ? указывает на наличие параметров, а символы & разделяют параметры, каждый из которых представлен парой *имя=значение*. (Символ ; не входит в данный URL, но часто используется вместо & для разделения параметров.) Если вы хотите вклю-

чить какие-то из этих символов в URL как литералы, необходимо закодировать их, чтобы броузер не интерпретировал их как специальные. Специальной обработки требуют и другие символы, такие как пробелы. Пробелы внутри URL не разрешены, так что, если вы хотите сослаться на страницу *my home page.html* с сайта *apache.snake.net*, то URL в гиперссылке нельзя записывать так:

```
<a href="http://apache.snake.net/my home page.html">My Home Page</a>
```

URL-кодирование специальных и зарезервированных символов выполняется путем преобразования каждого такого символа в его ASCII-код (две шестнадцатеричные цифры), перед которым нужно поставить знак %. Например, ASCII-код символа пробела равен 32 в десятичной системе счисления или 20 в шестнадцатеричной, так что предыдущую гиперссылку следует записать так:

```
<a href="http://apache.snake.net/my%20home%20page.html">My Home Page</a>
```

Иногда вы можете встретить пробелы, закодированные в URL как +. Это тоже разрешено.

Применение кодирования

Убедитесь, что вы корректно кодируете информацию именно в том контексте, в котором она используется. Предположим, вы хотите создать гиперссылку для запуска поиска элементов, соответствующих заданному термину, и хотите, чтобы сам термин отображался на странице как текст ссылки. Тогда термин будет как параметром в URL, так и текстом HTML, заключенным в теги <a> и . Если поиск совпадений ведется для термина «cats & dogs», то незакодированная ссылка будет выглядеть так:

```
<a href="/cgi-bin/myscript?term=cats & dogs">cats & dogs</a>
```

Но такая запись не корректна, поскольку & является специальным символом как для HTML, так и для URL, пробелы тоже относятся к специальным символам URL. Ссылку следует изменить так:

```
<a href="/cgi-bin/myscript?term=cats%20%26%20dogs">cats &amp; dogs</a>
```

В данном случае & закодирован в HTML как & для заголовка ссылки, а в URL он уже закодирован как %26, при этом пробелы в URL тоже закодированы как %20.

Конечно, не слишком приятно кодировать весь текст перед записью в веб-страницу, и иногда у вас имеется достаточно информации о значении, чтобы не кодировать его (см. примечание «Всегда ли необходимо кодировать вывод веб-страницы?»). Однако надежнее все же постоянно использовать кодирование. К счастью, большинство API предоставляют функции, которые сделают эту работу за вас. Поэтому вам необязательно знать все специальные символы для заданного контекста. Нужно лишь знать, какой вид кодирования надо использовать, и вызывать соответствующую функцию для получения желаемого результата.

Всегда ли необходимо кодировать вывод веб-страницы?

Если вы уверены в законности использования значения в определенном контексте в рамках веб-страницы, можно не кодировать его. Например, если вы получаете значение из столбца целого типа таблицы базы данных, не допускающего использования NULL, оно точно является целым. Для включения значения в веб-страницу не следует применять ни HTML-, ни URL-кодирование, так как цифры не являются специальными символами ни в тексте HTML, ни в URL. Но давайте представим себе другую ситуацию – вы запрашиваете целое значение через поле веб-формы. Ожидаете, что пользователь вводит целое число, но вводится недопустимое значение. Необходимо вывести страницу с сообщением об ошибке, содержащую введенное значение и поясняющую, что оно не является целым. Но если значение включает специальные символы, а вы их не закодируете, то значение будет отображено неправильно, что может запутать пользователя.

Кодирование специальных символов при помощи веб-API

Приведенные далее примеры показывают, как извлечь значения из MySQL и выполнить их HTML- и URL-кодирование для формирования гиперссылки. Все примеры читают таблицу phrase, где хранятся короткие фразы, используют ее содержимое для построения гиперссылок, указывающих на (гипотетический) сценарий, который осуществляет поиск вхождений этих фраз в какую-то другую таблицу. Таблица выглядит так:

```
mysql> SELECT phrase_val FROM phrase ORDER BY phrase_val;
+--------------------------+
| phrase_val               |
+--------------------------+
| are we "there" yet?      |
| cats & dogs              |
| rhinoceros               |
| the whole > sum of parts |
+--------------------------+
```

Наша цель – сформировать список гиперссылок, в котором каждая фраза будет использоваться как в заголовке гиперссылки (то есть потребуется HTML-кодирование), так и в качестве параметра сценария (потребуется URL-кодирование). В результате ссылки будут такими:

```
<a href="/cgi-bin/mysearch.pl?phrase=are%20we%20%22there%22%20yet%3F">
are we &amp;quot;there&amp;quot; yet?</a>
<a href="/cgi-bin/mysearch.pl?phrase=cats%20%26%20dogs">
cats &amp;amp; dogs</a>
<a href="/cgi-bin/mysearch.pl?phrase=rhinoceros">
```

```
rhinoceros</a>
<a href="/cgi-bin/mysearch.pl?phrase=the%20whole%20%3E%20sum%20of%20parts">
the whole &gt; sum of parts</a>
```

Ссылки, формируемые некоторыми API, будут немного отличаться от представленных, так как они кодируют пробелы как +, а не %20.

Perl

Модуль Perl CGI.pm предлагает два метода, escapeHTML() и escape(), которые занимаются HTML- и URL-кодированием. Есть три способа использования этих методов для кодирования строки $str:

- Вызвать escapeHTML() и escape() как методы класса CGI, используя префикс CGI::

```
use CGI;
printf "%s\n%s\n", CGI::escape ($str), CGI::escapeHTML ($str);
```

- Создать объект CGI и вызвать escapeHTML() и escape() как методы объекта:

```
use CGI;
my $cgi = new CGI;
printf "%s\n%s\n", $cgi->escape ($str), $cgi->escapeHTML ($str);
```

- Явно импортировать имена в пространство имен вашего сценария. Тогда не будет необходимости ни в объекте CGI, ни в префиксе CGI::, и можно будет вызывать методы как автономные функции. Следующий пример импортирует имена двух методов как дополнение к множеству стандартных имен:

```
use CGI qw(:standard escape escapeHTML);
printf "%s\n%s\n", escape ($str), escapeHTML ($str);
```

Я предпочитаю последний способ, так как он согласуется с функциональным интерфейсом CGI.pm, который мы используем для обращения к другим импортированным методам. Не забудьте включить имена методов кодирования в предложение use CGI любого сценария Perl, которому они требуются, иначе при запуске сценария возникнет ошибка типа «undefined subroutine».

Следующий код считывает содержимое таблицы phrase и формирует из него гиперссылки при помощи escapeHTML() и escape():

```
my $query = "SELECT phrase_val FROM phrase ORDER BY phrase_val";
my $sth = $dbh->prepare ($query);
$sth->execute ();
while (my ($phrase) = $sth->fetchrow_array ())
{
    # URL-кодирование значения phrase для использования в URL
    # HTML-кодирование значения phrase для использования в тексте ссылки
    my $url = "/cgi-bin/mysearch.pl?phrase=" . escape ($phrase);
    my $label = escapeHTML ($phrase);
    print a ({-href => $url}, $label) . br () . "\n";
}
```

PHP

В PHP HTML- и URL-кодированием занимаются функции `htmlspecialchars()` и `urlencode()`, которые используются так:

```
$query = "SELECT phrase_val FROM phrase ORDER BY phrase_val";
$result_id = mysql_query ($query, $conn_id);
if ($result_id)
{
    while (list ($phrase) = mysql_fetch_row ($result_id))
    {
        # URL-кодирование значения phrase для использования в URL
        # HTML-кодирование значения phrase для использования в тексте ссылки
        $url = "/mcb/mysearch.php?phrase=" . urlencode ($phrase);
        $label = htmlspecialchars ($phrase);
        printf ("<a href=\"%s\">%s</a><br />\n", $url, $label);
    }
    mysql_free_result ($result_id);
}
```

Python

В Python модули `cgi` и `urllib` содержат соответствующие методы кодирования: `cgi.escape()` выполняет HTML-кодирование, а `urllib.quote()` – URL-кодирование:

```
import cgi
import urllib

query = "SELECT phrase_val FROM phrase ORDER BY phrase_val"
cursor = conn.cursor ()
cursor.execute (query)
for (phrase,) in cursor.fetchall ():
    # URL-кодирование значения phrase для использования в URL
    # HTML-кодирование значения phrase для использования в тексте ссылки
    url = "/cgi-bin/mysearch.py?phrase=" + urllib.quote (phrase)
    label = cgi.escape (phrase, 1)
    print "<a href=\"%s\">%s</a><br />" % (url, label)
cursor.close ()
```

Первым аргументом `cgi.escape()` является строка, которую нужно закодировать для HTML. По умолчанию эта функция преобразует символы `<`, `>` и `&` в соответствующие им сущности HTML. Чтобы указать `cgi.escape()` на необходимость дополнительно преобразовывать двойные кавычки в `&quot;`, передайте второй аргумент, равный `1`, как показано в примере. Это особенно важно, если вы кодируете значения, которые будут помещаться в атрибуты тегов, заключенные в двойные кавычки.

Java

Тег JSTL `<c:out>` автоматически выполняет HTML-кодирование страниц JSP. (Строго говоря, он выполняет XML-кодирование, но здесь речь идет о наборе символов `<`, `>`, `&`, `"` и `'`, включающем все необходимые для HTML-кодирова-

ния символы.) Используя <c:out> для отображения текста на веб-странице, не стоит даже задумываться о преобразовании специальных символов в объекты HTML. Если по какой-то причине вы хотите отменить кодирование, вызовите <c:out> так:

```
<c:out value="значение для отображения" encodeXML="false" />
```

Для URL-кодирования параметров, включаемых в URL, используйте тег <c:url>. Задайте строку URL в атрибуте тега value, а все имена и значения параметров – в тегах <c:param> в теле тега <c:url>. Значение параметра можно указывать в атрибуте value тега <c:param> или в его теле:

```
<c:url var="urlStr" value="myscript.jsp">
    <c:param name="id" value ="47" />
    <c:param name="color">sky blue</c:param>
</c:url>
```

Значения параметров id и color URL-кодируются и добавляются в конец URL. Результат помещается в объект urlStr, который можно отобразить так:

```
<c:out value="${urlStr}" />
```

Тег <c:url> не кодирует такие специальные символы, как пробелы, в строке, записанной в атрибуте value. Их следует кодировать самостоятельно, так что, вероятно, разумнее просто избегать создания страниц с пробелами в названиях, чтобы не пришлось впоследствии ссылаться на такие страницы.

Записи таблицы phrase при помощи тегов <c:out> и <c:url> можно вывести так:

```
<sql:query var="rs" dataSource="${conn}">
    SELECT phrase_val FROM phrase ORDER BY phrase_val
</sql:query>

<c:forEach var="row" items="${rs.rows}">
    # URL-кодирование значения phrase для использования в URL
    # HTML-кодирование значения phrase для использования в тексте ссылки
    <c:url var="urlStr" value="/mcb/mysearch.jsp">
        <c:param name="phrase" value ="${row.phrase_val}" />
    </c:url>
    <a href="<c:out value="${urlStr}" />"><c:out value="${row.phrase_val}" />
    </a>
    <br />
</c:forEach>
```

17

Внедрение результатов запросов в веб-страницы

17.0. Введение

Если информация хранится в базе данных, есть множество простых способов использования ее в Web. Результаты запросов можно отобразить как неструктурированные абзацы или структурировать их в виде списков или таблиц, можно вывести простой текст, а можно создать гиперссылки. При форматировании результатов запроса можно воспользоваться метаданными, например, сформировать из них заголовки столбцов HTML-таблицы, отображающей результирующее множество. Такие задачи сочетают в себе обработку запросов с веб-программированием и требуют, главным образом, правильного кодирования специальных символов (таких как `&` или `<`) и добавления соответствующих HTML-тегов, формирующих требуемые элементы разметки.

В этой главе рассматривается несколько способов преобразования результатов запроса в пригодный для Сети формат. В ней также описаны способы хранения в базе данных двоичной информации, ее извлечения и передачи клиентам. (Самая простая и часто возникающая задача – это создание текстовых веб-страниц на основе содержимого базы данных, но MySQL может обрабатывать и запросы на получение двоичных данных, таких как изображения, звуки или PDF-документы.)

Приведенные здесь рецепты основаны на описанных в главе 16 способах создания веб-страниц с помощью сценариев и преобразования вывода в пригодный для отображения формат. Вы можете обратиться к этой главе для получения базовых сведений по данному вопросу.

Сценарии, положенные в основу примеров этой главы, находятся в дистрибутиве `recipes` в каталогах, имена которых соответствуют серверам, на которых выполняются сценарии. Для примеров на Perl, PHP и Python это каталог *apache*. Для примеров на Java (JSP) – *tomcat*; возможно, вы их уже установили, когда создавали среду исполнения для приложения `mcb` (см. рецепт 16.3). Исключение представляют только некоторые из использованных здесь утилит, расположенные в каталоге библиотечных файлов *lib*.

Заметьте, что не все языки представлены в каждом из разделов. Если вы обнаружите, что в каком-то из разделов отсутствует пример на интересующем вас языке, загляните в дистрибутив recipes; там может оказаться то, что вы ищете, даже если этого нет в тексте книги.

17.1. Представление результатов запроса в виде абзацев

Задача

Вы хотите представить результат запроса в виде простого текста.

Решение

Используйте для его отображения только теги абзаца HTML.

Обсуждение

Абзацы используются для вывода простого текста, не имеющего определенной структуры. В этом случае все, что вам нужно, – это извлечь отображаемый текст, конвертировать специальные символы в соответствующие HTML-конструкции и ограничить каждый абзац тегами и . Следующий пример показывает, как сформировать абзац для служебной страницы, включающей в себя текущее время и дату, версию сервера, имя клиента и имя текущей базы данных (если имеется). Эти значения получены таким запросом:

```
mysql> SELECT NOW(), VERSION(), USER(), DATABASE();
+---------------------+----------------+----------------+------------+
| NOW()               | VERSION()      | USER()         | DATABASE() |
+---------------------+----------------+----------------+------------+
| 2002-05-18 11:33:12 | 4.0.2-alpha-log | paul@localhost | cookbook   |
+---------------------+----------------+----------------+------------+
```

Для Perl в модуле CGI.pm имеется функция p(), заключающая в теги абзаца передаваемую ей строку. Функция p() не занимается HTML-кодированием своего аргумента, поэтому вы должны сделать это самостоятельно, вызвав функцию escapeHTML():

```
($now, $version, $user, $db) =
    $dbh->selectrow_array ("SELECT NOW(), VERSION(), USER(), DATABASE()");
$db = "NONE" unless defined ($db);
$para = <<EOF;
Local time on the MySQL server is $now.
The server version is $version.
The current user is $user.
The current database is $db.
EOF
print p (escapeHTML ($para));
```

В PHP можно вывести теги и в начале и конце перекодированного текста абзаца:

```
$query = "SELECT NOW(), VERSION(), USER(), DATABASE()";
$result_id = mysql_query ($query, $conn_id);
if ($result_id)
{
    list ($now, $version, $user, $db) = mysql_fetch_row ($result_id);
    mysql_free_result ($result_id);
    if (!isset ($db))
        $db = "NONE";
    $para = "Local time on the MySQL server is $now."
                . " The server version is $version."
                . " The current user is $user."
                . " The current database is $db.";
    print ("<p>" . htmlspecialchars ($para) . "</p>\n");
}
```

Или можно после получения результата запроса начать вывод абзаца в режиме HTML, переключаясь затем между режимами:

```
<p>
Local time on the MySQL server is
<?php print (htmlspecialchars ($now)); ?>.
The server version is
<?php print (htmlspecialchars ($version)); ?>.
The current user is
<?php print (htmlspecialchars ($user)); ?>.
The current database is
<?php print (htmlspecialchars ($db)); ?>.
</p>
```

Чтобы вывести абзац в Python, сделайте так:

```
cursor = conn.cursor ()
cursor.execute ("SELECT NOW(), VERSION(), USER(), DATABASE()")
row = cursor.fetchone ()
if row is not None:
    if row[3] is None:  # проверить имя БД
        row[3] = "NONE"
    para = ("Local time on the MySQL server is %s." +
            " The server version is %s." +
            " The current user is %s." +
            " The current database is %s.") % (row)
    print "<p>" + cgi.escape (para, 1) + "</p>"
cursor.close ()
```

В JSP эту задачу можно решить так:

```
<sql:query var="rs" dataSource="${conn}">
    SELECT NOW(), VERSION(), USER(), DATABASE()
</sql:query>
<c:set var="row" value="${rs.rowsByIndex[0]}" />
<c:set var="db" value="${row[3]}" />
<c:if test="${empty db}">
    <c:set var="db" value="NONE" />
```

```
</c:if>

<p>
Local time on the server is <c:out value="${row[0]}" />.
The server version is <c:out value="${row[1]}" />.
The current user is <c:out value="${row[2]}" />.
The current database is <c:out value="${db}" />.
</p>
```

Сценарий JSP использует `rowsByIndex`, так что столбцы результирующего множества доступны по индексу.

17.2. Представление результатов запроса в виде списков

Задача

Результат запроса содержит ряд значений, которые надо структурировать в виде списка.

Решение

Заключите элементы списка в теги HTML, соответствующие требуемому типу списка.

Обсуждение

Списки, занимающие по степени структурированности промежуточное положение между абзацами и таблицами, удобно использовать для представления набора отдельных элементов. В языке HTML есть несколько видов списков, а именно, неупорядоченные (маркированные) списки, упорядоченные (нумерованные) списки и списки определений. Возможно, вам понадобятся вложенные списки, образующиеся при форматировании элемента списка в виде списка.

Списки обычно состоят из открывающего и закрывающего тегов, окружающих набор элементов, каждый из которых помечен собственным тегом. Элементы списка естественным образом соответствуют строкам, полученным в запросе, поэтому создание HTML-списка из программы заключается в перекодировании результата запроса, добавлении к каждой строке нужных тегов и выводе открывающего и закрывающего тегов. Наиболее распространены два подхода к созданию списков. Если вы хотите выводить список по мере получения результатов запроса, делайте так:

1. Выведите открывающий тег списка.
2. Выбирайте и выводите каждую строку результирующего множества как элемент списка с соответствующими тегами.
3. Выведите закрывающий тег списка.

Другой подход состоит в формировании списка в памяти:

1. Сохраните элементы списка в массиве.
2. Передайте массив в функцию генерации списка, которая добавит необходимые теги, затем выведите результат.

Следующий пример иллюстрирует оба подхода.

Нумерованные списки

Нумерованный список (ordered list) состоит из элементов, расположенных в определенном порядке. Броузеры обычно отображают такие списки как набор элементов с порядковыми номерами:

```
1. Первый элемент
2. Второй элемент
3. Третий элемент
```

Вам не надо задавать номера элементов, так как броузер проставляет их автоматически. В HTML нумерованный список начинается и заканчивается тегами `<ol>` и `</ol>` соответственно и состоит из элементов, заключенных в теги `<li>` и `</li>`:

```
<ol>
  <li>Первый элемент</li>
  <li>Второй элемент</li>
  <li>Третий элемент</li>
</ol>
```

Предположим, у вас есть таблица `ingredient`, содержащая пронумерованные ингредиенты кулинарного рецепта:

```
+----+---------------------------------+
| id | item                            |
+----+---------------------------------+
|  1 | 3 cups flour                    |
|  2 | 1/2 cup raw ("unrefined") sugar |
|  3 | 3 eggs                          |
|  4 | pinch (< 1/16 teaspoon) salt    |
+----+---------------------------------+
```

В таблице есть столбец идентификаторов `id`, но для представления рецепта в виде нумерованного списка достаточно вывести только текстовые значения, так как броузер пронумерует их самостоятельно. Значения содержат специальные символы `"` и `<`, поэтому придется перекодировать их, прежде чем добавлять теги, превращающие значения в элементы списка. Результат должен выглядеть так:

```
<ol>
<li>3 cups flour</li>
<li>1/2 cup raw (&amp;quot;unrefined&amp;quot;) sugar</li>
<li>3 eggs</li>
<li>pinch (&amp;lt; 1/16 teaspoon) salt</li>
</ol>
```

Один из путей создания такого списка в программе – вывод в HTML по мере выборки строк результирующего множества. Вот как это можно сделать в JSP-странице с использованием тегов JSTL:

```
<sql:query var="rs" dataSource="${conn}">
    SELECT item FROM ingredient ORDER BY id
</sql:query>
<ol>
<c:forEach var="row" items="${rs.rows}">
    <li><c:out value="${row.item}" /></li>
</c:forEach>
</ol>
```

В PHP ту же операцию можно проделать так:

```
$query = "SELECT item FROM ingredient ORDER BY id";
$result_id = mysql_query ($query, $conn_id);
if (!$result_id)
    die (htmlspecialchars (mysql_error ($conn_id)));
print ("<ol>\n");
while (list ($item) = mysql_fetch_row ($result_id))
    print ("<li>" . htmlspecialchars ($item) . "</li>\n");
mysql_free_result ($result_id);
print ("</ol>\n");
```

Нет необходимости добавлять символы конца строки после завершающих тегов, как это сделано в данном примере; броузер не интересуется их наличием или отсутствием. Я предпочитаю добавлять их, так как генерируемый сценарием HTML-код легче читается, если не состоит из единственной строки, а это облегчает отладку.

Подход, использованный в предыдущих примерах, предполагает чередование выборки записей с генерацией вывода. Возможен и другой способ: раздельное выполнение операций, при котором данные сначала выбираются, затем выводятся. Запросы для разных списков могут различаться, но генерация самого списка выполняется, как правило, единообразно. Написав функцию генерации списка, вы сможете использовать ее с разными запросами. Две вещи, которые должна выполнять такая функция, – это перекодирование элементов (если это еще не сделано) и добавление требуемых HTML-тегов. В PHP, например, функцию `make_ordered_list()` можно реализовать следующим образом (она принимает в качестве аргумента массив элементов списка и возвращает список в виде строки):

```
function make_ordered_list ($items, $encode = TRUE)
{
    if (!is_array ($items))
        return ("make_ordered_list: items argument must be an array");
    $str = "<ol>\n";
    reset ($items);
    while (list ($k, $v) = each ($items))
    {
        if ($encode)
```

```
            $v = htmlspecialchars ($v);
        $str .= "<li>$v</li>\n";
    }
    $str .= "</ol>\n";
    return ($str);
}
```

Написав такую функцию, вы можете сделать выборку и вывести результат в HTML, например, так:

```
# получить элементы списка
$query = "SELECT item FROM ingredient ORDER BY id";
$result_id = mysql_query ($query, $conn_id);
if (!$result_id)
    die (htmlspecialchars (mysql_error ($conn_id)));
$items = array ();
while (list ($item) = mysql_fetch_row ($result_id))
    $items[] = $item;
mysql_free_result ($result_id);

# сформировать HTML-список
print (make_ordered_list ($items));
```

В Python подобная функция будет выглядеть так:

```
def make_ordered_list (items, encode = 1):
    if type (items) not in (types.ListType, types.TupleType):
        return ("make_ordered_list: items argument must be a list")
    list = "<ol>\n"
    for item in items:
        if item is None:    # обработать возможный пустой элемент
            item = ""
        # убедиться, что элемент – строка, и при необходимости перекодировать
        if type (item) is not types.StringType:
            item = `item`
        if encode:
            item = cgi.escape (item, 1)
        list = list + "<li>" + item + "</li>\n"
    list = list + "</ol>\n"
    return list
```

А использовать ее надо так:

```
# получить элементы списка
query = "SELECT item FROM ingredient ORDER BY id"
cursor = conn.cursor ()
cursor.execute (query)
items = []
for (item,) in cursor.fetchall ():
    items.append (item)
cursor.close ()

# сформировать HTML-список
print make_ordered_list (items)
```

В обоих вариантах функции make_ordered_list() на PHP и Python проверяется, является ли первый аргумент массивом. Если нет, возвращается строка с описанием ошибки. Благодаря возврату содержательной строки проблема сразу становится очевидной при взгляде на веб-страницу, сформированную функцией. Если хотите, можете возвращать другие признаки ошибки или генерировать исключение, или завершать выполнение сценария.

Второй аргумент функции make_ordered_list() определяет, надо ли перекодировать элементы списка. Проще всего позволить функции определять необходимость этого самостоятельно (поэтому по умолчанию и передается «истина»). Но если вы формируете список из элементов, которые уже содержат теги HTML, то не захотите, чтобы функция преобразовывала специальные символы, содержащиеся в этих тегах. Например, если вы создаете список гиперссылок, то каждый элемент списка будет содержать тег <a>. Чтобы исключить перекодирование тега в <a>, передайте функции make_ordered_list() вторым параметром значение FALSE (для PHP) или 0 (для Python).

Конечно, если в вашем API есть функции генерации HTML-структур, вам не придется писать их самостоятельно. Это относится к модулю Perl CGI.pm: вызовите функцию li(), которая создает элемент, добавляя к нему открывающий и закрывающий теги, сохраните элементы в массиве, затем передайте этот массив в функцию ol(), которая добавит открывающий и закрывающий теги списка:

```
my $query = "SELECT item FROM ingredient ORDER BY id";
my $sth = $dbh->prepare ($query);
$sth->execute ();
my @items = ();
while (my $ref = $sth->fetchrow_arrayref ())
{
    # обработать возможный элемент NULL (undef)
    my $item = (defined ($ref->[0]) ? escapeHTML ($ref->[0]) : "");
    push (@items, li ($item));
}
print ol (@items);
```

Преобразовывать неопределенные значения (undef или NULL) в пустые строки необходимо для того, чтобы исключить генерацию предупреждений о неинициализированных значениях, когда Perl запускается с опцией *-w*. (Таблица ingredient не содержит значений NULL, но этот способ полезен при работе с таблицами, в которых такие значения встречаются.)

В предыдущем примере выборка записей и генерация HTML выполняются попеременно. Чтобы разделить операции выборки и формирования кода, поместите элементы в массив. Затем передайте массив по ссылке функции li(), а ее результат – функции ol():

```
# получить элементы списка
my $query = "SELECT item FROM ingredient ORDER BY id";
my $item_ref = $dbh->selectcol_arrayref ($query);

# сгенерировать список HTML, обрабатывая возможные элементы NULL (undef)
```

```
$item_ref = [ map { defined ($_) ? escapeHTML ($_) : "" } @{$item_ref} ];
print ol (li ($item_ref));
```

Имейте в виду следующие два свойства функции `li()`:

- Она не выполняет никакого HTML-кодирования; вы должны сделать это самостоятельно.
- Она может обработать как отдельное значение, так и массив. Но если вы передаете ей массив, вы должны передать его по ссылке. Тогда функция добавит теги `<li>` и `</li>` к каждому элементу массива, затем склеит их и вернет результирующую строку. Если вы передадите массив не по ссылке, а по значению, функция сначала объединит элементы, а затем заключит результат в единственную пару тегов, что, скорее всего, не соответствует вашим ожиданиям. Аналогичное поведение свойственно и некоторым другим функциям CGI.pm, которые могут оперировать как одним, так и несколькими значениями. Например, функция обработки табличных данных `td()` добавляет единственную пару тегов `<td>` и `</td>`, если получает скаляр или список. Если же ей передать ссылку на список, она добавит теги к каждому из его элементов.

Маркированные списки

Маркированный список (unordered list) аналогичен нумерованному, за исключением того, что броузер отображает его элементы, помечая их одним и тем же символом (маркером):

- Первый элемент
- Второй элемент
- Третий элемент

Маркированные списки являются неупорядоченными в том смысле, что символ маркера не содержит информацию о порядке элементов. Разумеется, вы можете вывести элементы в нужном вам порядке. Код HTML для маркированного списка аналогичен коду нумерованного списка, только вместо тегов `<ol>` и `</ol>` используются `<ul>` и `</ul>`:

```
<ul>
  <li>Первый элемент</li>
  <li>Второй элемент</li>
  <li>Третий элемент</li>
</ul>
```

В тех API, где вы непосредственно выводите теги, используйте ту же процедуру, что и для нумерованных списков, но вместо `<ol>` и `</ol>` выводите `<ul>` и `</ul>`. Вот пример на JSP:

```
<sql:query var="rs" dataSource="${conn}">
    SELECT item FROM ingredient ORDER BY id
</sql:query>
<ul>
<c:forEach var="row" items="${rs.rows}">
    <li><c:out value="${row.item}" /></li>
```

Объединять или разделять выборку записей и генерирование HTML?

Если при написании сценария для вас главное – срочность, то вы, вероятно, захотите как можно скорее запустить его и сразу начать выводить HTML-код по мере выборки строк. Однако отделение выборки данных от формирования результата имеет свои преимущества. Наиболее очевидным из них является то, что однажды написанную функцию для генерации HTML можно повторно использовать в других сценариях. (Еще лучше, если в вашем API уже имеется такая функция.) Но есть и другие преимущества:

- Функции, формирующие HTML-структуры, могут обрабатывать данные, полученные не только из баз данных, но и из других источников.
- При раздельном подходе не требуется генерировать вывод немедленно. Можно создать всю страницу в памяти и вывести ее уже готовой. Это особенно полезно для страниц, состоящих из нескольких частей, так как позволяет формировать компоненты в том порядке, который вам наиболее удобен.
- Отделение выборки данных от формирования результата позволяет достичь большей гибкости при выводе. Если вы решили сформировать маркированный список вместо нумерованного, просто вызовите другую функцию, процедура выборки данных при этом не изменится. Это преимущество сохраняется, даже если вы решили изменить формат вывода (например, заменить HTML на XML или WML). В таком случае вам придется лишь изменить функцию вывода, в части получения данных изменений не будет.
- Получив элементы списка заранее, вы свободны в выборе способа его отображения. Хотя мы и не дошли пока до обсуждения веб-форм, в них часто используются собственные типы списков. С этой точки зрения получение всего списка до начала формирования HTML-кода позволяет выбрать тип отображения в зависимости от его размера. Например, если список невелик, можно использовать переключатель (radio button), а для больших списков – всплывающие меню (pop-up menu) или списки с прокруткой (scrolling list).

```
</c:forEach>
</ul>
```

В Perl маркированный список создается функцией `ul()` модуля CGI.pm:

```
# получить элементы списка
my $query = "SELECT item FROM ingredient ORDER BY id";
my $item_ref = $dbh->selectcol_arrayref ($query);

# сгенерировать список HTML, обрабатывая возможные элементы NULL (undef)
```

```
$item_ref = [ map { defined ($_) ? escapeHTML ($_) : "" } @{$item_ref} ];
print ul (li ($item_ref));
```

Если вы пишете собственную функцию для маркированного списка, то можете адаптировать функцию формирования нумерованного списка. Например, обе версии функции `make_ordered_list()` для PHP и Python легко превращаются в `make_unordered_list()`, так как отличие заключается только в открывающем и закрывающем тегах.

Списки определений

Элементы списка определений (definition list) состоят из двух частей – термина и его определения. Здесь слова «термин» и «определение» можно толковать сколь угодно широко: вы можете размещать здесь любую информацию. Например, в приведенной ниже таблице `doremi` каждой ноте музыкальной гаммы (`note`) сопоставлена мнемоническая фраза (`mnemonic`), облегчающая запоминание (которая в действительности определением не является):

```
+----+------+---------------------------+
| id | note | mnemonic                  |
+----+------+---------------------------+
|  1 | do   | A deer, a female deer     |
|  2 | re   | A drop of golden sun      |
|  3 | mi   | A name I call myself      |
|  4 | fa   | A long, long way to run   |
|  5 | so   | A needle pulling thread   |
|  6 | la   | A note to follow so       |
|  7 | ti   | I drink with jam and bread |
+----+------+---------------------------+
```

Тем не менее столбцы `note` и `mnemonic` могут быть представлены списком определений:

```
do
    A deer, a female deer
re
    A drop of golden sun
mi
    A name I call myself
fa
    A long, long way to run
so
    A needle pulling thread
la
    A note to follow so
ti
    I drink with jam and bread
```

Код HTML для списка определений начинается и заканчивается тегами `<dl>` и `</dl>` соответственно. Каждый элемент содержит термин, заключенный в теги `<dt>` и `</dt>`, и определение, помещенное в теги `<dd>` и `</dd>`:

```
<dl>
  <dt>do</dt> <dd>A deer, a female deer</dd>
  <dt>re</dt> <dd>A drop of golden sun</dd>
  <dt>mi</dt> <dd>A name I call myself</dd>
  <dt>fa</dt> <dd>A long, long way to run</dd>
  <dt>so</dt> <dd>A needle pulling thread</dd>
  <dt>la</dt> <dd>A note to follow so</dd>
  <dt>ti</dt> <dd>I drink with jam and bread</dd>
</dl>
```

В JSP-страницах список определений можно сгенерировать так:

```
<sql:query var="rs" dataSource="${conn}">
    SELECT note, mnemonic FROM doremi ORDER BY note
</sql:query>
<dl>
<c:forEach var="row" items="${rs.rows}">
    <dt><c:out value="${row.note}" /></dt>
    <dd><c:out value="${row.mnemonic}" /></dd>
</c:forEach>
</dl>
```

В PHP можно сделать так:

```
$query = "SELECT item FROM ingredient ORDER BY id";
$result_id = mysql_query ($query, $conn_id);
if (!$result_id)
    die (htmlspecialchars (mysql_error ($conn_id)));
print ("<dl>\n");
while (list ($note, $mnemonic) = mysql_fetch_row ($result_id))
{
    print ("<dt>" . htmlspecialchars ($note) . "</dt>\n");
    print ("<dd>" . htmlspecialchars ($mnemonic) . "</dt>\n");
}
mysql_free_result ($result_id);
print ("</dl>\n");
```

Или можно написать функцию, принимающую массивы терминов и определений и возвращающую список в виде строки:

```
function make_definition_list ($terms, $definitions, $encode = TRUE)
{
    if (!is_array ($terms))
        return ("make_definition_list: terms argument must be an array");
    if (!is_array ($definitions))
        return ("make_definition_list: definitions argument must be an array");
    if (count ($terms) != count ($definitions))
        return ("make_definition_list: term and definition list size mismatch");
    $str = "<dl>\n";
    reset ($terms);
    reset ($definitions);
    while (list ($dtk, $dtv) = each ($terms))
    {
```

```
        list ($ddk, $ddv) = each ($definitions);
        if ($encode)
        {
            $dtv = htmlspecialchars ($dtv);
            $ddv = htmlspecialchars ($ddv);
        }
        $str .= "<dt>$dtv</dt>\n<dd>$ddv</dd>\n";
    }
    $str .= "</dl>\n";
    return ($str);
}
```

Приведем пример использования функции `make_definition_list()`**:**

```
# получить элементы списка
$query = "SELECT note, mnemonic FROM doremi ORDER BY id";
$result_id = mysql_query ($query, $conn_id);
if (!$result_id)
    die (htmlspecialchars (mysql_error ($conn_id)));
$terms = array ();
$defs = array ();
while (list ($note, $mnemonic) = mysql_fetch_row ($result_id))
{
    $terms[] = $note;
    $defs[] = $mnemonic;
}
mysql_free_result ($result_id);

# сгенерировать список HTML
print (make_definition_list ($terms, $defs));
```

В Perl создайте термины и определения вызовом функций `dt()` **и** `dd()`**, сохраните их в массиве и передайте его функции** `dl()`**:**

```
my $query = "SELECT note, mnemonic FROM doremi ORDER BY id";
my $sth = $dbh->prepare ($query);
$sth->execute ();
my @items = ();
while (my ($note, $mnemonic) = $sth->fetchrow_array ())
{
    # обработать возможные значения NULL (undef)
    $note = (defined ($note) ? escapeHTML ($note) : "");
    $mnemonic = (defined ($mnemonic) ? escapeHTML ($mnemonic) : "");
    push (@items, dt ($note));
    push (@items, dd ($mnemonic));
}
print dl (@items);
```

Ниже приведен чуть более сложный пример. Каждый термин – это имя базы данных, а в соответствующем определении указано количество таблиц в ней. Значения получены выполнением запроса `SHOW TABLES` **к каждой базе данных и подсчетом строк результата:**

```
# получить список имен баз данных
my $db_ref = $dbh->selectcol_arrayref ("SHOW DATABASES");
my @items = ();
foreach my $db_name (@{$db_ref})
{
    # получить список имен таблиц базы данных; отключить RaiseError для этого запроса,
    # чтобы сценарий не завершался, если пользователь не имеет доступа к текущей БД
    $dbh->{RaiseError} = 0;
    my $tbl_ref = $dbh->selectcol_arrayref ("SHOW TABLES FROM $db_name");
    $dbh->{RaiseError} = 1;
    my $tbl_count = (defined ($tbl_ref)         # ошибка?
                    ? @{$tbl_ref} . " tables"   # нет, считаем таблицы
                    : "cannot access");         # да, сообщаем о проблеме
    push (@items, dt (escapeHTML ($db_name)));
    push (@items, dd (escapeHTML ($tbl_count)));
}
print dl (@items);
```

Обратите внимание на то, что надо не допустить выхода из сценария при возникновении ошибки выполнения команды `SHOW TABLES` по причине запуска сценария пользователем, не имеющим доступа к какой-либо базе данных.

Немаркированные списки

В списках этого типа, обычно даже не рассматриваемых при обсуждении списков, вообще нет маркеров. Это просто набор элементов, каждый из которых расположен на отдельной строке. Создать немаркированный список очень просто: после каждой строки добавьте тег перевода строки. Вот пример на JSP:

```
<c:forEach var="row" items="${rs.rows}">
    <c:out value="${row.item}" /><br />
</c:forEach>
```

Если элементы уже помещены в массив, просто пройдите их в цикле. Например, в Perl, если элементы находятся в массиве `@items`, список генерируется так:

```
foreach my $item (@items)
{
    # обработать возможные значения NULL (undef)
    $item = (defined ($item) ? escapeHTML ($item) : "");
    print $item . br ();
}
```

Вложенные списки

Некоторые приложения выводят информацию, которая лучше всего воспринимается в виде списка списков. В следующем примере названия штатов, сгруппированные по первой букве, отображаются в списке определений. В каждом элементе списка термином служит начальная буква, а определение

представляет собой маркированный список названий штатов, начинающихся с этой буквы:

```
A
  • Alabama
  • Alaska
  • Arizona
  • Arkansas
C
  • California
  • Colorado
  • Connecticut
D
  • Delaware
...
```

Вот один из способов создания (на Perl) такого списка:

```
# получить список начальных букв
my $ltr_ref = $dbh->selectcol_arrayref (
            "SELECT DISTINCT UPPER(LEFT(name,1)) AS letter
            FROM states ORDER BY letter");
my @items = ();
# получить список штатов для каждой буквы
foreach my $ltr (@{$ltr_ref})
{
    my $item_ref = $dbh->selectcol_arrayref (
                  "SELECT name FROM states WHERE LEFT(name,1) = ?
                  ORDER BY name", undef, $ltr);
    $item_ref = [ map { escapeHTML ($_) } @{$item_ref} ];
    # сформировать маркированный список названий штатов
    my $item_list = ul (li ($item_ref));
    # для каждого элемента списка определений начальная буква - термин,
    # а список штатов - определение
    push (@items, dt ($ltr));
    push (@items, dd ($item_list));
}
print dl (@items);
```

В приведенном примере первый запрос возвращает список различных букв, затем другой запрос по каждой букве выбирает подходящие названия штатов. Можно получить всю информацию в одном запросе, затем пройти по результирующему множеству и для каждой новой буквы создать свой элемент списка:

```
my $sth = $dbh->prepare ("SELECT name FROM states ORDER BY name");
$sth->execute ();
my @items = ();
my @names = ();
my $cur_ltr = "";
while (my ($name) = $sth->fetchrow_array ())
{
```

```
    my $ltr = uc (substr ($name, 0, 1));    # начальная буква названия
    if ($cur_ltr ne $ltr)                   # начать с новой буквы?
    {
        if (@names)         # есть ли уже имена для предыдущих букв?
        {
            # для каждого элемента списка определений начальная буква – термин,
            # а список штатов - определение

            push (@items, dt ($cur_ltr));
            push (@items, dd (ul (li (\@names))));
        }
        @names = ();
        $cur_ltr = $ltr;
    }
    push (@names, escapeHTML ($name));
}
if (@names)                     # остались названия для последней буквы?
{
    push (@items, dt ($cur_ltr));
    push (@items, dd (ul (li (\@names))));
}
print dl (@items);
```

В третьем варианте используется единственный запрос, но разделены этапы подготовки данных и генерации HTML-кода:

```
# получить названия штатов и сопоставить их списку начальных букв
my $sth = $dbh->prepare ("SELECT name FROM states ORDER BY name");
$sth->execute ();
my %ltr = ();
while (my ($name) = $sth->fetchrow_array ())
{
    my $ltr = uc (substr ($name, 0, 1));    # первая буква названия
    # инициализировать список букв пустым массивом, если
    # это первый штат, затем добавлять штаты в массив
    $ltr{$ltr} = [] unless exists ($ltr{$ltr});
    push (@{$ltr{$ltr}}, $name);
}

# теперь создать результирующий список
my @items = ();
foreach my $ltr (sort (keys (%ltr)))
{
    # кодировать список названий штатов для данной буквы,
    # создать маркированный список
    my $ul_str = ul (li ([ map { escapeHTML ($_) } @{$ltr{$ltr}} ]));
    push (@items, dt ($ltr), dd ($ul_str));
}
print dl (@items);
```

17.3. Представление результатов запроса в виде таблиц

Задача

Вы хотите представить результат запроса в виде HTML-таблицы.

Решение

Поместите каждую строку результата в строку таблицы. Если вам нужна начальная строка заголовков столбцов, то подставьте свою либо воспользуйтесь метаданными, содержащими имена столбцов результирующего множества.

Обсуждение

Таблицы HTML удобны для представления сильно структурированных данных. Одна из причин их популярности при отображении результатов запросов заключается в естественном соответствии таблиц HTML таблицам баз данных или результирующим множествам. Кроме того, заголовки столбцов таблицы можно получить из метаданных запроса. Основу структуры таблиц HTML составляют следующие элементы:

- Таблица начинается и заканчивается тегами `<table>` и `</table>` и содержит набор строк.
- Каждая строка начинается и заканчивается тегами `<tr>` и `</tr>` и содержит набор ячеек.
- Ячейки ограничены тегами `<td>` и `</td>`. Для ячеек заголовка используются теги `<th>` и `</th>`. (Обычно броузеры выделяют ячейки заголовка жирным шрифтом или иным способом.)
- Теги могут иметь атрибуты. Например, для отображения границ ячеек используется атрибут `border="1"` тега `<table>`. Для выравнивания содержимого ячеек вправо – атрибут `align="right"` тега `<td>`.

Помните, что в каждом элементе таблицы всегда необходимо ставить закрывающий тег. Это правило полезно вообще для всех элементов HTML, но для таблиц оно особенно важно. При пропуске закрывающего тега некоторые броузеры подставляют его автоматически, но другие могут зависнуть или аварийно завершиться.

Предположим, вы хотите показать содержимое вашей коллекции компакт-дисков:

```
mysql> SELECT year, artist, title FROM cd ORDER BY artist, year;
+------+-----------------+----------------------+
| year | artist          | title                |
+------+-----------------+----------------------+
| 1992 | Charlie Peacock | Lie Down in the Grass |
| 1996 | Charlie Peacock | strangelanguage      |
| 1990 | Iona            | Iona                 |
| 1993 | Iona            | Beyond These Shores  |
```

```
| 1990 | Michael Gettel  | Return                |
| 1989 | Richard Souther | Cross Currents        |
| 1987 | The 77s         | The 77s               |
| 1982 | Undercover      | Undercover            |
+------+-----------------+-----------------------+
```

Чтобы отобразить результат этого запроса в HTML-таблице с рамкой, вам надо получить примерно такой код:

```
<table border="1">
  <tr>
    <th>Year</th>
    <th>Artist</th>
    <th>Title</th>
  </tr>
  <tr>
    <td>1992</td>
    <td>Charlie Peacock</td>
    <td>Lie Down in the Grass</td>
  </tr>
  <tr>
    <td>1996</td>
    <td>Charlie Peacock</td>
    <td>strangelanguage</td>
  </tr>
  ... еще строки ...
  <tr>
    <td>1982</td>
    <td>Undercover</td>
    <td>Undercover</td>
  </tr>
</table>
```

В процессе преобразования результатов запроса в таблицу HTML каждое значение результирующего множества помещается в теги ячейки, каждая строка – в теги строки, а все множество – в теги таблицы. Страница JSP, формирующая таблицу HTML из результатов запроса к таблице `cd`, может выглядеть так:

```
<table border="1">
    <tr>
        <th>Year</th>
        <th>Artist</th>
        <th>Title</th>
    </tr>
<sql:query var="rs" dataSource="${conn}">
    SELECT year, artist, title FROM cd ORDER BY artist, year
</sql:query>
<c:forEach var="row" items="${rs.rows}">
    <tr>
        <td><c:out value="${row.year}" /></td>
        <td><c:out value="${row.artist}" /></td>
        <td><c:out value="${row.title}" /></td>
```

```
    </tr>
</c:forEach>

</table>
```

В сценариях на Perl таблица, строка, ячейка данных и ячейка заголовка формируются функциями модуля CGI.pm `table()`, `tr()`, `td()` и `th()`. Но функцию `tr()`, генерирующую строку таблицы, следует вызывать как `Tr()`, чтобы избежать конфликта со встроенной функцией транслитерации `tr`.[1] Отображение содержимого таблицы `cd` в виде HTML-таблицы выполняется так:

```
my $sth = $dbh->prepare (
            "SELECT year, artist, title FROM cd ORDER BY artist, year"
        );
$sth->execute ();
my @rows = ();
push (@rows, Tr (th ("Year"), th ("Artist"), th ("Title")));
while (my ($year, $artist, $title) = $sth->fetchrow_array ())
{
    push (@rows, Tr (
                    td (escapeHTML ($year)),
                    td (escapeHTML ($artist)),
                    td (escapeHTML ($title))
            ));
}
print table ({-border => "1"}, @rows);
```

Иногда лучше воспринимается таблица со строками разных цветов, особенно если у ячеек нет рамки. Чтобы достичь такого эффекта, добавьте атрибут `bgcolor` в каждый из тегов `<th>` и `<td>` и укажите для каждой строки свой цвет. Это легко сделать с помощью переменной, переключающейся между двумя значениями. В следующем примере переменная `$bgcolor` попеременно принимает значения `silver` и `white`:

```
my $sth = $dbh->prepare (
            "SELECT year, artist, title FROM cd ORDER BY artist, year"
        );
$sth->execute ();
my $bgcolor = "silver";
my @rows = ();
push (@rows, Tr (
                th ({-bgcolor => $bgcolor}, "Year"),
                th ({-bgcolor => $bgcolor}, "Artist"),
                th ({-bgcolor => $bgcolor}, "Title")
            ));
while (my ($year, $artist, $title) = $sth->fetchrow_array ())
{
    $bgcolor = ($bgcolor eq "silver" ? "white" : "silver");
```

[1] Если вы используете объектно-ориентированный интерфейс CGI.pm, то неоднозначность не возникает. В этом случае вы вызываете `tr()` как метод объекта `CGI`, поэтому нет необходимости обращаться к нему как `Tr()`.

```
    push (@rows, Tr (
                    td ({-bgcolor => $bgcolor}, escapeHTML ($year)),
                    td ({-bgcolor => $bgcolor}, escapeHTML ($artist)),
                    td ({-bgcolor => $bgcolor}, escapeHTML ($title))
            ));
}
print table ({-border => "1"}, @rows);
```

В предыдущих примерах генерации таблиц количество столбцов и их заголовки задавались в явном виде. Приложив немного усилий, можно написать более общую функцию, которая принимает дескриптор базы данных и произвольный запрос, выполняет этот запрос и возвращает результат в виде HTML-таблицы. Такая функция может автоматически формировать заголовки столбцов по метаданным запроса; если имена столбцов таблицы не годятся в качестве заголовков, то в запросе можно назначить им псевдонимы:

```
my $tbl_str = make_table_from_query (
                $dbh,
                "SELECT
                    year AS Year, artist AS Artist, title AS Title
                FROM cd
                ORDER BY artist, year"
            );
print $tbl_str;
```

Эта функция может принимать любые запросы, формирующие результирующее множество. С ее помощью можно, например, сформировать таблицу HTML, отображающую все метаданные столбцов какой-либо таблицы базы данных:

```
my $tbl_str = make_table_from_query ($dbh, "SHOW COLUMNS FROM profile");
print $tbl_str;
```

Как же выглядит функция `make_table_from_query()`? Приведем ее реализацию на Perl:

```
sub make_table_from_query
{
# дескриптор БД, строка запроса, параметры для замены заполнителей (если есть)
my ($dbh, $query, @param) = @_;

    my $sth = $dbh->prepare ($query);
    $sth->execute (@param);
    my @rows = ();
    # поместить имена столбцов в ячейки заголовка
    push (@rows, Tr (th ([ map { escapeHTML ($_) } @{$sth->{NAME}} ])));
    # выбрать все строки данных
    while (my $row_ref = $sth->fetchrow_arrayref ())
    {
        # кодировать значения ячеек, избегая предупреждений для
        # неопределенных значений и вставляя   в пустые ячейки
        my @val = map {
                    defined ($_) && $_ !~ /^\s*$/ ? escapeHTML ($_) : " "
                } @{$row_ref};
```

```
        my $row_str;
        for (my $i = 0; $i < @val; $i++)
        {
            # числовые столбцы выровнять вправо
            if ($sth->{mysql_is_num}->[$i])
            {
                $row_str .= td ({-align => "right"}, $val[$i]);
            }
            else
            {
                $row_str .= td ($val[$i]);
            }
        }
        push (@rows, Tr ($row_str));
    }
    return (table ({-border => "1"}, @rows));
}
```

Функция `make_table_from_query()` выполняет некоторые дополнительные действия для выравнивания вправо числовых столбцов (так они лучше смотрятся). Она также позволяет передать значения для связывания с заполнителями в запросе. Укажите их сразу после строки запроса:

```
my $tbl_str = make_table_from_query (
                $dbh,
                "SELECT
                    year AS Year, artist AS Artist, title AS Title
                FROM cd
                WHERE year < ?
                ORDER BY artist, year",
                1990
            );
print $tbl_str;
```

Фокус для пустых ячеек: неразрывный пробел

В таблицах HTML, имеющих рамку вокруг ячеек, иногда возникает проблема с отображением: если ячейки пусты или содержат только пробельные символы, то многие броузеры не выводят рамку для такой ячейки, из-за чего портится внешний вид таблицы. Чтобы избежать этого, функция `make_table_from_query()` помещает в пустые ячейки неразрывный пробел (` `), благодаря чему рамка вокруг таких ячеек отображается правильно.

При использовании программно генерируемых таблиц следует иметь в виду, что броузеры не могут разместить таблицу в окне до тех пор, пока не получат ее целиком. Если вы работаете с большим результирующим множеством, то его отображение может занять много времени. Решением этой проблемы может быть распределение данных по нескольким таблицам в пределах одной страницы (чтобы броузер мог отображать их по мере получения) или по не-

скольким страницам. Если вы размещаете несколько таблиц на странице, то, вероятно, придется добавить атрибуты ширины в теги ячеек заголовка и данных. В противном случае каждая таблица подстроится под фактический размер значений в своих столбцах. Если этот размер меняется от таблицы к таблице, страница будет выглядеть неаккуратно.

См. также

Способ отображения таблицы, при котором содержимое таблицы сортируется по щелчку на заголовке столбца, рассматривается в рецепте 18.11.

17.4. Представление результатов запроса в виде гиперссылок

Задача

Вы хотите генерировать гиперссылки на основе содержимого базы данных.

Решение

Добавьте к содержимому теги ссылок.

Обсуждение

Примеры предыдущих разделов формируют статический текст, но на основе содержимого базы данных можно создавать и гиперссылки. Если вы храните в таблице адреса веб-сайтов или адреса электронной почты, то без труда можно преобразовать их в активные ссылки на веб-страницах. Нужно лишь соответствующим образом перекодировать информацию и добавить теги HTML.

Предположим, что у вас есть таблица с названиями компаний и их веб-сайтами, например, таблица `book_vendor` со списком книготорговцев и издателей:

```
mysql> SELECT * FROM book_vendor ORDER BY name;
+----------------------+------------------+
| name                 | website          |
+----------------------+------------------+
| Barnes & Noble       | www.bn.com       |
| Bookpool             | www.bookpool.com |
| Borders              | www.borders.com  |
| Fatbrain             | www.fatbrain.com |
| O'Reilly & Associates | www.oreilly.com  |
+----------------------+------------------+
```

Содержимое таблицы легко преобразуется в гиперссылку. Чтобы превратить строку в гиперссылку, добавьте спецификатор протокола `http://` к значению `website`, используйте результат как атрибут `href` тега `<a>` и используйте значение `name` в теле тега в качестве текста ссылки. Например, строку для Barnes & Noble можно записать так:

```
<a href="http://www.bn.com">Barnes &amp;amp; Noble</a>
```

Код JSP, выводящий маркированный (неупорядоченный) список гиперссылок для содержимого таблицы, выглядит так:

```
<sql:query var="rs" dataSource="${conn}">
    SELECT name, website FROM book_vendor ORDER BY name
</sql:query>

<ul>
<c:forEach var="row" items="${rs.rows}">
    <li>
        <a href="http://<c:out value="${row.website}" />">
            <c:out value="${row.name}" /></a>
    </li>
</c:forEach>
</ul>
```

При отображении на веб-странице название каждого продавца из списка становится активной ссылкой, выбрав которую вы попадете на его веб-сайт. В Python аналогичная операция выполняется так:

```
query = "SELECT name, website FROM book_vendor ORDER BY name"
cursor = conn.cursor ()
cursor.execute (query)
items = []
for (name, website) in cursor.fetchall ():
    items.append ("<a href=\"http://%s\">%s</a>" \
                % (urllib.quote (website), cgi.escape (name, 1)))
cursor.close ()

# вывести элементы, но не кодировать, так как они уже закодированы
print make_unordered_list (items, 0)
```

Сценарии на Perl, использующие CGI.pm, выводят гиперссылки при помощи вызова функции `a()`:

```
a ({-href => "url-значение"}, "наименование ссылки")
```

Эту функцию можно использовать для вывода списка ссылок на продавцов:

```
my $sth = $dbh->prepare (
                "SELECT name, website
                FROM book_vendor
                ORDER BY name");
$sth->execute ();
my @items = ();
while (my ($name, $website) = $sth->fetchrow_array ())
{
    push (@items, a ({-href => "http://$web-site"}, escapeHTML ($name)));
}
print ul (li (\@items));
```

Еще одной часто встречающейся задачей веб-программирования является формирование ссылок для адресов электронной почты. Предположим, что

у вас есть таблица newsstaff, в которой перечислены отдел (department), фамилия (name) и (если известен) адрес электронной почты (email) для корреспондентов и обозревателей службы новостей телекомпании WRRR:

```
mysql> SELECT * FROM newsstaff;
+------------------+----------------+------------------------+
| department       | name           | email                  |
+------------------+----------------+------------------------+
| Sports           | Mike Byerson   | mbyerson@wrrr-news.com |
| Sports           | Becky Winthrop | bwinthrop@wrrr-news.com |
| Weather          | Bill Hagburg   | bhagburg@wrrr-news.com |
| Local News       | Frieda Stevens | NULL                   |
| Local Government | Rex Conex      | rconex@wrrr-news.com   |
| Current Events   | Xavier Ng      | xng@wrrr-news.com      |
| Consumer News    | Trish White    | twhite@wrrr-news.com   |
+------------------+----------------+------------------------+
```

По таблице можно сформировать каталог он-лайн со ссылками на электронные адреса всех служащих, чтобы посетители сайта могли без проблем отправить письмо любому сотруднику. Например, строка для спортивного обозревателя Mike Byerson (электронный адрес mbyerson@wrrr-news.com) будет содержать такую конструкцию:

```
Sports: <a href="mailto:mbyerson@wrrr-news.com">Mike Byerson</a>
```

Содержимое таблицы без труда преобразуется в такой каталог. Для начала давайте поместим код генерирования ссылки на электронный адрес во вспомогательную функцию, поскольку такая операция наверняка пригодится нам и во многих других сценариях. В Perl эта функция может выглядеть так:

```
sub make_email_link
{
my ($name, $addr) = @_;

    # вернуть имя в виде статического текста, если адрес пустой или undef
    return (escapeHTML ($name)) if !defined ($addr) || $addr eq "";
    # в противном случае вернуть гиперссылку
    return (a ({-href => "mailto:$addr"}, escapeHTML ($name)));
}
```

Функция обрабатывает возможность отсутствия у кого-то электронного адреса, выводя в таком случае имя как обычный текст. Используем функцию в цикле, который будет извлекать имена и адреса и выводить перед ссылкой на каждый адрес название отдела сотрудника:

```
my $sth = $dbh->prepare (
            "SELECT department, name, email
            FROM newsstaff
            ORDER BY department, name");
$sth->execute ();
my @items = ();
while (my ($dept, $name, $email) = $sth->fetchrow_array ())
{
```

```
    push (@items, escapeHTML ($dept) . ": " . make_email_link ($name, $email));
}
print ul (li (\@items));
```

Такие же функции генерирования ссылок на адреса электронной почты для PHP и Python будут выглядеть так:

```
function make_email_link ($name, $addr)
{
    # вернуть имя в виде статического текста, если адрес не задан или пустой
    if (!isset ($addr) || $addr == "")
        return (htmlspecialchars ($name));
    # в противном случае вернуть гиперссылку
    return (sprintf ("<a href=\"mailto:%s\">%s</a>",
                     $addr, htmlspecialchars ($name)));
}
```

```
def make_email_link (name, addr):
    # вернуть имя в виде статического текста, если адрес None или пустой
    if addr is None or addr == "":
        return (cgi.escape (name, 1))
    # в противном случае вернуть гиперссылку
    return ("<a href=\"mailto:%s\">%s</a>" % (addr, cgi.escape (name, 1)))
```

Для страницы JSP можно вывести список `newsstaff` **так:**

```
<sql:query var="rs" dataSource="${conn}">
    SELECT department, name, email
    FROM newsstaff
    ORDER BY department, name
</sql:query>
```

```
<ul>
<c:forEach var="row" items="${rs.rows}">
    <li>
        <c:out value="${row.department}" />:
        <c:set var="name" value="${row.name}" />
        <c:set var="email" value="${row.email}" />
        <c:choose>
            <%-- null or empty value test --%>
            <c:when test="${empty email}">
                <c:out value="${name}" />
            </c:when>
            <c:otherwise>
                <a href="mailto:<c:out value="${email}" />">
                    <c:out value="${name}" /></a>
            </c:otherwise>
        </c:choose>
    </li>
</c:forEach>
</ul>
```

17.5. Создание навигационного индекса

Задача

Список элементов на веб-странице очень длинный. Вы хотите упростить передвижение по такой странице.

Решение

Создайте навигационный индекс со ссылками на различные разделы списка.

Обсуждение

Отображать списки на веб-страницах не представляет труда (см. рецепт 17.2). Но если список содержит очень много элементов, то содержащая его страница может стать очень длинной. В таких случаях может быть разумно разбить список на разделы и составить навигационный индекс в форме гиперссылок, которые позволяют пользователям быстро перемещаться от раздела к разделу, не прокручивая страницу вручную. Например, если вы извлекаете записи из таблицы и выводите их сгруппированными в разделы, то можете включить в страницу индекс, который обеспечивает переход напрямую к любому разделу. Этот же принцип применяется и для многостраничных отображений: на каждой странице выводится навигационный индекс, и пользователи легко могут попасть с одной страницы на любую другую.

В этом разделе приведено два примера, иллюстрирующих эти приемы. Оба они используют таблицу `kjv` из рецепта 4.11. Примеры реализуют два вида отображения для стихов Книги Есфирь (The Book of Esther), хранящихся в таблице `kjv`:

- Одностраничный вывод, представляющий все стихи всех глав Книги Есфирь. Список разбит на 10 разделов (по одному на главу) и содержит навигационный индекс, ссылки которого указывают на начало каждого раздела.
- Многостраничный вывод, содержащий страницу для стихов каждой из глав Книги Есфирь, а также главную страницу, которая предлагает пользователю выбрать нужную главу. Каждая из страниц выводит список глав в виде гиперссылок на страницы, отображающие стихи соответствующей главы. Благодаря таким ссылкам можно легко переходить с одной страницы на другую.

Создание одностраничного навигационного индекса

Рассмотрим пример, выводящий все стихи Книги Есфирь на одной странице, при этом стихи разбиты на разделы по главам. Для того чтобы вывести страницу так, чтобы каждый раздел содержал маркер навигации, поместите анкер `<a name>` перед началом стихов каждой новой главы:

```
<a name="1">Chapter 1</a>
    ... список стихов главы 1...
```

```
<a name="2">Chapter 2</a>
    ... список стихов главы 2...
<a name="3">Chapter 3</a>
    ... список стихов главы 3...
...
```

Будет сгенерирован список, включающий набор маркеров с названиями 1, 2, 3 и т. д. Для создания навигационного индекса сформируйте набор гиперссылок, каждая из которых указывает на один из маркеров name:

```
<a href="#1>Chapter 1</a>
<a href="#2>Chapter 2</a>
<a href="#3>Chapter 3</a>
...
```

Символ # в каждом атрибуте href означает, что ссылка указывает на элемент той же страницы. Например, href="#3" указывает на анкер с атрибутом name="3".

Реализовать такой навигационный индекс можно двумя способами:

- Извлечь записи (records) стихов в память и определить по ним, какие элементы должны присутствовать в навигационном индексе. Затем вывести и индекс, и список стихов.
- Заранее определить все необходимые анкеры и построить индекс. Список номеров глав можно получить так:

  ```
  SELECT DISTINCT cnum FROM kjv WHERE bname = 'Esther' ORDER BY cnum;
  ```

  Можно использовать результат запроса для построения навигационного индекса, а потом выбрать стихи из глав для создания разделов страницы, на которые указывают элементы индекса.

Рассмотрим сценарий *esther1.pl*, реализующий первый подход к решению задачи. Это переделанный пример из раздела о вложенных списках (см. рецепт 17.2):

```
#! /usr/bin/perl -w
# esther1.pl - выводит Книгу Есфирь на одной странице с навигационным индексом

use strict;
use lib qw(/usr/local/apache/lib/perl);
use CGI qw(:standard escape escapeHTML);
use Cookbook;

my $title = "The Book of Esther";

my $page = header ()
           . start_html (-title => $title, -bgcolor => "white")
           . h3 ($title);

my $dbh = Cookbook::connect ();

# Извлекаем стихи из Книги Есфирь и сопоставляем каждому из них
# список стихов главы, в которую он входит.
```

```
my $sth = $dbh->prepare (
            "SELECT cnum, vnum, vtext FROM kjv
            WHERE bname = 'Esther'
            ORDER BY cnum, vnum");
$sth->execute ();
my %verses = ();
while (my ($cnum, $vnum, $vtext) = $sth->fetchrow_array ())
{
    # инициализируем списком стихов главы пустой массив, если это ее первый стих,
    # затем добавляем в массив номер и текст стиха.
    $verses{$cnum} = [] unless exists ($verses{$cnum});
    push (@{$verses{$cnum}}, p (escapeHTML ("$vnum. $vtext")));
}

# Определяем номера всех глав и используем их для построения навигационного
# индекса. Это будут ссылки в виде <a href="#num>Chapter num</a>, где num -
# это номер главы, а '#' означает ссылку внутри одной страницы.
# Ни URL-, ни HTML-кодирование не применяется (отображаемый текст в этом
# не нуждается). Проверьте, что номера глав упорядочиваются в числовом порядке
# (используйте { a <=> b }). Разделяем ссылки неразрывными пробелами.

my $nav_index;
foreach my $cnum (sort { $a <=> $b } keys (%verses))
{
    $nav_index .= " " if $nav_index;
    $nav_index .= a ({-href => "#$cnum"}, "Chapter $cnum");
}

# Теперь выводим список стихов каждой главы. Перед каждым разделом вводится
# заголовок с номером главы и элементом навигационного индекса.

foreach my $cnum (sort { $a <=> $b } keys (%verses))
{
    # добавляем анкер <a name> для этого раздела
    $page .= p (a ({-name => $cnum}, font ({-size => "+2"}, "Chapter $cnum"))
                . br ()
                . $nav_index);
    $page .= join ("", @{$verses{$cnum}});  # добавить список стихов главы
}

$dbh->disconnect ();

$page .= end_html ();

print $page;

exit (0);
```

Создание многостраничного навигационного индекса

Теперь рассмотрим сценарий на Perl *esther2.pl*, который может генерировать любую из нескольких страниц, отображающих стихи Книги Есфирь из таблицы `kjv`. Первая страница отображает список глав книги, а также инструкции по выбору главы. Каждая глава списка – это гиперссылка, которая по-

вторно вызывает сценарий для вывода списка стихов соответствующей главы. Так как сценарий отвечает за формирование нескольких страниц, он должен уметь определять, какую из страниц выводить при каждом запуске. Для этого сценарий исследует параметр `chapter` собственного URL, указывающий номер главы для вывода. Если параметр `chapter` отсутствует или его значение не целое, то сценарий выводит первую страницу.

URL первой страницы будет таким:

http://apache.snake.net/cgi-bin/esther2.pl

Ссылки на страницы отдельных глав выглядят так (`cnum` — это номер главы):

http://apache.snake.net/cgi-bin/esther2.pl?chapter=cnum

Сценарий *esther2.pl* использует функцию CGI.pm `param()` для получения значения параметра `chapter`:

```
my $cnum = param ("chapter");
if (!defined ($cnum) || $cnum !~ /^\d+$/)
{
    # Номер главы отсутствует или неправильный
}
else
{
    # Номер главы указан правильно
}
```

Если параметр `chapter` отсутствует в URL, то `$cnum` будет содержать `undef`. В противном случае `$cnum` устанавливается в значение параметра, после того как проверено, что оно целочисленное (это делается для выявления случаев ввода некорректных значений с целью остановки работы сценария).

Вот полный текст сценария *esther2.pl*:

```
#! /usr/bin/perl -w
# esther2.pl - выводит Книгу Есфирь на нескольких страницах,
# по одной на главу, с навигационным индексом

use strict;
use lib qw(/usr/local/apache/lib/perl);
use CGI qw(:standard escape escapeHTML);
use Cookbook;

my $title = "The Book of Esther";

my $page = header () . start_html (-title => $title, -bgcolor => "white");

my ($left_panel, $right_panel);

my $dbh = Cookbook::connect ();

my $cnum = param ("chapter");
if (!defined ($cnum) || $cnum !~ /^\d+$/)
{
    # Отсутствующая или неправильно сформированная страница; вывести
    # главную страницу, левая панель которой содержит список всех глав
```

```
    # в виде гиперссылок, а правая - инструкции по выбору главы.
    $left_panel = get_chapter_list ($dbh, 0);
    $right_panel = p (strong ("The Book of Esther"))
                   . p ("Select a chapter from the list at left.");
}
else
{
    # Указан номер главы; вывести в левой панели список глав, все главы,
    # кроме текущей, представлены гиперссылками, а текущая выделена жирным шрифтом.
    # Правая панель отображает список стихов текущей главы.
    $left_panel = get_chapter_list ($dbh, $cnum);
    $right_panel = p (strong ("The Book of Esther"))
                   . get_verses ($dbh, $cnum);
}

$dbh->disconnect ();

# Оформить страницу как однострочную таблицу из трех ячеек
# (средняя ячейка - разделитель)

$page .= table (Tr (
                   td ({-valign => "top", -width => "15%"}, $left_panel),
                   td ({-valign => "top", -width => "5%"}, " "),
                   td ({-valign => "top", -width => "75%"}, $right_panel)
               ));

$page .= end_html ();

print $page;

exit (0);

# ----------------------------------------------------------------------

# Создать навигационный индекс в виде списка ссылок на страницы каждой главы
# Книги Есфирь. Заголовки имеют форму "Chapter n"; номера глав встроены
# в ссылки в виде параметров chapter=num

# $dbh - это дескриптор базы данных, $cnum - номер главы, информация которой
# выводится в настоящий момент. Заголовок из списка глав, соответствующий
# этому номеру, выводится как статический текст, а все остальные -
# как гиперссылки на другие страницы глав. Введите 0, чтобы все записи
# стали гиперссылками (ни одна из глав не имеет номера 0).

# Кодирование не производится, так как номера глав - это числа,
# и кодирование не требуется.

sub get_chapter_list
{
my ($dbh, $cnum) = @_;

    my $nav_index;
    my $ref = $dbh->selectcol_arrayref (
                "SELECT DISTINCT cnum FROM kjv
                WHERE bname = 'Esther' ORDER BY cnum"
            );
```

```
    foreach my $cur_cnum (@{$ref})
    {
        my $link = url () . "?chapter=$cur_cnum";
        my $label = "Chapter $cur_cnum";
        $nav_index .= br () if $nav_index;       # разделить записи, используя <br>
        # использовать статический, выделенный жирным шрифтом текст
        # для текущей главы, для остальных - гиперссылку
        $nav_index .= ($cur_cnum == $cnum
                       ? strong ($label)
                       : a ({-href => $link}, $label));
    }
    return ($nav_index);
}

# Получить список стихов указанной главы. Если отсутствует, то введен
# неправильный номер, но будем разумно обрабатывать такую ситуацию.

sub get_verses
{
my ($dbh, $cnum) = @_;

    my $ref = $dbh->selectall_arrayref (
                  "SELECT vnum, vtext FROM kjv
                  WHERE bname = 'Esther' AND cnum = ?",
                  undef, $cnum);
    my $verses = "";
    foreach my $row_ref (@{$ref})
    {
        $verses .= p (escapeHTML ("$row_ref->[0]. $row_ref->[1]"));
    }
    return ($verses eq ""           # нет стихов?
            ? p ("No verses in chapter $cnum were found.")
            : p ("Chapter $cnum:") . $verses);
}
```

См. также

Сценарий *esther2.pl* исследует свое окружение с помощью функции `param()`. В рецепте 18.5 подробно рассматривается обработка параметров веб-сценариев.

17.6. Хранение изображений и других двоичных данных

Задача

Вы хотите хранить в MySQL изображения.

Решение

Это несложно, если вы правильно закодируете данные.

Обсуждение

На веб-сайтах можно отображать не только текст, но и различные виды двоичных данных, такие как изображения, звуки, PDF-документы и так далее. Самым распространенным видом двоичных данных являются изображения, а базы данных подходят для их хранения, поэтому периодически возникает вопрос «Как хранить изображения в MySQL?». Многие сразу ответят: «Не делайте этого!», и на то есть ряд причин, некоторые из них приведены в примечании «Следует ли хранить изображения в базе данных?». Поскольку уметь работать с двоичными данными полезно, в этом разделе показано, как хранить изображения в MySQL. Но поскольку это не всегда удачная идея, рассказано и о том, как хранить изображения в файловой системе.

В разделе говорится об изображениях, но эти же принципы можно применять и к любым другим двоичным данным, таким как PDF-файлы или сжатый текст. В действительности они вообще применяются к любым видам данных, включая текст. Об изображениях часто думают как о чем-то особенном, но на самом деле это не так.

Одна из причин, по которой хранение изображений представляется чем-то более сложным, чем хранение текстовых строк или чисел, заключается в том, что значение изображения сложно ввести вручную. Например, легко использовать *mysql* для выполнения предложения `INSERT`, вставляющего число типа `3.48` или строку `Je voudrais une bicyclette rouge`, но изображения содержат двоичные данные, и оперировать их значениями нелегко. Придется найти какой-то другой выход. Ваши варианты:

- Использовать функцию `LOAD_FILE()`.
- Написать функцию, которая читает файл изображения и формирует для вас соответствующее предложение `INSERT`.

Сохранение изображений с помощью функции LOAD_FILE()

Функция `LOAD_FILE()` принимает аргумент, указывающий, какой файл следует прочитать и сохранить в базе данных. Например, хранящееся в */tmp/myimage.png* изображение можно сохранить в таблице так:

```
mysql> INSERT INTO mytbl (image_data) VALUES(LOAD_FILE('/tmp/myimage.png'));
```

Для сохранения изображений в MySQL при помощи `LOAD_FILE()` требуется выполнение нескольких условий:

- Файл изображения должен располагаться на хосте сервера MySQL.
- Файл должен быть доступен серверу для чтения.
- Вы должны обладать привилегией `FILE`.

То есть функция `LOAD_FILE()` доступна не всем пользователям MySQL.

Сохранение изображений с помощью сценария

Если использование функции LOAD_FILE() недоступно или нежелательно, вы можете написать небольшую программу, которая будет загружать ваши изображения. Программа будет или читать содержимое файла изображения и создавать запись, содержащую данные изображения, или создавать запись, указывающую, где в файловой системе расположен файл изображения. Если вы решите хранить изображение в MySQL, включите данные изображения в предложение создания записи, как вы сделали бы это для данных любого другого типа. То есть можно использовать заполнитель и связать с ним значение данных или закодировать данные и поместить их непосредственно в строку запроса.

Сценарий *store_image.pl* запускается из командной строки и сохраняет файл изображения для дальнейшего использования. Этот сценарий не держится ни одной из сторон в дебатах о хранении изображений в базе данных или файловой системе. Вместо этого он реализует *оба* способа. Конечно, это удваивает объем пространства для хранения, так что, адаптируя сценарий для собственного использования, вы можете оставить в нем только то, что соответствует предпочитаемому вами способу хранения изображений. Необходимые изменения будут рассмотрены в конце раздела.

Сценарий *store_image.pl* работает с таблицей image, в которой есть столбцы идентификатора, имени файла и MIME-типа изображения, а также столбец собственно данных изображения:

```
CREATE TABLE image
(
    id      INT UNSIGNED NOT NULL AUTO_INCREMENT,   # идентификатор изображения
    name    VARCHAR(30) NOT NULL,                   # имя файла изображения
    type    VARCHAR(20) NOT NULL,                   # MIME-тип изображения
    data    MEDIUMBLOB NOT NULL,                    # данные изображения
    PRIMARY KEY (id),                               # id и name уникальны
    UNIQUE (name)
);
```

Столбец name указывает имя файла изображения в каталоге, в котором изображения хранятся в файловой системе. Столбец data имеет тип MEDIUMBLOB и подходит для изображений размером не более 16 Мбайт. Если вы предполагаете хранить более объемистые изображения, используйте столбец типа LONGBLOB.

Можно использовать столбец name для хранения полных путей к изображениям в базе данных, но если поместить их все в один каталог, то можно хранить имена относительно этого каталога, тогда значения name будут занимать меньше места. Сценарий *store_image.pl* именно так и поступает, но, естественно, ему нужно откуда-то узнать путь к каталогу для хранения изображений. Для этого применяется переменная $image_dir. Вы проверяете значение этой переменной и (при необходимости) изменяете его перед запуском сценария. Значение по умолчанию соответствует тому каталогу, где

я предпочитаю хранить изображения, но вы можете изменить его согласно собственным пожеланиям. Убедитесь в том, что указанный каталог существует (или создайте его) перед запуском сценария. Вам также необходимо проверить и, возможно, изменить путь к каталогу изображений в сценарии *display_image.pl*, который будет рассмотрен далее в этой главе.

Следует ли хранить изображения в базе данных?

Выбирая способ хранения изображений, вы должны пойти на компромисс. И у хранения в файловой системе, и у хранения в базе данных есть свои «за» и «против»:

- Хранение изображений в таблице базы данных приводит к раздуванию таблицы. Если изображений будет много, то вероятнее всего вы используете все место, которое операционная система отводит для таблицы. С другой стороны, если хранить изображения в файловой системе, то замедлится просмотр каталогов. Чтобы избежать этого, можно создать иерархическую структуру или использовать файловую систему, обладающую высокой производительностью для больших каталогов (например, файловая система Reiser).
- При помощи базы данных можно локализовать хранение изображений, используемых несколькими веб-серверами на различных хостах. Изображения, хранящиеся в файловой системе, должны храниться локально на хосте веб-сервера. Если работа ведется с несколькими хостами, то необходимо продублировать набор изображений в файловой системе каждого хоста. Если вы храните изображения в MySQL, необходима всего одна копия изображения, любой веб-сервер может получить изображения с одного и того же сервера базы данных.
- При хранении изображений в файловой системе они по сути представляют собой внешний ключ. Для обработки изображений необходимо выполнить две операции: одну над базой данных, вторую – над файловой системой. То есть если вам требуется использовать транзакции, их сложнее реализовать – операций не просто две, они к тому же производятся в разных областях. Хранить изображения в базе данных проще, так как добавление, обновление и удаление изображения требует выполнения всего одной операции. Исчезает необходимость обеспечивать синхронность таблицы и файловой системы.
- Может оказаться, что предоставлять изображения через Сеть из файловой системы получается быстрее, чем из базы данных, так как веб-сервер сам открывает файл, читает его и отправляет клиенту. Хранящиеся в базе данных изображения должны читаться и записываться дважды. Сначала MySQL читает изображение из базы данных и записывает в ваш веб-сценарий, а затем сценарий читает изображение и отсылает его в клиентское приложение.

- На изображения, хранящиеся в файловой системе, можно непосредственно ссылаться в веб-странице при помощи тега <img>, который указывает на файлы изображений. Изображения, хранящиеся в MySQL, должны обслуживаться сценарием, который извлекает изображение и отправляет его клиенту. Однако даже если изображения хранятся в файловой системе и доступны веб-серверу, вы все равно можете захотеть отправить их при помощи сценария. Это удобно, если вы хотите подсчитывать количество отправлений каждого изображения (например, если вы платите клиентам за количество нажатых баннеров) или если вы хотите выбирать изображение в момент запроса (например, при случайном выборе баннера для отображения).
- Если вы храните изображения в базе данных, то должны использовать такой тип столбца, как BLOB. Это тип переменной длины, поэтому сама таблица будет иметь строки переменной длины. Операции со строками фиксированной длины часто выполняются быстрее, поэтому можно получить некоторый выигрыш производительности при просмотре таблиц за счет хранения изображений в файловой системе и использования типов фиксированной длины для столбцов таблицы изображений.

Сценарий *store_image.pl* выглядит так:

```
#! /usr/bin/perl -w
# store_image.pl - читает файл изображения, сохраняет в таблице
# и в файловой системе. (Обычно изображение сохраняется только в каком-то
# одном месте, сценарий же приводит два способа.)

use strict;
use lib qw(/usr/local/apache/lib/perl);
use Fcntl;      # для O_RDONLY, O_WRONLY, O_CREAT
use FileHandle;
use Cookbook;

# Значения по умолчанию для каталога хранения изображения и разделителя пути
# (ИЗМЕНИТЬ ЗДЕСЬ, ЕСЛИ ТРЕБУЕТСЯ)
my $image_dir = "/usr/local/apache/htdocs/mcb/images";
my $path_sep = "/";

# Установка каталога и разделителя пути для Windows/DOS
if ($^O =~ /^MSWin/i || $^O =~ /^dos/)
{
    $image_dir = "D:\\apache\\htdocs\\mcb\\images";
    $path_sep = "\\";
}

-d $image_dir or die "$0: image directory ($image_dir)\ndoes not exist\n";

# Вывод справочного сообщения в случае, если сценарий вызван некорректно
```

```
(@ARGV == 2 || @ARGV == 3) or die <<EOF;
Usage: $0 image_file mime_type [image_name]

image_file = name of the image file to store
mime_time = the image MIME type (e.g., image/jpeg or image/png)
image_name = alternate name to give the image

image_name is optional; if not specified, the default is the
image file basename.
EOF

my $file_name = shift (@ARGV);  # имя файла изображения
my $mime_type = shift (@ARGV);  # MIME-тип изображения
my $image_name = shift (@ARGV); # название изображения (необязательно)

# если не указано название изображения, использовать исходное имя
# (разрешить разделитель / или \)
($image_name = $file_name) =~ s|.*[/\\]|| unless defined $image_name;

my $fh = new FileHandle;
my ($size, $data);

sysopen ($fh, $file_name, O_RDONLY)
    or die "Cannot read $file_name: $!\n";
binmode ($fh);      # для работы в двоичном режиме
$size = (stat ($fh))[7];
sysread ($fh, $data, $size) == $size
    or die "Failed to read entire file $file_name: $!\n";
$fh->close ();

# Сохранить файл изображения в файловой системе в каталоге $image_dir.
# (Перезаписать файл, если он уже существует.)

my $image_path = $image_dir . $path_sep . $image_name;

sysopen ($fh, $image_path, O_WRONLY|O_CREAT)
    or die "Cannot open $image_path: $!\n";
binmode ($fh);      # для работы в двоичном режиме
syswrite ($fh, $data, $size) == $size
    or die "Failed to write entire image file $image_path: $!\n";
$fh->close ();

# Сохранить изображение в таблице базы данных. (Использовать REPLACE
# для удаления старых изображений с таким же именем.)

my $dbh = Cookbook::connect ();
$dbh->do ("REPLACE INTO image (name,type,data) VALUES(?,?,?)",
            undef,
            $image_name, $mime_type, $data);
$dbh->disconnect ();

exit (0);
```

Если вы вызываете сценарий без аргументов, он выводит короткое справочное сообщение. В противном случае он требует ввести два аргумента: имя файла изображения и MIME-тип изображения. По умолчанию имя исходного

файла (последняя составляющая пути) используется и как название изображения, хранящегося в базе данных и в каталоге. Если вы хотите использовать другое имя, укажите его как третий необязательный аргумент.

Сценарий достаточно прост. Он выполняет следующие действия:

1. Проверяет, правильное ли количество аргументов указано, и устанавливает по ним некоторые переменные.
2. Убеждается в наличии каталога изображений. В случае его отсутствия сценарий не может продолжать работу.
3. Открывает и читает файл изображения.
4. Сохраняет изображение в файле в соответствующем каталоге.
5. Сохраняет запись, содержащую идентификационную информацию и данные изображения, в таблице `image`.

Сценарий *store_image.pl* использует предложение `REPLACE`, а не `INSERT`, так что для замены старого изображения новым просто загрузите его с тем же именем. В запросе не указано значение для столбца `id`; это столбец `AUTO_INCREMENT`, поэтому MySQL автоматически присваивает ему уникальный номер последовательности. (Имейте в виду, что если вы заменяете изображение, загружая новое с тем же именем, то предложение `REPLACE` сгенерирует новое значение `id`. Если вы хотите сохранить старое значение, необходимо сначала выполнить `SELECT`, чтобы узнать, существует ли указанное имя, затем изменить `REPLACE`, указав имеющееся значение `id`, если запись была обнаружена, и `NULL` – в противном случае.)

Предложение `REPLACE`, сохраняющее информацию об изображении в MySQL, выглядит вполне привычно:

```
$dbh->do ("REPLACE INTO image (name,type,data) VALUES(?,?,?)",
          undef,
          $image_name, $mime_type, $data);
```

Если вы посмотрите на предложение, пытаясь обнаружить какой-то специальный указатель того, как следует обрабатывать двоичные данные, то ничего не увидите. Переменная `$data`, содержащая изображение, никаким специальным образом не обрабатывается. Запрос одинаково ссылается на все столбцы, используя символы-заполнители `?`, с передачей значений в конце вызова `do()`. Есть и другой способ получения такого же результата – явно кодировать значения столбцов, а затем вставлять их прямо в строку запроса:

```
$image_name = $dbh->quote ($image_name);
$mime_type = $dbh->quote ($mime_type);
$data = $dbh->quote ($data);
$dbh->do ("REPLACE INTO image (name,type,data)
          VALUES($image_name,$mime_type,$data)");
```

Многие склонны неоправданно усложнять работу с изображениями. Если вы корректно обрабатываете изображения в запросе, используя заполнители или кодируя их, проблем не возникнет. В противном случае возникнут ошибки. Все просто. Нет никаких отличий от обработки других видов данных,

в том числе и текста. В конце концов, если вставить в запрос элемент текста, содержащий кавычки или другие специальные символы, и не экранировать их, то запрос выскажет все, что он о вас думает. Так что использование заполнителей или кодирование присуще не только изображениям, а необходимо для любых данных. Повторяйте за мной: «Я буду всегда использовать заполнители или кодировать значения столбцов. Всегда, всегда, *всегда*!» (Сказав это, уточню, что если у вас есть достоверные данные о том, что указанное значение, например, является целочисленным, то вы можете нарушить данное обещание. Однако применение такого правила никогда не повредит.)

Для того чтобы опробовать сценарий, перейдите в каталог *apache/images* дистрибутива `recipes`. Каталог содержит сценарий *store_image.pl*, а в подкаталоге *flags* содержится несколько тестовых изображений (это национальные флаги различных стран). Чтобы загрузить одно из таких изображений, запустите сценарий в UNIX так:

```
% ./store_image.pl flags/iceland.jpg image/jpeg
```

Тем, кто работает в Windows, следует поступить так:

```
C:\> store_image.pl flags\iceland.jpg image/jpeg
```

Сценарий *store_image.pl* занимается сохранением изображения, а в следующем разделе будет рассказано об извлечении изображений и отправке их через Сеть. А как насчет удаления изображений? Напишите утилиту, удаляющую ненужные изображения, самостоятельно. Если вы храните изображения в файловой системе, помните о необходимости удаления как записи базы данных, так и файла, на который эта запись указывает.

Сценарий *store_image.pl* сохраняет каждое изображение и в базе данных, и в файловой системе для демонстрации обоих методов, что, естественно, делает его неэффективным. Ранее я уже говорил, что если вы используете этот сценарий как основу для собственных приложений, вам следует изменить его так, чтобы он реализовывал только один из методов хранения изображений, но не оба сразу. Изменения должны быть такими:

- Чтобы сценарий сохранял изображения только в MySQL, не определяйте каталог изображений и удалите код, проверяющий наличие каталога, а затем записывающий в него файлы изображений.
- Чтобы сценарий сохранял изображения только в файловой системе, удалите столбец данных из таблицы `image` и измените предложение `REPLACE` так, чтобы оно не ссылалось на этот столбец.

Такие же изменения нужно внести в сценарий обработки изображений *display_image.pl* из рецепта 17.7.

См. также

В рецепте 17.7 показано, как извлекать изображения для вывода через Сеть. Получение изображений с веб-страницы для сохранения в MySQL обсуждается в рецепте 18.8.

17.7. Извлечение изображений и других двоичных данных

Задача

Отлично, мы научились сохранять изображения и другие двоичные данные в базе данных (см. рецепт 17.6). Но как теперь получить их обратно?

Решение

Нужно всего лишь выполнить предложение SELECT. Естественно, ответить на вопрос о том, что делать с этой информацией после извлечения, несколько сложнее.

Обсуждение

В рецепте 17.6 упоминалось, что было бы тяжело вручную задавать буквальное значение, представляющее собой изображение. Однако ввод запроса, извлекающего изображения, не создает никаких трудностей:

```
mysql> SELECT * FROM image WHERE id = 1;
```

Но двоичная информация часто не очень удачно выводится на устройствах ASCII, поэтому, вероятно, вы не захотите осуществлять интерактивный вывод из программы *mysql*, чтобы терминальное окно не заполнилось мусором и, тем более, не заблокировалось. Наиболее часто информацию используют для отображения на веб-странице. Или можно отправить ее клиенту для загрузки (хотя такой способ чаще применяется для двоичных данных, не являющихся изображениями, таких как файлы PDF). О загрузке поговорим в рецепте 17.9.

Вывод изображения на веб-страницу реализуется при помощи тега <img>, который сообщает броузеру клиента, где взять изображение. Если вы храните изображения в виде файлов в каталоге, к которому веб-сервер имеет доступ, то можете ссылаться на него напрямую. Например, если файл изображения *iceland.jpg* расположен в каталоге */mcb/images* корневого каталога документов сервера, то вы можете сослаться на него так:

```
<img src="/mcb/images/iceland.jpg" />
```

Если вы применяете этот подход, проверьте, все ли файлы изображений имеют расширение (например, *.gif* или *.png*), которое позволит веб-серверу определить, какой тип заголовка Content-Type: следует отправлять в ответе на запрос.

Если изображения хранятся в базе данных или в каталоге, недоступном для веб-сервера, то тег <img> может ссылаться на сценарий, который знает, как извлекать изображения и отправлять их клиенту. Для этого сценарий должен отправлять ответ, содержащий заголовок Content-Type: для указания формата изображения, заголовок Content-Length:, сообщающий о количестве байт в изображении, пустую строку и, наконец, само изображение как тело ответа.

Сценарий *display_image.pl* показывает, как отправлять изображения через Сеть. Необходимо задать параметр `name`, указывающий на то, какое изображение следует вывести, кроме того, можно задать необязательный параметр `location`, показывающий, откуда нужно извлекать изображение: из таблицы `image` или из файловой системы. По умолчанию данные извлекаются из таблицы `image`. Например, следующие URL выводят изображение из базы данных и из файловой системы соответственно:

http://apache.snake.net/cgi-bin/display_image.pl?name=iceland.jpg

http://apache.snake.net/cgi-bin/display_image.pl?name=iceland.jpg;location=fs

Сценарий выглядит так:

```
#! /usr/bin/perl -w
# display_image.pl - отправить изображение через веб

use strict;
use lib qw(/usr/local/apache/lib/perl);
use CGI qw(:standard escapeHTML);
use FileHandle;
use Cookbook;

# Значения по умолчанию для каталога хранения изображений и разделителя пути
# (CHANGE THESE AS NECESSARY)
my $image_dir = "/usr/local/apache/htdocs/mcb/images";
my $path_sep = "/";

# Установить каталог и разделитель пути для Windows/DOS
if ($^O =~ /^MSWin/i || $^O =~ /^dos/)
{
    $image_dir = "D:\\apache\\htdocs\\mcb\\images";
    $path_sep = "\\";
}

my $name = param ("name");
my $location = param ("location");

# убедиться в том, что имя изображения было указано
defined ($name) or error ("image name is missing");
# использовать умолчание - "db", если местоположение не указано, или не "db" и не "fs"
(defined ($location) && $location eq "fs") or $location = "db";

my $dbh = Cookbook::connect ();

my ($type, $data);

# Если местоположение - "db", получить данные и MIME-тип из таблицы image.
# Если местоположение - "fs", получить MIME-тип из таблицы image,
# а данные считать из файловой системы.

if ($location eq "db")
{
    ($type, $data) = $dbh->selectrow_array (
                        "SELECT type, data FROM image WHERE name = ?",
```

```
                        undef,
                        $name)
                or error ("Cannot find image with name $name");
}
else
{
    $type = $dbh->selectrow_array (
                        "SELECT type FROM image WHERE name = ?",
                        undef,
                        $name)
                or error ("Cannot find image with name $name");
    my $fh = new FileHandle;
    my $image_path = $image_dir . $path_sep . $name;
    open ($fh, $image_path)
        or error ("Cannot read $image_path: $!");
    binmode ($fh);      # для работы в двоичном режиме
    my $size = (stat ($fh))[7];
    read ($fh, $data, $size) == $size
        or error ("Failed to read entire file $image_path: $!");
    $fh->close ();
}

$dbh->disconnect ();

# Отправить изображение клиенту с заголовками Content-Type: и Content-Length:.

print header (-type => $type, -Content_Length => length ($data));
print $data;

exit (0);

# ------------------------------------------------------------------------

sub error
{
my ($msg) = shift;

    print header (), start_html ("Error"), p (escapeHTML ($msg)), end_html ();
    exit (0);
}
```

17.8. Работа с баннерами

Задача

Вы хотите выводить рекламные баннеры, произвольно выбирая изображения из имеющегося набора.

Решение

Используйте сценарий, случайным образом выбирающий строку из таблицы изображений и отправляющий изображение клиенту.

Обсуждение

В только что рассмотренном сценарии *display_image.pl* предполагалось, что URL содержит параметр с указанием имени изображения для отправки клиенту. Другие приложения могут самостоятельно определять, какое изображение выводить. Весьма популярным применением веб-программирования с участием изображений является вывод рекламных баннеров на веб-страницах. Можно без труда решить эту задачу с помощью сценария, при каждом запуске выбирающего произвольное изображение. Рассмотрим сценарий на Python *banner.py*, который показывает, как это сделать, при этом «рекламой» будут служить изображения флагов из таблицы `image`:

```
#! /usr/bin/python
# banner.py - отправка произвольно выбранного баннера из таблицы image
# (если изображение не найдено, ответ не отправляется)

import sys
sys.path.insert (0, "/usr/local/apache/lib/python")
import MySQLdb
import Cookbook

conn = Cookbook.connect ()

try:
    query = "SELECT type, data FROM image ORDER BY RAND() LIMIT 1"
    cursor = conn.cursor ()
    cursor.execute (query)
    row = cursor.fetchone ()
    cursor.close ()
    if row is not None:
        (type, data) = row
        # Отправить изображение клиенту с заголовками Content-Type:
        # и Content-Length:. Заголовки Expires:, Cache-Control: и Pragma: помогают
        # предотвратить кэширование изображения в броузере и его повторное
        # использование при последующем запуске сценария.
        print "Content-Type: %s" % type
        print "Content-Length: %s" % len (data)
        print "Expires: Sat, 01 Jan 2000 00:00:00 GMT"
        print "Cache-Control: no-cache"
        print "Pragma: no-cache"
        print ""
        print data
except MySQLdb.Error, e:
    pass

conn.close ()
```

Сценарий *banner.py* отправляет еще несколько заголовков в дополнение к обычным `Content-Type:` и `Content-Length:`. Эти новые заголовки не дают броузерам кэшировать изображение. `Expires:` указывает дату в прошлом, чтобы сообщить броузеру о том, что изображение уже устарело. Заголовки `Cache-Control:` и `Pragma:` сообщают броузеру о том, что не следует кэшировать изо-

бражение. Сценарий отправляет оба заголовка, так как некоторые броузеры понимают только первый, а некоторые – только второй.

Почему запрещается кэширование? Если не сделать этого, броузер отправит запрос *banner.py* только первый раз, когда он увидит такую ссылку. При последующих запросах сценария броузер будет повторно использовать изображение, что сделает невозможным вывод произвольно выбранного изображения для каждой такой ссылки.

Установите сценарий *banner.py* в каталог *cgi-bin*. Для того чтобы разместить баннер на веб-странице, используйте тег `<img>` для вызова сценария. Например, если сценарий сохранен как */cgi-bin/banner.py*, то следующая страница будет ссылаться на него, выводя изображение под поясняющим абзацем:

```
<!-- bannertest1.html - строка с одной ссылкой на сценарий вывода баннеров -->
<html>
<head>
<title>Banner Ad Test Page 1</title>
</head>
<body bgcolor="white">

<p>You should see an image below this paragraph.</p>

<img src="/cgi-bin/banner.py" />

</body>
</html>
```

Если запросить эту страницу, она должна вывести изображение, причем при каждом перезапросе страницы должно выводиться новое изображение. (Я считаю, что вы к этому времени загрузили в таблицу `image` множество изображений, используя сценарий *store_image.pl* из рецепта 17.6. В противном случае вы можете вообще ничего не увидеть!) Если вы изменяете *banner.py* так, чтобы он не отправлял заголовки, управляющие кэшированием, то, вероятно, при каждой загрузке страницы будет отображаться одна и та же картинка.

Управляющие кэшированием заголовки запрещают его для ссылок на *banner.py* в процессе последовательных обращений к странице. Все усложняется, если несколько ссылок на сценарий присутствует на *одной* странице – смотрите, что получится:

```
<!-- bannertest2.html - страница с несколькими ссылками
                        на сценарий вывода баннеров -->
<html>
<head>
<title>Banner Ad Test Page 2</title>
</head>
<body bgcolor="white">

<p>You should see two images below this paragraph,
and they probably will be the same.</p>

<img src="/cgi-bin/banner.py" />
<img src="/cgi-bin/banner.py" />
```

```
<p>You should see two images below this paragraph,
and they probably will be different.</p>

<img src="/cgi-bin/banner.py?image1" />
<img src="/cgi-bin/banner.py?image2" />

</body>
</html>
```

Две первые ссылки на *banner.py* идентичны. Запросив эту страницу в броузере вы, вероятно, заметите, что на веб-сервер отправляется только один запрос и для обеих ссылок подставляется одно и то же изображение. В результате две первые картинки будут идентичны. Две другие ссылки на *banner.py* демонстрируют решение проблемы. В конце URL ссылок присутствуют параметры, которые делают их разными. Сценарий *banner.py* не использует эту информацию, но нам удалось обмануть броузер – считая, что ссылки разные, броузер отправляет два запроса на изображение. В результате вторая пара включает разные изображения – если только не получится так, что *banner.py* случайно выберет одну и ту же картинку два раза подряд.

См. также

Если вы работаете с версией MySQL более ранней, чем 3.23.2, то не сможете использовать инструкцию `ORDER BY RAND()` из запроса, выбирающего изображения. Обходной маневр описан в рецепте 13.7.

17.9. Использование результатов запроса для загрузки файлов

Задача

Вы хотите отправить информацию из базы данных в броузер для скачивания, а не для отображения.

Решение

К сожалению, нет хорошего способа инициирования загрузки. Броузер будет обрабатывать полученную информацию в соответствии со значением заголовка `Content-Type:`, и если у него есть обработчик для значения этого заголовка, данные будут интерпретированы соответствующим образом. Однако можно обмануть броузер, используя такой тип содержимого, для которого вряд ли найдется обработчик.

Обсуждение

В предыдущих разделах этой главы было показано, как встраивать результаты запросов в веб-страницы, выводить их в виде абзацев, списков, таблиц или изображений. Но что, если вы хотите сформировать результат запроса, который пользователь мог бы сохранить в файл? Сам ответ получить не-

сложно: отправляем перед собственно данными заголовок `Content-Type:`, например `text/plain` для простого текста, `image/jpeg` для JPEG-изображения, `application/pdf` или `application/msexcel` для документа PDF или Excel. Затем отправляем пустую строку и содержимое результата запроса. Проблема в том, что невозможно заставить броузер загрузить информацию. Если по типу содержимого броузер может определить, что с ним делать, то он попытается обработать информацию. Если броузер умеет выводить текст или изображения, он это и сделает. Если он подумает, что необходимо передать PDF- или Excel-документ в программу просмотра файлов PDF или в Excel, он так и поступит. Большинство броузеров позволяет пользователю явно указать на необходимость загрузки файла (например, щелчком по ссылке правой кнопкой мыши или при нажатой клавише <Ctrl>), но это клиентский механизм, к которому нельзя получить доступ со стороны сервера.

Единственное, что можно сделать, – попытаться обмануть броузер, сфальсифицировав тип содержимого. Наиболее общим типом является `application/octet-stream`. У большей части пользователей вряд ли имеется обработчик для такого содержимого, поэтому если отправить ответ, указав такой тип, то весьма вероятно, что броузер начнет загрузку. Очевидный недостаток этого приема в том, что в ответе содержится ложный указатель типа содержимого. Можно упростить ситуацию, предложив имя по умолчанию, которое броузер должен использовать при сохранении файла. Если имя файла имеет расширение, информирующее о типе файла, такое как *.txt*, *.jpg*, *.pdf* или *.xls*, то клиенту (или операционной системе клиентского хоста) будет проще определить, как обрабатывать файл. Для указания имени по умолчанию включите в ответ заголовок `Content-Disposition:`:

```
Content-disposition: attachment; filename="рекомендуемое_имя_файла"
```

Рассмотрим сценарий на PHP *download.php*, который реализует вывод содержимого для загрузки. При первом обращении к сценарию он выводит страницу, содержащую ссылку, которую можно выбрать для запуска загрузки. Ссылка также указывает на *download.php*, но содержит параметр `download`. Если вы активизируете эту ссылку, она повторно вызовет сценарий, который принимает параметр и отвечает запуском запроса, извлечением результирующего множества и отправкой его в броузер для загрузки. Заголовки `Content-Type:` и `Content-Disposition:` задаются при помощи вызова функции `header()`. (Это необходимо сделать, прежде чем сценарий сформирует какой-то другой вывод, иначе не будет смысла в вызове `header()`.)

```
<?php
# download.php - извлекает результат запроса и отправляет его пользователю
# для загрузки, а не для вывода на веб-странице

include "Cookbook.php";
include "Cookbook_Webutils.php";

$title = "Result Set Downloading Example";

# Если параметр download не указан, вывести страницу с инструкциями

if (!get_param_val ("download"))
```

```
{
    # сформировать самовызывающий URL, включающий параметр download
    $url = get_self_path () . "?download=1";
?>

<html>
<head>
<title><?php print ($title); ?></title>
</head>
<body bgcolor="white">

<p>
Select the following link to commence downloading:
<a href="<?php print ($url); ?>">download</a>
</p>

</body>
</html>

<?php
    exit ();
}   # конец "if"

# Параметр download указан; извлечь результат запроса и отправить его клиенту
# в виде документа - текста с разделителями -  табуляцией, признак конца строки -
# символ новой строки. Использовать тип содержимого application/octet-stream,
# пытаясь инициировать загрузку, и предложить имя по умолчанию "result.txt".

$conn_id = cookbook_connect ();

$query = "SELECT * FROM profile";
if (!($result_id = mysql_query ($query, $conn_id)))
    die ("Cannot execute query\n");

header ("Content-Type: application/octet-stream");
header ("Content-Disposition: attachment; filename=\"result.txt\"");

while ($row = mysql_fetch_row ($result_id))
    print (join ("\t", $row) . "\n");
mysql_free_result ($result_id);

mysql_close ($conn_id);

?>
```

Сценарий *download.php* использует несколько функций, о которых мы еще не говорили. Функция `get_self_path()` возвращает путь к самому сценарию. Используется для построения URL, указывающего на сценарий и содержащего параметр `download`. Функция `get_param_val()` используется для проверки наличия параметра. Эти функции включены в файл *Cookbook_Webutils.php* и будут обсуждаться в рецептах 18.1 и 18.5.

Есть еще один способ формирования загружаемого содержимого – генерируем результат запроса, записываем его в файл (с серверной стороны), архивируем его и отправляем результат в броузер. Можно ожидать, что броузер запустит какую-нибудь утилиту для разархивирования исходного файла.

18

Обработка ввода через Web с помощью MySQL

18.0. Введение

В предыдущей главе было рассказано о том, как извлекать информацию из MySQL и отображать ее на веб-страницах, используя различные конструкции HTML, такие как таблицы и гиперссылки. То есть было рассмотрено использование MySQL для отправки информации в одном направлении (от веб-сервера к пользователю). Но веб-программирование с участием баз данных может заниматься и задачами сбора информации, отправленной в обратном направлении – от пользователя к веб-серверу, например, для содержимого заполненной формы. Если вы обрабатываете форму опроса, то можете сохранить информацию для дальнейшего использования. Если форма содержит ключевые слова поиска, можно использовать их как основу для запроса, который ищет в базе данных информацию, необходимую пользователю.

MySQL во всех этих операциях выступает во вполне очевидной роли хранилища информации или источника, откуда извлекаются результаты поиска. Но прежде чем обрабатывать ввод из формы, необходимо такую форму создать и отправить пользователю. MySQL может помочь и в этом, поскольку можно использовать информацию, хранящуюся в базе данных, для создания таких элементов формы, как переключатели (radio button), флажки (checkbox), всплывающие (pop-up) меню и прокручиваемые списки:

- Вы можете выбрать набор элементов из таблицы стран, штатов или провинций и преобразовать их во всплывающее меню для формы сбора адресной информации.
- Можно использовать список разрешенных значений столбца типа `ENUM`, содержащего возможные обращения (Mr., Mrs. и т. д.) для формирования переключателей.
- Можно использовать список доступных цветов, размеров или стилей, хранящихся в складской базе данных, для создания полей формы заказа одежды.

- Если у вас есть приложение, позволяющее пользователю выбрать базу данных или таблицу, то можно выполнить предложение SHOW DATABASES или SHOW TABLES и преобразовать полученные имена в элементы списка.

Используя содержимое базы данных для построения элементов формы, вы уменьшаете объем знаний о таблицах, который необходимо иметь вашим программам, и позволяете им автоматически определить, что им необходимо. Сценарий, использующий базу данных для того, чтобы понять, как ему создать элементы формы, адаптируется к изменениям данных. Чтобы добавить новую страну, создайте новую строку в таблице, хранящей список стран. Чтобы добавить новое обращение, измените определение столбца ENUM. В каждом из случаев вы изменяете набор значений для элементов формы, обновляя базу данных, но не изменяя сценарий. Сценарий автоматически адаптируется к сделанным изменениям, никакого дополнительного программирования не требуется.

Первая часть главы рассматривает вопросы, связанные с обработкой ввода через Web:

Генерирование форм и элементов форм. Одним из способов применения содержимого базы данных для построения формы является выбор записей из таблицы и использование их для создания списка вариантов выбора. Можно использовать и метаданные. Существует естественное соответствие столбцов ENUM элементам формы с выбором ровно одного значения, таким как переключатели или всплывающее меню. В обоих случаях выбрано может быть всего одно значение из набора. Аналогичное соответствие есть между столбцами SET и элементами, допускающими выбор нескольких значений, такими как флажки. Можно выбирать любые (в том числе – все) возможные значения. Для создания элементов формы на основе метаданных получите описание столбца из результата, возвращенного предложением SHOW COLUMNS, извлеките набор возможных значений и используйте их в элементе формы.

Инициализация форм при помощи содержимого базы данных. В дополнение к применению базы данных для создания структурных элементов форм вы можете использовать ее для инициализации значений полей форм. Например, чтобы разрешить пользователю редактировать имеющуюся запись, извлеките ее из базы данных и загрузите в поле редактируемой формы перед отправкой формы пользователю.

Обработка ввода, полученного через Web. Речь идет не только о вводе в поля формы, но и о содержимом переданных клиентом файлов или параметрах, содержащихся в URL. Вне зависимости от источника информации возникает ряд схожих вопросов, связанных с ее обработкой: извлечение и декодирование данных, проверка корректности и соответствия ограничениям, повторное кодирование для построения запроса (чтобы не допустить формирования некорректных запросов или неточного сохранения данных).

Во второй части главы приведено несколько способов применения приемов, описанных в первой части. Показано, как использовать MySQL для создания интерфейса веб-поиска, организации постраничного вывода, содержа-

щего ссылки на предыдущую и последующую страницы, реализации постраничного счетчика посещаемости и протоколирования, ведения логов Apache с использованием базы данных.

Сценарии примеров на Perl, PHP и Python хранятся в каталоге *apache* дистрибутива `recipes`. Для JSP сценарии расположены в каталоге *tomcat*; возможно, вы уже установили их в процессе создания среды исполнения для приложения `mcb` (рецепт 16.3). Библиотечные функции, используемые в сценариях, содержатся в файлах каталога *lib*. Сценарии создания таблиц, упоминаемых в примерах, расположены в каталоге *tables*.

Обратите внимание: несмотря на то что сценарии главы предназначены для вызова из броузера, многие из них (за исключением страниц JSP) могут вызываться и из командной строки, если вам нужно посмотреть на формируемый ими HTML-код.

Чтобы конкретизировать обсуждение, будем использовать в большинстве примеров обработки форм такой сюжет: вы занимаетесь коммерческой деятельностью в привлекательной сфере «конструирования коров» – производите собираемые по заказу керамические фигурки коров. Вам хотелось бы написать приложение, позволяющее клиентам в режиме реального времени выбирать различные характеристики товара. Для каждого заказа необходимо хранить различные виды данных:

Цвет коровы. Список доступных цветов периодически меняется, поэтому для обеспечения гибкости можно хранить значения в таблице базы данных. Для изменения набора цветов, которые могут выбирать клиенты, просто обновите содержимое таблицы.

Размер коровы. Есть фиксированный набор размеров, который редко изменяется (маленький, средний, большой), поэтому значения можно представить как элементы столбца `ENUM`.

Все необходимые аксессуары. Имеются в виду колокольчик, рога, красивый бантик на хвосте и кольцо в носу. Аксессуары можно представить столбцом `SET`, так как заказчик может пожелать выбрать несколько таких значений. Кроме того, ваш опыт подсказывает, что обычно выбирают рога и колокольчик, так что можно использовать их как значения по умолчанию.

Фамилия и адрес клиента (улица, город, штат). Возможные названия штатов уже хранятся в таблице `states`, их можно использовать как основу для соответствующего элемента формы.

Учитывая все вышесказанное, таблицу `cow_order` можно спроектировать так:

```
CREATE TABLE cow_order
(
    id          INT UNSIGNED NOT NULL AUTO_INCREMENT,
    # цвет коровы, размер фигурки и аксессуары
    color       CHAR(20),
    size        ENUM('small','medium','large') DEFAULT 'medium',
    accessories SET('cow bell','horns','nose ring','tail ribbon')
                    DEFAULT 'cow bell,horns',
```

```
    # фамилия клиента, улица, город и штат (аббревиатура)
    cust_name   CHAR(40),
    cust_street CHAR(40),
    cust_city   CHAR(40),
    cust_state  CHAR(2),
    PRIMARY KEY (id)
);
```

Столбец `id` содержит уникальный идентификатор каждой записи. Иметь такое значение удобно и даже необходимо; это станет понятно в рецепте 18.4, где показано, как использовать веб-формы для редактирования существующих записей. Выполняя подобные операции, вы должны иметь возможность указать на то, какую запись редактировать (что затруднительно в отсутствие уникального идентификатора).

Список доступных цветов поддерживается в отдельной таблице `cow_color`:

```
CREATE TABLE cow_color
(
    color   CHAR(20)
);
```

Будем считать, что таблица цветов выглядит так:

```
+---------------+
| color         |
+---------------+
| Black         |
| Black & White |
| Brown         |
| Cream         |
| Red           |
| Red & White   |
| See-Through   |
+---------------+
```

Приложение может использовать эти таблицы для формирования элементов списка в форме ввода заказа, что избавляет его от необходимости содержать в себе массу специфической информации об имеющихся возможностях. Следующие разделы описывают, как это сделать и как обрабатывать ввод, получаемый при отправке формы пользователем.

18.1. Создание форм в сценариях

Задача

Вы хотите написать сценарий, который получает ввод от пользователя.

Решение

Создайте в сценарии форму для заполнения и отправьте ее пользователю. Сценарий может сам вызывать себя повторно для обработки содержимого формы после ее отправки пользователем.

Обсуждение

Веб-формы – это удобный способ получения информации от посетителей вашего сайта, например, они могут вводить ключевые слова для выполнения поиска или заполнять анкету. Формы также удобны для разработчиков, поскольку обеспечивают возможность сопоставления значений данных с названиями, по которым на них можно ссылаться.

Форма начинается тегом `<form>`, а заканчивается тегом `</form>`. Между этими двумя тегами можно размещать другие HTML-конструкции, включая специальные элементы, которые превратятся в поля ввода на странице, выводимой броузером. Начинающий форму тег `<form>` содержит два атрибута: `action` и `method`. Атрибут `action` сообщает броузеру, что следует сделать с формой, когда пользователь отправит ее. Он представляет собой URL сценария, который должен быть вызван для обработки содержимого формы. Атрибут `method` указывает броузеру, какой вид запроса HTTP необходимо использовать для отправки формы; возможны значения `GET` или `POST`, в зависимости от того, какой тип запроса для отправки формы вы предпочитаете. Различие методов запроса обсуждается в рецепте 18.5, а пока мы будем использовать только `POST`.

Большинство веб-сценариев, приведенных в главе, ведет себя во многом похоже:

- При первом вызове сценарий генерирует форму и отправляет ее пользователю для заполнения.
- Атрибут `action` формы указывает на тот же самый сценарий, поэтому после того, как пользователь завершает заполнение формы и отсылает ее, сценарий вновь вызывается для обработки содержимого формы.
- Сценарий определяет, вызван ли он впервые или же он должен обрабатывать полученную форму, проверяя, с какими параметрами он вызван. При первом вызове окружение не будет содержать параметров, указанных в форме.

Конечно, это не единственный подход, который можно применять. Одним из альтернативных вариантов может быть помещение формы в статическую страницу HTML, указывающую на сценарий, обрабатывающий форму. Другим – создание двух сценариев, первый из которых генерирует форму, а второй – обрабатывает ее.

Если сценарий создания формы хочет повторно вызывать себя при отправке пользователем формы, он должен определить путь к самому себе в дереве каталогов веб-сервера и использовать это значение для атрибута `action` открывающего тега `<form>`. Например, если сценарий установлен как */cgi-bin/myscript*, тег можно записать так:

```
<form action="/cgi-bin/myscript" method="POST">
```

Каждый API предлагает для сценария способ получения пути к самому себе, поэтому не нужно жестко кодировать имя в сценарии. Тем самым вы обеспечиваете возможность его установки в любом месте.

Для сценариев на Perl модуль CGI.pm предлагает три метода создания элементов <form> и формирования атрибута action. Методы start_form() и end_form() создают открывающий и закрывающий теги формы, а url() возвращает путь к сценарию. При помощи этих трех методов сценарий может сгенерировать форму так:

```
print start_form (-action => url (), -method => "POST");
# ... генерируем элементы формы ...
print end_form ();
```

В действительности атрибут method необязателен; если не указать его, start_form() передает значение по умолчанию – POST.

В PHP простой способ получения пути к сценарию – использование глобальной переменной $PHP_SELF:

```
print ("<form action=\"$PHP_SELF\" method=\"POST\">\n");
# ... генерируем элементы формы...
print ("</form>\n");
```

Однако в некоторых конфигурациях PHP (когда отключен параметр register_globals) такой способ не работает.[1] Можно получить путь к сценарию, обратившись к члену "PHP_SELF" массива $HTTP_SERVER_VARS или, начиная с версии PHP 4.1, массива $_SERVER. Для того чтобы просто получить путь к сценарию способом, подходящим для различных версий языка, приходится терять много времени на проверку разных источников информации, поэтому разумно написать функцию, которая получала бы для нас этот путь. Рассмотрим функцию get_self_path(), которая использует массив $_SERVER, если он доступен, и обращается к $HTTP_SERVER_VARS или $PHP_SELF в противном случае. То есть эта функция предпочитает использовать последние возможности языка, но будет работать и для сценариев старых версий PHP:

```
function get_self_path ()
{
global $HTTP_SERVER_VARS, $PHP_SELF;

    if (isset ($_SERVER["PHP_SELF"]))
        $val = $_SERVER["PHP_SELF"];
    else if (isset ($HTTP_SERVER_VARS["PHP_SELF"]))
        $val = $HTTP_SERVER_VARS["PHP_SELF"];
    else
        $val = $PHP_SELF;
    return ($val);
}
```

Переменные $HTTP_SERVER_VARS и $PHP_SELF являются глобальными, но должны быть объявлены таковыми явно, при помощи ключевого слова global, если они используются не в глобальном контексте, а, например, внутри функции. $_SERVER – это «суперглобальный» массив, доступный в любом контексте без объявления его глобальным.

---

[1] Параметр register_globals описан в рецепте 18.5.

Функция `get_self_path()` входит в состав библиотечного файла *Cookbook_Webutils.php* из каталога *lib* дистрибутива `recipes`. Если вы устанавливаете этот файл в каталог, который PHP просматривает в поиске включаемых файлов, то сценарий может получить путь к самому себе и использовать его для генерирования формы так:

```
include "Cookbook_Webutils.php";

$self_path = get_self_path ();
print ("<form action=\"$self_path\" method=\"POST\">\n");
# ... генерируем элементы формы...
print ("</form>\n");
```

Сценарии на Python могут получить путь к сценарию, импортируя модуль `os` и обращаясь к члену `SCRIPT_NAME` объекта `os.environ`:

```
import os

print "<form action=\"" + os.environ["SCRIPT_NAME"] + "\" method=\"POST\">"
# ... генерируем элементы формы......
print "</form>"
```

Для страниц на JSP путь сценария доступен через неявный объект `request`, который становится доступным благодаря процессору JSP. Используйте метод объекта `getRequestURI()`:

```
<form action="<%= request.getRequestURI () %>" method="POST">
<%-- ... генерируем элементы формы...--%>
</form>
```

См. также

Все примеры этого раздела имеют пустое тело, заключенное между тегами открытия и закрытия формы. Для того чтобы форма приносила пользу, необходимо создать элементы, соответствующие тем видам данных, которые вы хотели бы получить от пользователей. Можно «зашить» эти элементы в сценарий; в рецептах 18.2 и 18.3 показано, как MySQL может помочь в динамическом создании элементов на основе информации, хранящейся в базе данных.

18.2. Создание элементов формы с возможностью выбора одного значения

Задача

Форма должна содержать поле, которое предлагает пользователю много вариантов, но разрешает выбрать только один из них.

Решение

Используйте элементы с возможностью выбора единственного значения из списка (single-pick list element). К ним относятся наборы переключателей, всплывающие меню или прокручиваемые списки.

Обсуждение

Элементы формы с возможностью выбора единственного значения позволяют предлагать на выбор несколько вариантов, из которых можно указать всего один. В нашем примере есть несколько наборов таких элементов:

- Список цветов из таблицы `cow_color`. Их можно получить при помощи такого запроса:

```
mysql> SELECT color FROM cow_color ORDER BY color;
+---------------+
| color         |
+---------------+
| Black         |
| Black & White |
| Brown         |
| Cream         |
| Red           |
| Red & White   |
| See-Through   |
+---------------+
```

  Обратите внимание на то, что некоторые названия цветов содержат символ `&`, обладающий специальным значением в HTML. То есть при использовании для элементов списка такие значения необходимо будет кодировать по стандарту HTML. (На самом деле мы будем выполнять кодирование для всех списковых элементов формы, но эти значения наглядно демонстрируют, что кодирование – хорошая привычка.)

- Список допустимых размеров фигурок из столбца `size` таблицы `cow_order`. Столбец относится к типу `ENUM`, поэтому возможные значения и значения по умолчанию можно получить при помощи предложения `SHOW COLUMNS`:

```
mysql> SHOW COLUMNS FROM cow_order LIKE 'size'\G
*************************** 1. row ***************************
  Field: size
   Type: enum('small','medium','large')
   Null: YES
    Key:
Default: medium
  Extra:
```

- Список названий и аббревиатур штатов, которые хранятся в таблице `states`:

```
mysql> SELECT abbrev, name FROM states ORDER BY name;
+--------+----------------+
| abbrev | name           |
+--------+----------------+
| AL     | Alabama        |
| AK     | Alaska         |
| AZ     | Arizona        |
| AR     | Arkansas       |
| CA     | California     |
```

```
| CO     | Colorado       |
| CT     | Connecticut    |
...
```

Количество значений для выбора во всех этих списках разное: 3 размера, 7 цветов и 50 штатов. Значения размеров лучше всего представить в виде переключателей, в прокручиваемом списке необходимости нет из-за небольшого количества значений для выбора. Множество цветов приемлемо представить любым из элементов для выбора единственного значения. Оно достаточно маленькое, так что набор переключателей займет не слишком много места, но и достаточно большое для того, чтобы использовать прокрутку – особенно если будут добавляться дополнительные цвета. Список штатов, вероятно, имеет слишком много значений, чтобы быть представленным в виде переключателей, поэтому наиболее разумным его отображением будет всплывающее меню или прокручиваемый список.

Рассмотрим синтаксис для каждого из типов элементов, а затем попробуем сгенерировать их из сценариев.

Переключатели. Группа переключателей состоит из элементов `<input>` типа `radio` с одинаковыми атрибутами `name`. У каждого элемента также есть атрибут `value`. Отображаемый текст можно указать после тега `<input>`. Для того чтобы пометить значение как исходный вариант выбора по умолчанию, добавьте атрибут `checked`. Следующая группа переключателей выводит возможные размеры фигурок коров, при этом среднее значение (`medium`) помечено как выбираемое по умолчанию:

```
<input type="radio" name="size" value="small" />small
<input type="radio" name="size" value="medium" checked="checked" />medium
<input type="radio" name="size" value="large" />large
```

Всплывающие меню. Всплывающее меню – это список, заключенный между тегами `<select>` и `</select>`, в котором каждый пункт меню заключен в теги `<option>` и `</option>`. У каждого элемента `<option>` есть атрибут `value`, а в теле элемента содержится видимый текст тега для вывода. Для указания выбора по умолчанию добавьте в соответствующий элемент `<option>` атрибут `selected`. Если ни один из элементов не помечен, то умолчанием становится первый элемент, как это и сделано в примере:

```
<select name="color">
<option value="Black">Black</option>
<option value="Black &amp; White">Black &amp; White</option>
<option value="Brown">Brown</option>
<option value="Cream">Cream</option>
<option value="Red">Red</option>
<option value="Red &amp; White">Red &amp; White</option>
<option value="See-Through">See-Through</option>
</select>
```

Прокручиваемые списки. Прокручиваемые списки отображаются как набор значений в окне. Список может включать больше значений, чем помещается в окне, тогда броузер выводит полосу прокрутки, с помощью

которой пользователь может увидеть остальные элементы. Синтаксис для прокручиваемых списков похож на синтаксис всплывающих меню, только открывающий тег <select> содержит атрибут size, указывающий, сколько строк списка должны быть видны в окне. По умолчанию список прокрутки является элементом с возможностью выбора единственного значения (в рецепте 18.3 рассказано о том, как разрешить для него множественный выбор).

Следующий прокручиваемый список с возможностью выбора единственного значения содержит элемент для каждого штата США, при этом одновременно выводятся только шесть значений:

```
</select>
<select name="state" size="6">
<option value="AL">Alabama</option>
<option value="AK">Alaska</option>
<option value="AZ">Arizona</option>
<option value="AR">Arkansas</option>
<option value="CA">California</option>
...
<option value="WV">West Virginia</option>
<option value="WI">Wisconsin</option>
<option value="WY">Wyoming</option>
</select>
```

У всех этих списковых элементов есть много общего:

- Имя элемента. Когда пользователь отправляет форму, броузер сопоставляет это имя со значением, выбранным пользователем.
- Набор значений, по одному для каждого элемента списка. Так определяются значения, доступные для выбора.
- Необязательное значение по умолчанию, которое определяет, какое из значений списка будет выбрано при первоначальном отображении списка броузером.
- Видимый текст тегов (label), по одному значению для каждого элемента. Он определяет то, что пользователь увидит при выводе формы, и отбрасывается при отсылке формы пользователем.

Для формирования спискового элемента формы на основе содержимого базы данных запустите запрос для выбора соответствующих значений и видимого текста, закодируйте все содержащиеся в них специальные символы и добавьте теги HTML для того типа списка, который вы хотите вывести. Если вам нужно указать вариант выбора по умолчанию, добавьте атрибут checked или selected для соответствующего элемента в списке.

Давайте посмотрим, как создать элементы формы для списков цветов и штатов, получаемых извлечением строк из таблицы.

В JSP вы можете вывести переключатель для цветов, применяя теги JSTL так (названия цветов используются и как значения, и как видимый текст, поэтому выводятся дважды):

```
<sql:query var="rs" dataSource="${conn}">
    SELECT color FROM cow_color ORDER BY color
</sql:query>

<c:forEach var="row" items="${rs.rows}">
    <input type="radio" name="color"
        value="<c:out value="${row.color}" />"
    /><c:out value="${row.color}" /><br />
</c:forEach>
```

Тег `<c:out>` выполняет кодирование по стандарту HTML, поэтому содержащийся в некоторых значениях цветов символ `&` будет автоматически преобразован в `&amp;`, и проблемы при отображении результирующей веб-страницы не возникнут.

Для вывода всплывающего меню запрос будет таким же, изменится лишь цикл извлечения строк:

```
<sql:query var="rs" dataSource="${conn}">
    SELECT color FROM cow_color ORDER BY color
</sql:query>

<select name="color">
<c:forEach var="row" items="${rs.rows}">
    <option value="<c:out value="${row.color}" />">
    <c:out value="${row.color}" /></option>
</c:forEach>
</select>
```

Всплывающее меню очень легко заменить прокручиваемым списком – просто добавьте атрибут `size` в открывающий тег `<select>`. Например, для одновременного отображения трех цветов сформируйте список так:

```
<sql:query var="rs" dataSource="${conn}">
    SELECT color FROM cow_color ORDER BY color
</sql:query>

<select name="color" size="3">
<c:forEach var="row" items="${rs.rows}">
    <option value="<c:out value="${row.color}" />">
    <c:out value="${row.color}" /></option>
</c:forEach>
</select>
```

Создание спискового элемента для штатов будет аналогичным, только в данном случае видимый текст не совпадает со значениями. Для того чтобы сделать видимый текст более понятным для пользователей, будем выводить полные названия штатов. Но значение, возвращаемое при передаче формы, должно быть аббревиатурой, так как именно она хранится в таблице `cow_order`. Для формирования такого списка выбираем как названия штатов, так и их аббревиатуры, затем вставляем их в соответствующие места элементов списка. Например, для создания всплывающего меню сделайте следующее:

```
<sql:query var="rs" dataSource="${conn}">
    SELECT abbrev, name FROM states ORDER BY name
```

```
</sql:query>

<select name="state">
<c:forEach var="row" items="${rs.rows}">
    <option value="<c:out value="${row.abbrev}" />">
    <c:out value="${row.name}" /></option>
</c:forEach>
</select>
```

Примеры на JSP используют подход, обеспечивающий индивидуальный вывод каждого значения списка. Формирование спискового элемента в CGI.pm-сценариях Perl происходит по другому принципу: сначала извлекается информация из базы данных, затем вся она передается функции, которая возвращает строку, представляющую элемент формы. Элементы единичного выбора генерируются функциями `radio_group()`, `popup_menu()` и `scrolling_list()`. У них есть ряд общих аргументов:

`name`

Указывает, как вы хотите назвать элемент.

`values`

Указывает значения элементов в списке. Аргумент должен быть ссылкой на массив.

`default`

Указывает изначально выбранный элемент в списке. Аргумент не обязателен. В наборе переключателей CGI.pm автоматически по умолчанию выбирает первый, если аргумент отсутствует. Чтобы отменить такое поведение, укажите значение по умолчанию, которое не входит в список `values`. (Пустая строка и `undef` не разрешены.)

`labels`

Указывает видимый текст для каждого значения. Аргумент не обязателен; если он отсутствует, то CGI.pm использует значения в качестве видимого текста. В противном случае аргумент `labels` должен быть ссылкой на хеш, который сопоставляет каждому значению видимый текст. Например, при формировании спискового элемента для цветов коров значения и видимый текст совпадают, поэтому в аргументе `labels` нет необходимости. А вот при выводе списка штатов `labels` будет ссылкой на хеш, сопоставляющий каждой аббревиатуре штата его полное название.

Некоторые функции принимают дополнительные аргументы. Для `radio_group()` можно указать аргумент `linebreak`, означающий вертикальный, а не горизонтальный вывод переключателей. Функция `scrolling_list()` принимает аргумент `size`, который указывает, сколько элементов должно выводиться одновременно. (Документация по CGI.pm описывает дополнительные аргументы, которые не используются в книге. Например, есть аргументы для представления переключателей в табличном формате, но я не стану так изощряться.)

Для создания элемента формы на основе цветов из таблицы `cow_color` необходимо извлечь их в массив:

```
my $color_ref = $dbh->selectcol_arrayref (
                    "SELECT color FROM cow_color ORDER BY color");
```

`selectcol_arrayref()` возвращает ссылку на массив. Значение массива можно получить по ссылке так:

```
my @colors = @{$color_ref};
```

Но аргумент `values` функций CGI.pm, создающих списковые элементы, в любом случае должен быть ссылкой, так что используем ссылку и для `$color_ref`. Для создания переключателей, всплывающего меню или прокручиваемого списка с единичным выбором вызовите функцию так:

```
print radio_group (-name => "color",
                   -values => $color_ref,
                   -linebreak => 1);       # выводить кнопки вертикально

print popup_menu (-name => "color",
                   -values => $color_ref);

print scrolling_list (-name => "color",
                   -values => $color_ref,
                   -size => 3);            # выводить 3 элемента одновременно
```

Значения и видимый текст в списке цветов совпадают, поэтому не нужно указывать аргумент `labels`; по умолчанию CGI.pm использует значения как видимый текст. Обратите внимание на то, что мы не выполняем HTML-кодирование цветов, хотя некоторые из них содержат символ `&`. В CGI.pm функции генерирования элементов формы автоматически осуществляют HTML-кодирование, в отличие от его функций создания элементов не для форм.

Для формирования списка штатов, в котором значениями являются аббревиатуры, а видимым текстом – полные названия, необходимо использовать аргумент `labels`. Он должен быть ссылкой на хеш, отображающий каждое значение на соответствующее название. Создаем список значений и хеш видимого текста так:

```
my @state_values;
my %state_labels;
my $sth = $dbh->prepare ("SELECT abbrev, name FROM states ORDER BY name");
$sth->execute ();
while (my ($abbrev, $name) = $sth->fetchrow_array ())
{
    push (@state_values, $abbrev);  # сохранение всех значений в массиве
    $state_labels{$abbrev} = $name; # сопоставление каждому значению названия
}
```

Передайте получившиеся список и хеш по ссылке в функцию `popup_menu()` или `scrolling_list()` в зависимости от того, какой элемент вы хотите построить:

```
print popup_menu (-name => "state",
                   -values => \@state_values,
                   -labels => \%state_labels);
```

```
print scrolling_list (-name => "state",
                      -values => \@state_values,
                      -labels => \%state_labels,
                      -size => 6);            # показывать 6 элементов одновременно
```

Если вы используете API, в котором нет готовых функций создания элементов форм подобно CGI.pm, то можно выводить HTML по мере извлечения значений списка из MySQL или написать функции, которые будут генерировать для вас элементы списка. Далее будут рассмотрены реализации обоих способов на языках PHP и Python.

В PHP список значений таблицы `cow_color` можно представить в виде всплывающего меню при помощи цикла выборки и вывода записей:

```
$query = "SELECT color FROM cow_color ORDER BY color";
$result_id = mysql_query ($query, $conn_id);
print ("<select name=\"color\">\n");
while (list ($color) = mysql_fetch_row ($result_id))
{
    $color = htmlspecialchars ($color);
    print ("<option value=\"$color\">$color</option>\n");
}
mysql_free_result ($result_id);
print ("</select>\n");
```

Код на Python, делающий то же самое, выглядит так:

```
query = "SELECT color FROM cow_color ORDER BY color"
cursor = conn.cursor ()
cursor.execute (query)
print "<select name=\"color\">"
for (color, ) in cursor.fetchall ():
    color = cgi.escape (color, 1)
    print "<option value=\"%s\">%s</option>" % (color, color)
cursor.close ()
print "</select>"
```

Для списка штатов необходимы отличающиеся друг от друга видимый текст и значения, поэтому код становится несколько сложнее. На PHP он будет таким:

```
$query = "SELECT abbrev, name FROM states ORDER BY name";
$result_id = mysql_query ($query, $conn_id);
print ("<select name=\"state\">\n");
while (list ($abbrev, $name) = mysql_fetch_row ($result_id))
{
    $abbrev = htmlspecialchars ($abbrev);
    $name = htmlspecialchars ($name);
    print ("<option value=\"$abbrev\">$name</option>\n");
}
mysql_free_result ($result_id);
print ("</select>\n");
```

А на Python таким:

```
query = "SELECT abbrev, name FROM states ORDER BY name"
cursor = conn.cursor ()
cursor.execute (query)
print "<select name=\"state\">"
for (abbrev, name) in cursor.fetchall ():
    abbrev = cgi.escape (abbrev, 1)
    name = cgi.escape (name, 1)
    print "<option value=\"%s\">%s</option>" % (abbrev, name)
cursor.close ()
print "</select>"
```

Переключатели и прокручиваемые списки можно получить аналогичным способом. Но вместо этого давайте применим другой подход и создадим набор функций, которые, получив соответствующую информацию, будут генерировать элементы формы. Функции возвращают строку, представляющую определенный вид элементов формы, и вызываются с такими аргументами:

```
make_radio_group (name, values, labels, default, vertical)
make_popup_menu (name, values, labels, default)
make_scrolling_list (name, values, labels, default, size, multiple)
```

У функций есть несколько общих аргументов:

`name`

Указывает имя элемента формы.

`values`

Массив или список значений для записей элемента.

`labels`

Еще один массив с видимым текстом для каждого значения. Два массива должны иметь одинаковые размеры. (Если вы хотите использовать значения в качестве видимого текста, просто передайте функции один и тот же массив дважды.)

`default`

Указывает начальное значение элемента формы. Это должно быть скалярное значение для всех функций, кроме `scrolling_list()`. Эта функция будет написана так, чтобы обрабатывать списки с возможностью как единичного, так и множественного выбора (она будет использоваться в дальнейшем в рецепте 18.3), поэтому ее значение `default` может быть и скаляром, и массивом. Если умолчание не задано, передайте значение, которое не содержится в массиве `values`; обычно используется пустая строка.

У некоторых функций есть дополнительные аргументы, которые применяются только к определенным типам элементов:

- `vertical` применяется к группам переключателей. Предписывает расположить переключатели в вертикальном, а не горизонтальном порядке.

- Аргументы `size` и `multiple` применяются к прокручиваемым спискам: `size` указывает количество видимых элементов списка, а `multiple` устанавливается в значение «истина», если список допускает множественный выбор.

Мы рассмотрим реализацию некоторых функций формирования списков, а код всех таких функций хранится в каталоге *lib* дистрибутива `recipes`. Все они, подобно функциям модуля CGI.pm, автоматически выполняют HTML-кодирование для значений аргументов, встраиваемых в список.

В PHP можно написать функцию создания группы переключателей: `make_radio_group()` так:

```
function make_radio_group ($name, $values, $labels, $default, $vertical)
{
    if (!is_array ($values))
        return ("make_radio_group: values argument must be an array");
    if (!is_array ($labels))
        return ("make_radio_group: labels argument must be an array");
    if (count ($values) != count ($labels))
        return ("make_radio_group: value and label list size mismatch");
    $str = "";
    for ($i = 0; $i < count ($values); $i++)
    {
        # выбрать элемент, если он соответствует значению по умолчанию
        $checked = ($values[$i] == $default ? " checked=\"checked\"" : "");
        $str .= sprintf (
                "<input type=\"radio\" name=\"%s\" value=\"%s\"%s />%s",
                htmlspecialchars ($name),
                htmlspecialchars ($values[$i]),
                $checked,
                htmlspecialchars ($labels[$i]));
        if ($vertical)
            $str .= "<br />";   # вывести элементы вертикально
        $str .= "\n";
    }
    return ($str);
}
```

Функция выполняет предварительную проверку аргументов, затем создает элемент формы в виде строки, которую и возвращает. Чтобы использовать функцию для представления цветов фигурок коров, вызовем ее после извлечения записей из таблицы `cow_color`:

```
$values = array ();
$query = "SELECT color FROM cow_color ORDER BY color";
$result_id = mysql_query ($query, $conn_id);
if ($result_id)
{
    while (list ($color) = mysql_fetch_row ($result_id))
        $values[] = $color;
    mysql_free_result ($result_id);
}
print (make_radio_group ("color", $values, $values, "", TRUE));
```

Массив $values передается функции дважды, так как он используется и для значений, и для видимого текста.

Если вы хотите вывести всплывающее меню, используйте другую функцию:

```
function make_popup_menu ($name, $values, $labels, $default)
{
    if (!is_array ($values))
        return ("make_popup_menu: values argument must be an array");
    if (!is_array ($labels))
        return ("make_popup_menu: labels argument must be an array");
    if (count ($values) != count ($labels))
        return ("make_popup_menu: value and label list size mismatch");
    $str = "";
    for ($i = 0; $i < count ($values); $i++)
    {
        # выбрать элемент, если он соответствует значению по умолчанию
        $checked = ($values[$i] == $default ? " selected=\"selected\"" : "");
        $str .= sprintf (
                "<option value=\"%s\"%s>%s</option>\n",
                htmlspecialchars ($values[$i]),
                $checked,
                htmlspecialchars ($labels[$i]));
    }
    $str = sprintf (
                "<select name=\"%s\">\n%s</select>\n",
                htmlspecialchars ($name),
                $str);
    return ($str);
}
```

У функции make_popup_menu() нет параметра $vertical, но в остальном она вызывается точно так же, как make_radio_group():

```
print (make_popup_menu ("color", $values, $values, ""));
```

Функция make_scrolling_list() похожа на make_popup_menu(), поэтому я не привожу здесь ее реализацию. Если вы хотите вызвать ее для формирования списка с возможностью выбора единственного элемента, передайте ей те же аргументы, что и make_popup_menu(), укажите, сколько строк должно отображаться одновременно и добавьте аргумент multiple со значением FALSE:

```
print (make_scrolling_list ("color", $values, $values, "", 3, FALSE));
```

Список штатов использует различные наборы данных для значений и видимого текста. Извлечем их так:

```
$values = array ();
$labels = array ();
$query = "SELECT abbrev, name FROM states ORDER BY name";
$result_id = mysql_query ($query, $conn_id);
if ($result_id)
{
```

```
    while (list ($abbrev, $name) = mysql_fetch_row ($result_id))
    {
        $values[] = $abbrev;
        $labels[] = $name;
    }
    mysql_free_result ($result_id);
}
```

Затем используем видимый текст и значения для формирования необходимого списка:

```
print (make_popup_menu ("state", $values, $labels, ""));

print (make_scrolling_list ("state", $values, $labels, "", 6, FALSE));
```

Реализация функций на Python аналогична версиям на PHP. Например, функция `make_popup_menu()` **выглядит так:**

```
def make_popup_menu (name, values, labels, default):
    if type (values) not in (types.ListType, types.TupleType):
        return ("make_popup_group: values argument must be a list")
    if type (labels) not in (types.ListType, types.TupleType):
        return ("make_popup_group: labels argument must be a list")
    if len (values) != len (labels):
        return ("make_popup_group: value and label list size mismatch")
    str = ""
    for i in range (len (values)):
        value = values[i]
        label = labels[i]
        # проверить, являются ли значения и видимый текст строками
        if type (value) is not types.StringType:
            value = `value`
        if type (label) is not types.StringType:
            label = `label`
        # выбрать элемент, если он соответствует значению по умолчанию
        if type (default) is not types.StringType:
            default = `default`
        if value == default:
            checked = " selected=\"selected\""
        else:
            checked = ""
        str = str + \
                "<option value=\"%s\"%s>%s</option>\n" \
                    % (cgi.escape (value, 1),
                        checked,
                        cgi.escape (label, 1))

    if type (name) is not types.StringType:
        name = `name`
    str = "<select name=\"%s\">\n%s</select>\n" \
                % (cgi.escape (name, 1), str)
    return (str)
```

Для представления в форме цветов коров извлекаем их из таблицы:

```
values = []
query = "SELECT color FROM cow_color ORDER BY color"
cursor = conn.cursor ()
cursor.execute (query)
for (color, ) in cursor.fetchall ():
    values.append (color)
cursor.close ()
```

Затем преобразуем список в элемент формы:

```
print make_radio_group ("color", values, values, "", 1)

print make_popup_menu ("color", values, values, "")

print make_scrolling_list ("color", values, values, "", 3, 0)
```

Для вывода списка штатов извлекаем их названия и аббревиатуры:

```
values = []
labels = []
query = "SELECT abbrev, name FROM states ORDER BY name"
cursor = conn.cursor ()
cursor.execute (query)
for (abbrev, name) in cursor.fetchall ():
    values.append (abbrev)
    labels.append (name)
cursor.close ()
```

Затем передаем соответствующей функции:

```
print make_popup_menu ("state", values, labels, "")

print make_scrolling_list ("state", values, labels, "", 6, 0)
```

Есть одна вещь, которую функции на Python делают, а их аналоги на PHP – нет: явное преобразование значений аргументов, встраиваемых в список в строковом формате. Это необходимо, так как метод `cgi.escape()` порождает исключение, если вы пытаетесь использовать его для HTML-кодирования нестрокового значения.

Пока что мы говорили об извлечении строк из таблиц `cow_color` и `states` и их преобразовании в элементы формы. Для онлайн-приложения для заказа фигурок коров необходим еще один элемент формы – поле, указывающее размер статуэтки. Разрешенные значения этого поля определяются столбцом `size` таблицы `cow_order`. Этот столбец относится к типу `ENUM`, так что для получения допустимых значений элемента формы следует получить определение столбца и выделить из него нужную информацию.

К счастью, большая часть работы по решению этой задачи уже была проделана в рецепте 9.6, где создавались функции для вывода метаданных столбцов `ENUM` и `SET`. Например, в Perl для получения метаданных столбца `size` вызовите функцию `get_enumorset_info()`:

```
my $size_info = get_enumorset_info ($dbh, "cow_order", "size");
```

Результирующее значение $size_info — это ссылка на хеш, содержащий множество элементов, два из которых нас сейчас интересуют:

```
$size_info->{values}
$size_info->{default}
```

Член values представляет собой ссылку на разрешенные значения перечислимого типа, а default — это значение столбца по умолчанию. Формат информации позволяет непосредственно преобразовать ее в элемент формы, например в группу переключателей или всплывающее меню:

```
print radio_group (-name => "size",
                   -values => $size_info->{values},
                   -default => $size_info->{default},
                   -linebreak => 1);       # расположить кнопки вертикально

print popup_menu (-name => "size",
                  -values => $size_info->{values},
                  -default => $size_info->{default});
```

Значением по умолчанию является medium, это значение будет выбрано при открытии формы броузером.

Функция извлечения метаданных для PHP возвращает ассоциативный массив. Используем его для генерирования элементов формы на основе метаданных столбца size так:

```
$size_info = get_enumorset_info ($conn_id, "cow_order", "size");

print (make_radio_group ("size",
                         $size_info["values"],
                         $size_info["values"],
                         $size_info["default"],
                         TRUE));     # расположить элементы вертикально

print (make_popup_menu ("size",
                         $size_info["values"],
                         $size_info["values"],
                         $size_info["default"]));
```

Версия функции на Python возвращает словарь, который используется аналогично:

```
size_info = get_enumorset_info (conn, "cow_order", "size")

print make_radio_group ("size",
                        size_info["values"],
                        size_info["values"],
                        size_info["default"],
                        1)

print make_popup_menu ("size",
                       size_info["values"],
                       size_info["values"],
                       size_info["default"])
```

Если вы именно так используете значения ENUM для создания списковых элементов, то они выводятся в том порядке, в котором были указаны в определении столбца. В определении столбца size значения представлены в удобном для отображения порядке (small, medium, large), если же вас не устраивает порядок столбцов в определении, отсортируйте их.

Чтобы проиллюстрировать создание элементов формы для JSP-страниц на основе метаданных столбца, я буду использовать функцию, встраиваемую в страницу. Более удачным способом является создание в библиотеке тегов пользовательского действия (custom action), которое соответствует классу, возвращающему информацию, но создание пользовательских тегов выходит за рамки данной книги. Поэтому в примерах будут применяться следующие приемы:

- Используем теги JSTL для выполнения запроса SHOW COLUMNS с целью получения определения столбца ENUM, затем перемещаем определение в контекст страницы.
- Пишем функцию, извлекающую определение из контекста страницы, разбиваем его на массив отдельных значений и помещаем массив обратно в контекст страницы.
- Обращаемся к массиву при помощи итератора JSTL, который выводит каждое из значений как элемент списка. Каждое значение сравнивается со значением столбца по умолчанию и, в случае совпадения, значение помечается как первоначально выбранное.

Функция, извлекающая разрешенные значения из определения столбца ENUM или SET, называется getEnumOrSetValues(). Поместим ее в страницу JSP так:[1]

```
<%@ page import="java.util.*" %>
<%@ page import="org.apache.oro.text.perl.*" %>

<%!
// объявить метод класса для выделения значений ENUM/SET. typeDefAttr - имя атрибута
// контекста страницы, который содержит определение типа столбца
// valListAttr - имя атрибута контекста страницы, в который будет
// помещен список значений столбца

void getEnumOrSetValues (PageContext ctx,
                         String typeDefAttr,
                         String valListAttr)
{
    Perl5Util util = new Perl5Util ();
    String typeDef = ctx.getAttribute (typeDefAttr).toString ();
    // убрать начальное "enum(" и конечное ")", чтобы остался список
    // закавыченных значений, разделенных запятыми
    String qValStr = util.substitute ("s/^(enum|set)\\((.*)\\)$/$2/", typeDef);
```

[1] Функции getEnumOrSetValues() необходима библиотека регулярных выражений Jakarta ORO, которую можно получить на сайте Jakarta: *http://jakarta.apache.org*. Скопируйте файл JAR в каталог *common/lib* сервера Tomcat и перезапустите его, тогда библиотека будет доступна вашим JSP-страницам.

```
    List quotedVal = new ArrayList ();
    List unquotedVal = new ArrayList ();
    // разбить строку по запятым для формирования списка закавыченных значений
    util.split (quotedVal, "/,/", qValStr);
    for (int i = 0; i < quotedVal.size (); i++)
    {
        // убрать кавычки из каждого значения
        String s = quotedVal.get (i).toString ();
        s = util.substitute ("s/^'(.*)'$/$1/g", s);
        unquotedVal.add (s);
    }
    ctx.setAttribute (valListAttr, unquotedVal);
}

%>
```

Функция принимает три аргумента:

- Объект контекста страницы.
- Имя атрибута страницы, который содержит определение столбца. Это «вход» функции.
- Имя атрибута страницы, в который помещается результирующий массив разрешенных значений столбца. Это «выход» функции.

Для формирования спискового элемента на основе столбца `size` начнем с получения метаданных столбца: извлечем список значений столбца в переменную JSTL `values`, а значение по умолчанию – в переменную `default`:

```
<sql:query var="rs" dataSource="${conn}">
    SHOW COLUMNS FROM cow_order LIKE 'size'
</sql:query>
<c:set var="typeDef" scope="page" value="${rs.rowsByIndex[0][1]}" />
<% getEnumOrSetValues (pageContext, "typeDef", "values"); %>
<c:set var="default" scope="page" value="${rs.rowsByIndex[0][4]}" />
```

Затем используем список значений и значение по умолчанию для построения элемента формы. Например, создадим группу переключателей:

```
<c:forEach var="val" items="${values}">
    <input type="radio" name="size"
        value="<c:out value="${val}" />"
        <c:if test="${val == default}">checked="checked"</c:if>
    /><c:out value="${val}" /><br />
</c:forEach>
```

Или всплывающее меню:

```
<select name="size">
<c:forEach var="val" items="${values}">
    <option
        value="<c:out value="${val}" />"
        <c:if test="${val == default}">selected="selected"</c:if>
    >
    <c:out value="${val}" /></option>
```

```
</c:forEach>
</select>
```

Обсуждаемые методы формирования списков не связаны с какой-то определенной таблицей базы данных, поэтому могут использоваться при создании элементов формы для любых типов данных, а не только рассматриваемых нами для производства фигурок коров. Например, чтобы дать пользователю возможность выбрать таблицу в приложении администрирования базы данных, можно сформировать прокручиваемый список имен всех ее таблиц. Сценарий CGI.pm мог бы делать это так:

```
my $table_ref = $dbh->selectcol_arrayref ("SHOW TABLES");
print scrolling_list (-name => "table",
                      -values => $table_ref,
                      -size => 10);           # показывать 10 записей одновременно
```

Результаты запроса даже необязательно должны быть связаны с таблицами базы данных. Например, если вы хотите вывести на странице JSP список, содержащий запись для каждого из последних семи дней, то можете получить эти даты при помощи такого запроса:

```
<sql:query var="rs" dataSource="${conn}">
    SELECT
        DATE_SUB(CURDATE(),INTERVAL 5 DAY),
```

Не забывайте об HTML-кодировании всего содержимого списков формы

Функции формирования списковых элементов на PHP и Python, описанные в этом разделе, выполняют HTML-кодирование значений атрибутов тегов, образующих список, таких как атрибуты `name` и `value`. Кроме того, они кодируют видимый текст тегов. Я обратил внимание на то, что во многих опубликованных примерах формирования списков этого не делается, или же кодируется текст, но не значения. Это неправильно. Если значение или текст содержит специальный символ (например, `&` или `<`), то броузер неправильно интерпретирует их, и приложение будет работать некорректно. Необходимо также убедиться в том, что функция превращает двойные кавычки в объекты `&quot;` (или `&#34`), так как атрибуты тегов очень часто заключаются в двойные кавычки. Если не преобразовать двойную кавычку в значении атрибута, то образуется синтаксическая конструкция с символом двойной кавычки внутри строки, заключенной в двойные кавычки, что недопустимо.

Если вы используете модуль Perl CGI.pm или теги JSTL для генерирования элементов HTML формы, то кодированием за вас займутся другие. Функции генерирования форм CGI.pm автоматически выполняют кодирование. Использование тега JSTL `<c:out>` для записи значений атрибутов JSP-страниц приводит к выводу корректно кодированных значений.

```
        DATE_SUB(CURDATE(),INTERVAL 4 DAY),
        DATE_SUB(CURDATE(),INTERVAL 3 DAY),
        DATE_SUB(CURDATE(),INTERVAL 2 DAY),
        DATE_SUB(CURDATE(),INTERVAL 1 DAY),
        CURDATE()
</sql:query>
```

Затем использовать даты для формирования спискового элемента:

```
<c:set var="dateList" value="${rs.rowsByIndex[0]}" />
<c:forEach var="date" items="${dateList}">
    <input type="radio" name="date"
        value="<c:out value="${date}" />"
    /><c:out value="${date}" /><br />
</c:forEach>
```

(Конечно, если API обеспечивает простой способ вычисления дат, то может быть эффективнее выводить список дат с клиентской стороны, не отправляя запрос на сервер MySQL.)

18.3. Создание элементов формы с возможностью выбора нескольких значений

Задача

Форма должна содержать поле, которое предлагает пользователю ряд вариантов и разрешает выбор нескольких из них.

Решение

Используйте списковый элемент с возможностью выбора нескольких значений, такой как группа флажков (checkbox) или прокручиваемый список (scrolling list).

Обсуждение

Элементы формы с возможностью выбора нескольких значений позволяют выбирать любое количество предложенных вариантов, в том числе ни одного. В нашем примере, посвященном заказу фигурок коров, элемент с возможностью выбора нескольких значений представлен набором доступных аксессуаров. Столбец `accessory` таблицы `cow_order` относится к типу `SET`, поэтому его допустимые значения и значение по умолчанию могут быть получены следующим запросом:

```
mysql> SHOW COLUMNS FROM cow_order LIKE 'accessories'\G
*************************** 1. row ***************************
  Field: accessories
   Type: set('cow bell','horns','nose ring','tail ribbon')
   Null: YES
    Key:
```

```
Default: cow bell,horns
  Extra:
```

Такой набор элементов разумно представлять в виде группы флажков или прокручиваемого списка с возможностью множественного выбора. В любом случае необходимо изначально выбрать элементы `cow bell` и `horns`, так как каждый из них является значением столбца по умолчанию. Поговорим о синтаксисе этих элементов, затем рассмотрим их формирование в сценариях. (Материал этого раздела базируется на сведениях из рецепта 18.2, где были описаны переключатели, всплывающие меню и прокручиваемые списки с возможностью выбора единственного значения. Я буду считать, что вы уже ознакомились с тем рецептом.)

Флажки. Группа флажков похожа на группу переключателей тем, что она состоит из элементов `<input>`, имеющих одинаковые атрибуты `name`. Однако атрибут `type` содержит `checkbox`, а не `radio`, кроме того необходимо пометить как `checked` все те элементы группы, которые вы хотите видеть выбранными по умолчанию. Если ни один элемент не помечен, ни один не будет изначально выбран. Следующий набор флажков показывает аксессуары для фигурок коров, при этом по умолчанию выбраны два первых элемента:

```
<input type="checkbox" name="accessories" value="cow bell"
    checked="checked" />cow bell
<input type="checkbox" name="accessories" value="horns"
    checked="checked" />horns
<input type="checkbox" name="accessories" value="nose ring" />nose ring
<input type="checkbox" name="accessories" value="tail ribbon" />tail ribbon
```

Прокручиваемый список. Прокручиваемый список с возможностью выбора нескольких значений формируется аналогично списку с возможностью выбора единственного значения. Отличия состоят в том, что вы включаете атрибут `multiple` в открывающий тег `<select>`, а значение по умолчанию обрабатывается по-другому. Для списка с единичным выбором вы можете добавить `selected` не более чем к одному элементу, при этом по умолчанию в отсутствие явного установленного атрибута `selected` выбирается первое значение списка. Для списка с множественным выбором вы можете добавить атрибут `selected` для любого количества значений, в отсутствие же таких атрибутов ни один элемент по умолчанию не будет выбран.

Представим набор аксессуаров для коров в виде прокручиваемого списка с множественным выбором, где по умолчанию выбираются `cow bell` и `horns`:

```
<select name="accessories" size="3" multiple="multiple">
<option value="cow bell" selected="selected">cow bell</option>
<option value="horns" selected="selected">horns</option>
<option value="nose ring">nose ring</option>
<option value="tail ribbon">tail ribbon</option>
</select>
```

В CGI.pm-сценариях на Perl вы можете создавать группы флажков и прокручиваемые списки, вызывая функции `checkbox_group()` и `scrolling_list()` соответственно. Эти функции принимают аргументы `name`, `values`, `labels` и `default`,

как и их родственницы с возможностью выбора единственного значения. Но так как изначально может быть выбрано несколько элементов, CGI.pm разрешает указывать аргумент `default` в виде скаляра или в виде ссылки на массив значений. (Имя аргумента `defaults` является синонимом `default`.)

Для получения списка разрешенных значений столбца `SET` можно сделать то же самое, что для столбцов `ENUM` в рецепте 18.2 – то есть вызвать функцию, возвращающие метаданные столбца:

```
my $acc_info = get_enumorset_info ($dbh, "cow_order", "accessories");
```

Однако значение по умолчанию для столбца `SET` имеет не совсем тот вид, который подходил бы для генерирования элемента формы. MySQL представляет значения `SET` по умолчанию в виде списка элементов, разделенных запятыми. Например, значение по умолчанию для столбца `accessories` – это `cow bell,horns`. Функции CGI.pm рассчитывают не на такой формат, поэтому необходимо разделить значение по умолчанию по запятым, чтобы получить массив. Следующее выражение показывает, как это сделать с учетом того, что значением по умолчанию может быть `undef` (`NULL`):

```
my @acc_def = (defined ($acc_info->{default})
                    ? split (/,/, $acc_info->{default})
                    : () );
```

После разделения значения по умолчанию на части передайте полученный массив по ссылке той функции формирования списка, которую вы планируете использовать:

```
print checkbox_group (-name => "accessories",
                      -values => $acc_info->{values},
                      -default => \@acc_def,
                      -linebreak => 1);       # выводить кнопки вертикально

print scrolling_list (-name => "accessories",
                      -values => $acc_info->{values},
                      -default => \@acc_def,
                      -size => 3,             # показывать 3 значения одновременно
                     -multiple => 1);        # создать список с множественным выбором
```

Если вы так используете значения `SET` для создания элементов списка, то значения будут отображаться в том порядке, в котором они перечислены в определении столбца. Если этот порядок вас не устраивает, отсортируйте значения соответствующим образом.

Для PHP и Python можно создать функции, формирующие элементы с множественным выбором. Формат их вызова будет таким:

```
make_checkbox_group (name, values, labels, default, vertical)
make_scrolling_list (name, values, labels, default, size, multiple)
```

Аргументы `name`, `values` и `labels` этих функций аналогичны аргументам функций PHP и Python для единичного выбора, описанных в рецепте 18.2. Функция `make_checkbox_group()` принимает аргумент `vertical`, который (если установлен в значение «истина») указывает на необходимость вертикального, а не

горизонтального расположения элементов. Функция `make_scrolling_list()` **уже была описана в рецепте 18.2 при формировании списков единичного выбора. Укажите для аргумента** `multiple` **значение «истина», чтобы использовать функцию для вывода списка с множественным выбором. Для обеих функций аргумент** `default` **может быть массивом значений, если изначально необходимо выбрать несколько элементов.**

Функция `make_checkbox_group()` **выглядит так (рассматриваем версию на Python; для PHP все аналогично):**

```
def make_checkbox_group (name, values, labels, default, vertical):
    if type (values) not in (types.ListType, types.TupleType):
        return ("make_checkbox_group: values argument must be a list")
    if type (labels) not in (types.ListType, types.TupleType):
        return ("make_checkbox_group: labels argument must be a list")
    if len (values) != len (labels):
        return ("make_checkbox_group: value and label list size mismatch")
    if type (default) not in (types.ListType, types.TupleType):
        default = [ default ]       # преобразовать скаляр в список
    str = ""
    for i in range (len (values)):
        value = values[i]
        label = labels[i]
        # проверить, являются ли значение и видимый текст строками
        if type (value) is not types.StringType:
            value = `value`
        if type (label) is not types.StringType:
            label = `label`
        # выбрать элемент, если он совпадает с одним из значений по умолчанию
        checked = ""
        for d in default:
            if type (d) is not types.StringType:
                d = `d`
            if value == d:
                checked = " checked=\"checked\""
                break
        if type (name) is not types.StringType:
            name = `name`
        str = str + \
                "<input type=\"checkbox\" name=\"%s\" value=\"%s\"%s />%s" \
                    % (cgi.escape (name, 1),
                        cgi.escape (value, 1),
                        checked,
                        cgi.escape (label, 1))
        if vertical:
            str = str + "<br />"    # вертикальный вывод элементов
        str = str + "\n"
    return (str)
```

Извлекаем информацию об аксессуарах для фигурок коров и представляем их флажками:

```
import re            # требуется re.split()

acc_info = get_enumorset_info (conn, "cow_order", "accessories")
if acc_info["default"] == None:
    acc_def = ""
else:
    acc_def = re.split (",", acc_info["default"])

print make_checkbox_group ("accessories",
                           acc_info["values"],
                           acc_info["values"],
                           acc_def,
                           1)          # вертикальный вывод элементов
```

Для формирования не флажков, а прокручиваемого списка, вызываем `make_scrolling_list()`:

```
print make_scrolling_list ("accessories",
                           acc_info["values"],
                           acc_info["values"],
                           acc_def,
                           3,          # показать 3 значения одновременно
                           1)          # создать список с множественным выбором
```

В PHP вы извлекаете информацию об аксессуарах, затем формируете флажки или прокручиваемый список так:

```
$acc_info = get_enumorset_info ($conn_id, "cow_order", "accessories");
$acc_def = explode (",", $acc_info["default"]);

print (make_checkbox_group ("accessories[]",
                           $acc_info["values"],
                           $acc_info["values"],
                           $acc_def,
                           TRUE));     # вертикальный вывод элементов

print (make_scrolling_list ("accessories[]",
                           $acc_info["values"],
                           $acc_info["values"],
                           $acc_def,
                           3,          # показывать 3 значения одновременно
                           TRUE));     # создать список с множественным выбором
```

Обратите внимание на то, что в примерах на PHP название поля указывается как `accessories[]`, а не `accessories`. Если вы хотите разрешить полю иметь несколько значений в PHP, необходимо добавить после его имени `[]`. Если опустить `[]`, то пользователь при заполнении формы сможет отметить несколько вариантов, но PHP вернет в сценарий только один из них. Мы еще столкнемся с этой проблемой при изучении обработки содержимого полученных от пользователя форм в рецепте 18.5.

Что касается страниц JSP, использованная ранее для получения списка значений столбца `size` (типа `ENUM`) функция `getEnumOrSetValues()` может применяться и для столбца `accessory` (типа `SET`). Определение столбца и значение по умолчанию содержатся во втором и пятом столбцах запроса `SHOW COLUMNS`, ко-

торый возвращает информацию о столбце accessory. Выполните запрос, преобразуйте определение типа в список значений values и поместите значение по умолчанию в defList:

```
<sql:query var="rs" dataSource="${conn}">
    SHOW COLUMNS FROM cow_order LIKE 'accessories'
</sql:query>
<c:set var="typeDef" scope="page" value="${rs.rowsByIndex[0][1]}" />
<% getEnumOrSetValues (pageContext, "typeDef", "values"); %>
<c:set var="defList" scope="page" value="${rs.rowsByIndex[0][4]}" />
```

Для столбца типа SET defList может содержать несколько значений, разделенных запятыми. Специальная обработка не требуется, тег JSTL <c:forEach> умеет обрабатывать такие строки в цикле, так что значения по умолчанию для флажков можно инициализировать так:

```
<c:forEach var="val" items="${values}">
    <input type="checkbox" name="accessories"
        value="<c:out value="${val}" />"
        <c:forEach var="default" items="${defList}">
            <c:if test="${val == default}">checked="checked"</c:if>
        </c:forEach>
    /><c:out value="${val}" /><br />
</c:forEach>
```

Формируем прокручиваемый список с множественным выбором:

```
<select name="accessories" size="3" multiple="multiple">
<c:forEach var="val" items="${values}">
    <option
        value="<c:out value="${val}" />"
        <c:forEach var="default" items="${defList}">
            <c:if test="${val == default}">selected="selected"</c:if>
        </c:forEach>
    >
    <c:out value="${val}" /></option>
</c:forEach>
</select>
```

18.4. Загрузка в форму записи базы данных

Задача

Вы хотите вывести форму, используя в качестве начальных значений содержимое записей базы данных. Это позволит вам представить форму для редактирования записей.

Решение

Создавайте форму как обычно, только вместо использования ее обычных значений по умолчанию установите элементы формы в значения столбцов записи базы данных.

Обсуждение

Рассматриваемые ранее примеры создания полей формы или не задавали значений по умолчанию, или использовали значение по умолчанию из определения столбца ENUM или SET как умолчание для поля. Именно так обычно и создаются «пустые» формы, которые должен заполнить пользователь. Однако если речь идет о приложении, предлагающем веб-интерфейс для редактирования записи, то вы, вероятно, захотите, чтобы форма при открытии была заполнена содержимым существующей записи базы данных. В этом разделе рассказано о том, как это сделать.

Примеры этого раздела показывают, как создать форму для редактирования записей таблицы cow_order. Обычно пользователю предлагается выбрать, какую запись он будет редактировать. Мы же для простоты будем считать, что используется запись со значением id, равным 1, которая содержит следующее:

```
mysql> SELECT * FROM cow_order WHERE id = 1\G
*************************** 1. row ***************************
         id: 1
      color: Black & White
       size: large
accessories: cow bell,nose ring
  cust_name: Farmer Brown
cust_street: 123 Elm St.
  cust_city: Katy
 cust_state: TX
```

При генерировании формы с содержимым, соответствующим записи базы данных, используйте значения полей для установки значений по умолчанию в компонентах формы:

- Для элементов <input>, таких как переключатели и флажки, добавьте атрибут checked каждому элементу списка, совпадающему со значением столбца.
- Для элементов <select>, таких как всплывающие меню и прокручиваемые списки, добавьте атрибут checked каждому элементу списка, совпадающему со значением столбца.
- Для текстовых полей, представленных как элементы <input> типа text, установите атрибут value равным значению соответствующего столбца. Например, 60-символьное поле для cust_name первоначально может быть установлено в значение Farmer Brown:

  ```
  <input type="text" name="cust_name" value="Farmer Brown" size="60" />
  ```

  Для представления элемента <textarea> установите его тело равным значению столбца. Создадим поле из трех 40-символьных строк:

  ```
  <textarea name="cust_name" cols="40" rows="3">
  Farmer Brown
  </textarea>
  ```

- При редактировании записей удобно включать в форму уникальное значение, которое бы указывало, какую запись представляет содержимое формы, переданной пользователем. Можно сделать это при помощи скрытого поля – его значение не выводится для пользователя, но броузер возвращает его вместе со значениями всех остальных полей. Наша тестовая запись содержит столбец id со значением 1, поэтому скрытое поле может выглядеть так:

```
<input type="hidden" name="id" value="1" />
```

Следующий пример показывает, как вывести форму, в которой столбец id представлен как скрытое поле, color – как всплывающее меню, size – как группа переключателей, а accessories – как группа флажков. Информация для пользователя отображается в текстовых полях ввода, за исключением столбца cust_state, выводимого в виде прокручиваемого списка с единичным выбором. Конечно, вы можете выбрать и другие варианты, например, показывать размеры (size) как всплывающее меню, а не как группу переключателей.

Сценарии примеров этого раздела называются *edit_cow.pl*, *edit_cow.jsp* и т. д.

Рассмотрим процесс загрузки записи таблицы cow_table в форму редактирования при помощи сценария на основе CGI.pm.

1. Извлекаем значения столбцов для той записи, которую нужно загрузить в форму:

```
my $id = 1;          # выбрать запись номер 1
my ($color, $size, $accessories,
    $cust_name, $cust_street, $cust_city, $cust_state) =
                $dbh->selectrow_array (
                        "SELECT
                            color, size, accessories,
                            cust_name, cust_street, cust_city, cust_state
                        FROM cow_order WHERE id = ?",
                        undef, $id);
```

2. Начинаем создавать форму:

```
print start_form (-action => url ());
```

3. Формируем скрытое поле, содержащее значение id, которое однозначно идентифицирует запись cow_order:

```
print hidden (-name => "id", -value => $id, -override => 1);
```

Аргумент override заставляет CGI.pm использовать значение, указанное в аргументе value, как скрытое значение поля. Это необходимо потому, что обычно CGI.pm пытается использовать значения переменных окружения сценария для инициализации полей формы, даже если значения передаются в вызовах генерирования полей. (CGI.pm пытается тем самым упростить повторное отображение формы с уже введенными пользователем значениями. Например, если вы обнаружите, что форма была запол-

нена неверно, то можете повторно запросить ее и все исправить. Чтобы удостовериться в том, что элемент формы содержит указанное вами значение, необходимо запретить такое поведение.)

4. Создаем поля, описывающие параметры фигурок коров. Эти поля формируются так, как описывалось в рецептах 18.2 и 18.3, только значения по умолчанию берутся из содержимого записи 1. Код представляет `color` как всплывающее меню, `size` – как группу переключателей, а `accessories` – как группу флажков. Обратите внимание на то, что он разбивает значение `accessories` по запятым для создания массива значений, так как значение столбца может содержать несколько аксессуаров:

```
my $color_ref = $dbh->selectcol_arrayref (
                    "SELECT color FROM cow_color ORDER BY color");

print br (), "Cow color:", br ();
print popup_menu (-name => "color",
                    -values => $color_ref,
                    -default => $color,
                    -override => 1);

my $size_info = get_enumorset_info ($dbh, "cow_order", "size");

print br (), "Cow figurine size:", br ();
print radio_group (-name => "size",
                    -values => $size_info->{values},
                    -default => $size,
                    -override => 1,
                    -linebreak => 1);

my $acc_info = get_enumorset_info ($dbh, "cow_order", "accessories");
my @acc_val = (defined ($accessories)
                    ? split (/,/, $accessories)
                    : () );

print br (), "Cow accessory items:", br ();
print checkbox_group (-name => "accessories",
                    -values => $acc_info->{values},
                    -default => \@acc_val,
                    -override => 1,
                    -linebreak => 1);
```

5. Создаем поля с информацией о клиенте. Они будут текстовыми полями ввода, за исключением штата, который выводится как прокручиваемый список с единичным выбором:

```
print br (), "Customer name:", br ();
print textfield (-name => "cust_name",
                    -value => $cust_name,
                    -override => 1,
                    -size => 60);

print br (), "Customer street address:", br ();
print textfield (-name => "cust_street",
```

```
                    -value => $cust_street,
                    -override => 1,
                    -size => 60);

print br (), "Customer city:", br ();
print textfield (-name => "cust_city",
                    -value => $cust_city,
                    -override => 1,
                    -size => 60);

my @state_values;
my %state_labels;
my $sth = $dbh->prepare ("SELECT abbrev, name FROM states ORDER BY name");
$sth->execute ();
while (my ($abbrev, $name) = $sth->fetchrow_array ())
{
    push (@state_values, $abbrev);  # сохранить каждое значение в массиве
    $state_labels{$abbrev} = $name; # сопоставить каждому значению название
}

print br (), "Customer state:", br ();
print scrolling_list (-name => "cust_state",
                    -values => \@state_values,
                    -labels => \%state_labels,
                    -default => $cust_state,
                    -override => 1,
                    -size => 6);            # показывать 6 значений одновременно
```

6. Создаем кнопку отправки формы и завершаем форму:

```
print br (),
        submit (-name => "choice", -value => "Submit Form"),
        end_form ();
```

Процедура для других API будет такой же. Например, в странице JSP вы можете извлечь запись для редактирования и присвоить значения ее полей скалярным переменным так:

```
<c:set var="id" value="1" />
<sql:query var="rs" dataSource="${conn}">
    SELECT
        id, color, size, accessories,
        cust_name, cust_street, cust_city, cust_state
    FROM cow_order WHERE id = ?
    <sql:param value="${id}" />
</sql:query>

<c:set var="row" value="${rs.rows[0]}" />
<c:set var="id" value="${row.id}" />
<c:set var="color" value="${row.color}" />
<c:set var="size" value="${row.size}" />
<c:set var="accessories" value="${row.accessories}" />
<c:set var="cust_name" value="${row.cust_name}" />
<c:set var="cust_street" value="${row.cust_street}" />
```

```
<c:set var="cust_city" value="${row.cust_city}" />
<c:set var="cust_state" value="${row.cust_state}" />
```

Затем использовать эти значения для инициализации различных элементов формы, таких как:

- Скрытое поля для значения идентификатора:

```
<input type="hidden" name="id" value="<c:out value="${id}" />" />
```

- Всплывающее меню `color`:

```
<sql:query var="rs" dataSource="${conn}">
    SELECT color FROM cow_color ORDER BY color
</sql:query>
<br />Cow color:<br />
<select name="color">
<c:forEach var="row" items="${rs.rows}">
    <option
        value="<c:out value="${row.color}" />"
        <c:if test="${row.color == color}">selected="selected"</c:if>
    ><c:out value="${row.color}" /></option>
</c:forEach>
</select>
```

- Текстовое поле `cust_name`:

```
<br />Customer name:<br />
<input type="text" name="cust_name"
    value="<c:out value="${cust_name}" />"
    size="60" />
```

В PHP и Python создавайте форму, используя функции из рецептов 18.2 и 18.3 (см. сценарии *cow_edit.php* и *cow_edit.py*).

18.5. Получение входных данных через Web

Задача

Вы хотите получить значения входных параметров, переданных в составе формы или указанных в конце URL.

Решение

Каждый API предлагает средство доступа к именам и значениям входных параметров в среде исполнения веб-сценария.

Обсуждение

В предыдущих разделах обсуждалось извлечение информации из MySQL и ее использование для формирования различных видов вывода: статического текста, гиперссылок и элементов формы. В этом разделе мы рассмотрим обратную задачу – получение входных данных через Web. Приемы, обсуждае-

мые в этом разделе, можно использовать, например, для извлечения содержимого формы, отправленной пользователем. Интерпретируем полученные значения как ключевые слова поиска, затем выполняем запрос к каталогу товаров и показываем пользователю результат. В данном случае Web используется для получения информации, по которой мы определяем, что именно интересует заказчика. По этой информации строим запрос и выводим результаты. Если форма представляет собой опрос, подписку на список рассылки или голосование, то можно просто сохранить значения, используя полученные данные для создания новой записи в базе данных (или для обновления существующей записи).

Сценарий, получающий входные данные через Web и использующий их для взаимодействия с MySQL, обычно обрабатывает информацию поэтапно:

1. Извлекает входные данные из окружения. При получении запроса, содержащего входные параметры, веб-сервер помещает их в окружение сценария, обрабатывающего запрос, а сценарий запрашивает параметры у окружения. Может потребоваться декодирование специальных символов в параметрах для выявления истинных значений, переданных клиентом, если ваш API не делает этого сам при извлечении. (Например, может потребоваться преобразовать `%20` в пробел.)
2. Проверяет корректность входных данных. Нельзя надеяться на то, что пользователи отправляют только разрешенные данные, поэтому всегда стоит проверить разумность входных параметров. Например, если предполагается, что пользователь должен ввести в поле число, необходимо проверить, действительно ли введенное значение является числовым. Если форма содержит всплывающее меню, созданнное на основе столбца `ENUM`, то ожидается, что полученное от пользователя значения должно быть одним из элементов списка. Но быть уверенным в этом можно лишь после проверки. Иначе возникает риск ввода в базу данных некорректных значений.
3. Создает запрос на основе входных данных. Обычно входные параметры используются для добавления записи в базу данных или извлечения информации из базы для их отображения клиенту. В любом случае ввод используется для создания запроса и отправки его серверу MySQL. Запрос на основе входных данных следует формировать осторожно, используя экранирование специальных символов во избежание создания синтаксически неверных и опасных предложений SQL.

Оставшаяся часть раздела посвящена изучению первого этапа обработки ввода. Второй и третий этапы рассматриваются в рецептах 18.6 и 18.7. Первый этап (извлечение входных данных из окружения) имеет весьма отдаленное отношение к MySQL, но описать его необходимо, так как это единственный способ получения информации, используемой на последующих этапах обработки.

Входные данные могут быть получены через Web различными способами, два из которых встречаются чаще:

- Как составляющая запроса GET, тогда входные параметры добавляются в конец URL. Например, следующий URL вызывает сценарий на PHP *price_quote.php* и указывает параметры item и quantity со значениями D-0214 и 60:

  http://apache.snake.net/mcb/price_quote.php?item=D-0214&quantity=60

  Такие запросы обычно получаются, когда пользователь выбирает гиперссылку или отправляет форму, у которой в теге <form> указано method="GET". Список параметров в URL начинается с ? и состоит из пар *имя=значение*, разделенных символами ; или &. (Можно поместить информацию и в середину URL, но здесь я не буду об этом рассказывать.)
- Как составляющая запроса POST при отправке формы, у которой в теге <form> указано method="POST". Содержимое формы для запроса POST отправляется как входные параметры в теле запроса, а не в конце URL.

Вам могут встретиться и другие виды ввода, например, загружаемые файлы. Они отправляются при помощи запросов POST, но внутри специальной формы, речь о которой пойдет в рецепте 18.8.

Когда вы собираете входные параметры для веб-сценария, может понадобиться информация о том, каким способом они были получены. (Некоторые API различают ввод, полученный при помощи GET и POST, некоторые – нет.) Однако после того, как информация извлечена, метод запроса уже не имеет значения. На этапах проверки корректности и построения запроса уже не важно, посредством какого метода были получены параметры.

Дистрибутив recipes содержит ряд сценариев обработки входных параметров в каталоге *apache/params* (*tomcat/mcb* для JSP). Каждый сценарий позволяет отправлять запросы GET и POST и показывает, как извлекать и выводить значения отправленных параметров. Изучите эти сценарии, чтобы понять, как методы извлечения параметров используются в разных API. Вызываемые сценариями функции хранятся в библиотечных модулях каталога *lib* дистрибутива.

Правила извлечения Web-ввода

Для получения входных параметров, переданных в сценарий, необходимо изучить возможности вашего API, чтобы понимать, что он может вам предложить, а какие операции вы должны проделать самостоятельно. Например, вы должны знать ответы на следующие вопросы:

- Как определить, какие параметры доступны?
- Как извлечь значение параметра из окружения?
- Полученные значения – это реальные значения, переданные клиентом, или их необходимо раскодировать?
- Как обрабатываются параметры с множественными значениями (например, если выбрано несколько элементов группы флажков)?
- Какой символ API предполагает увидеть в качестве разделителя параметров, передаваемых в URL? В одних API это может быть &, в других – ;.

Последнее предпочтительнее, так как символ ; , в отличие от &, не является специальным для HTML, но многие броузеры и другие пользовательские агенты разделяют параметры при помощи &. Если вы создаете URL в сценарии, включающем параметры в конец адреса, используйте символ разделителя параметров, который будет понятен принимающему сценарию.

Perl

Модуль Perl CGI.pm обеспечивает доступ сценариев к входным параметрам при помощи функции param(). Эта функция упрощает жизнь создателю сценария при работе с параметрами, переданными любым из методов GET и POST. Если форма, содержащая параметры id и name, была передана при помощи POST, вы можете обрабатывать ее так же, как если бы параметры были указаны в конце URL и переданы в GET. Кроме того, не придется заниматься декодированием, функция param() все сделает сама.

Для получения списка имен всех доступных параметров вызовите param() без аргументов:

```
@names = param ();
```

Чтобы получить значение определенного параметра, передайте его имя в param():

```
$id = param ("id");
@options = param ("options");
```

В скалярном контексте param() возвращает значение параметра, если оно одно, первое значение, если у параметра их несколько, и undef – если параметр недоступен. В контексте массива param() возвращает список, содержащий все значения параметра, или пустой список, если параметр недоступен.

Параметр с указанным именем может быть недоступен, если поле формы с таким именем оставлено незаполненным или такого поля нет вообще. Имейте в виду, что значение параметра может быть указано как пустое. Для полноты можно проверять обе возможности. Например, чтобы проверить параметр age и присвоить ему значение по умолчанию unknown, если параметр отсутствует или пуст, сделайте следующее:

```
$age = param ("age");
$age = "unknown" if !defined ($age) || $age eq "";
```

CGI.pm воспринимает и ; , и & как символы-разделители параметров в URL.

PHP

В PHP к входным параметрам можно обращаться разными способами в зависимости от версии PHP и параметров конфигурации:

- Если установлен параметр register_globals, то входные параметры присваиваются глобальным переменным с такими же именами. Тогда к значению поля id можно обращаться как к переменной $id независимо от того, отправлен запрос через GET или POST.

- Если установлен параметр track_vars, то входные параметры доступны в массивах $HTTP_GET_VARS и $HTTP_POST_VARS. Например, если в форме есть поле id, можно обратиться к его значению при помощи $HTTP_GET_VARS["id"] или $HTTP_POST_VARS["id"] в зависимости от того, каким методом была отправлена форма. $HTTP_GET_VARS и $HTTP_POST_VARS необходимо объявить, используя ключевое слово global, чтобы они были доступны в неглобальных сценариях, например, внутри функции.
- Начиная с версии PHP 4.1 входные параметры доступны в массивах $_GET и $_POST. Они аналогичны $HTTP_GET_VARS и $HTTP_POST_VARS, только являются «суперглобальными» в том смысле, что они автоматически доступны в любом контексте. (Например, нет необходимости в объявлении $_GET и $_POST при помощи global внутри функций.) В настоящее время массивы $_GET и $_POST являются предпочтительным средством получения входных параметров.

Параметры track_vars и register_globals могут быть указаны при сборке или заданы в файле *php.ini*. Начиная с версии PHP 4.0.3 параметр track_vars всегда установлен, и я подозреваю, что и в ранних версиях он чаще всего был включен. Поэтому будем считать, что в вашей версии PHP track_vars установлен.

Параметр register_globals обеспечивает удобный доступ к входным параметрам через глобальные переменные, но PHP-разработчики рекомендуют не использовать его из соображений безопасности. Почему? Предположим, что вы пишете сценарий, который запрашивает у пользователя пароль, представленный переменной $password. Проверить пароль в сценарии можно так:

```
if (check_password ($password))
    $password_is_ok = 1;
```

Если пароль подходит, то сценарий устанавливает $password_is_ok в 1. В противном случае $password_is_ok остается неустановленным (аналог логического значения «ложь»). Но предположим, что установлен параметр register_variables, и кто-то вызывает сценарий так:

http://your.host.com/chkpass.php?password_is_ok=1

В этом случае PHP видит, что параметр password_is_ok установлен в 1, и устанавливает переменную $password_is_ok в 1. В результате при исполнении сценария $password_is_ok равна 1 независимо от того, какой пароль был введен или даже пароль вовсе *не был* введен! Проблема register_globals в том, что сторонние пользователи могут задавать значения по умолчанию для глобальных переменных ваших сценариев. Одним из решений может стать отключение register_globals, тогда за значениями входных параметров придется обращаться к глобальным массивам ($_GET, $_POST). Если вы не хотите этого делать, не рассчитывайте на то, что переменные PHP не имеют начальных значений. Лучше явно задать для них какие-то исходные значения, если конечно вы не хотите, чтобы глобальная переменная устанавливалась в значение входного параметра. Чтобы гарантировать присвоение значения переменной $password_is_ok вне зависимости от результата проверки, код проверки пароля следует переписать так:

```
$password_is_ok = 0;
if (check_password ($password))
    $password_is_ok = 1;
```

Сценарии на PHP в этой книге не полагаются на `register_globals` и получают входные параметры из глобальных массивов.

Извлечение параметров в PHP осложняется возможной необходимостью декодирования в зависимости от значения параметра конфигурации `magic_quotes_gpc`. Если магические кавычки разрешены, то все символы кавычек, обратного слэша и NUL в значениях входных параметров экранируются обратным слэшем. Так вы можете «проскочить» один этап, извлекая значения и используя их непосредственно в строке запроса. Однако такая возможность удобна только в том случае, если вы собираетесь использовать веб-ввод в запросе без предварительной обработки и проверки корректности, а это весьма опасно. Необходимо сначала проверить ввод, так или иначе избавившись ото всех символов обратного слэша. То есть разрешать магические кавычки особого смысла нет.

Извлечение ввода в сценариях на PHP становится интересной задачей: входные параметры могут поступать из различных источников и могут содержать или не содержать дополнительные символы обратного слэша. Если вы можете управлять сервером и устанавливать значения параметров конфигурации, то, конечно же, можете создавать сценарии на основе этих параметров. Но если вы не управляете сервером или пишете сценарии, которые будут работать на нескольких компьютерах, то заранее неизвестно, какой будет конфигурация. К счастью, немного постаравшись, вы сможете написать достаточно универсальный код извлечения параметров, корректно работающий при весьма небольших допущениях относительно вашего PHP-окружения. Рассмотрим функцию `get_param_val()`, которая принимает в качестве аргумента имя параметра и возвращает соответствующее значение. Если параметр недоступен, функция возвращает неустановленное (unset) значение:

```
function get_param_val ($name)
{
global $HTTP_GET_VARS, $HTTP_POST_VARS;

    unset ($val);
    if (isset ($_GET[$name]))
        $val = $_GET[$name];
    else if (isset ($_POST[$name]))
        $val = $_POST[$name];
    else if (isset ($HTTP_GET_VARS[$name]))
        $val = $HTTP_GET_VARS[$name];
    else if (isset ($HTTP_POST_VARS[$name]))
        $val = $HTTP_POST_VARS[$name];
    if (isset ($val) && get_magic_quotes_gpc ())
        $val = strip_slash_helper ($val);
    return (@$val);
}
```

Если вы хотите применить функцию для получения единственного значения параметра id, вызовите ее так:

```
$id = get_param_val ("id");
```

Можно проверить $id, чтобы определить, присутствует ли параметр id во вводе:

```
if (isset ($id))
    ... id parameter is present ...
else
    ... id parameter is not present ...
```

Если поле формы может иметь несколько значений (как группа флажков или прокручиваемый список с множественным выбором), необходимо представить его в форме именем, которое заканчивалось бы на []. Например, списковый элемент, созданный на основе столбца accessories типа SET из таблицы cow_order, имеет одно значение для каждого разрешенного значения множества. Для того чтобы PHP воспринимал значения списка как массив, назовите поле не accessories, а accessories[]. (См. пример в рецепте 18.3.) При отправке формы PHP помещает массив значений в параметр с таким же именем, но без [], обратиться к которому можно так:

```
$accessories = get_param_val ("accessories");
```

Переменная $accessories будет массивом. (Вне зависимости от того, имеет ли параметр множество значений, одно или ни одного. Определяющим фактором является не то количество значений, которое параметр имеет в действительности, а то, использовалась ли нотация [] для соответствующего поля формы.)

Функция get_param_val() проверяет в поиске значений параметров массивы $_GET, $_POST, $HTTP_GET_VARS и $HTTP_POST_VARS. То есть она корректно работает с PHP 3 и PHP 4 для запросов, сделанных GET и POST, вне зависимости от того, включен ли параметр конфигурации register_globals. Функция предполагает лишь одно – параметр track_vars включен.

Функция get_param_val() работает корректно и вне зависимости от того, разрешены ли магические кавычки. Она использует вспомогательную функцию strip_slash_helper(), которая при необходимости удаляет символ обратного слэша из значений параметров:

```
function strip_slash_helper ($val)
{
    if (!is_array ($val))
        $val = stripslashes ($val);
    else
    {
        reset ($val);
        while (list ($k, $v) = each ($val))
            $val[$k] = strip_slash_helper ($v);
    }
    return ($val);
}
```

Функция strip_slash_helper() определяет, скаляром или массивом является значение, и обрабатывает его соответствующим образом. Для значений-массивов используется рекурсия, так как в PHP 4 можно создавать вложенные массивы входных параметров.

Для того чтобы упростить получение списка имен всех параметров, напишем еще одну функцию:

```
function get_param_names ()
{
global $HTTP_GET_VARS, $HTTP_POST_VARS;

    # Создать ассоциативный массив, в котором для каждого элемента
    # имя параметра является и ключом, и значением.
    # (Использование ассоциативного массива устраняет дубликаты.)
    $keys = array ();
    if (isset ($_GET))
    {
        reset ($_GET);
        while (list ($k, $v) = each ($_GET))
            $keys[$k] = $k;
    }
    else if (isset ($HTTP_GET_VARS))
    {
        reset ($HTTP_GET_VARS);
        while (list ($k, $v) = each ($HTTP_GET_VARS))
            $keys[$k] = $k;
    }
    if (isset ($_POST))
    {
        reset ($_POST);
        while (list ($k, $v) = each ($_POST))
            $keys[$k] = $k;
    }
    else if (isset ($HTTP_POST_VARS))
    {
        reset ($HTTP_POST_VARS);
        while (list ($k, $v) = each ($HTTP_POST_VARS))
            $keys[$k] = $k;
    }
    return ($keys);
}
```

Функция get_param_names() возвращает список имен параметров, присутствующих в массивах переменных HTTP, при этом если массивы частично совпадают, повторяющиеся имена удаляются. Возвращенное значение – это ассоциативный массив, в котором и ключи, и значения установлены в имена параметров, так что и ключи, и значения можно использовать как список имен. Следующий пример выводит имена, используя значения:

```
$param_names = get_param_names ();
while (list ($k, $v) = each ($param_names))
    print (htmlspecialchars ($v) . "<br />\n");
```

Для сценариев на PHP 3 необходимо разделять параметры в URL символами &. По умолчанию этого требует и PHP 4, хотя в данном случае вы можете заменить символ-разделитель, используя конфигурационный параметр arg_separator в файле инициализации PHP.

Python

Модуль cgi для Python предоставляет доступ к входным параметрам, присутствующим в окружении сценария. Импортируйте этот модуль, затем создайте объект FieldStorage, используя одноименный метод:

```
import cgi

param = cgi.FieldStorage ()
```

Метод FieldStorage возвращает информацию о параметрах, переданных посредством GET или POST, поэтому вам не нужно знать, какой именно метод был использован для отправки запроса. Объект FieldStorage содержит по элементу для каждого параметра, присутствующего в окружении. Список доступных параметров можно получить так:

```
names = param.keys ()
```

Если указанный параметр name имеет одно значение, то сопоставленное ему значение будет скаляром, к которому можно обратиться так:

```
val = param[name].value
```

Если у параметра несколько значений, то param[name] будет списком объектов MiniFieldStorage, имеющих атрибуты name и value. Все они имеют одинаковые имена (совпадающие с name) и одно из значений параметра. Для создания списка, включающего все значения такого параметра, выполните следующее:

```
val = []
for item in param[name]:
    val.append (item.value)
```

Отличить параметр с единственным значением от параметра с несколькими значениями можно, проверив его тип. Приведем код, показывающий, как получить имена параметров и обойти их все, выводя имена и значения, при этом отображая параметры с множеством значений в виде списка, разделенного запятыми:[1]

```
param = cgi.FieldStorage ()
param_names = param.keys ()
param_names.sort ()
print "<p>Parameter names:", param_names, "</p>"
items = []
for name in param_names:
    if type (param[name]) is not types.ListType:    # это скаляр
```

[1] Этот код требует импортирования модулей string и types в дополнение к cgi.

```
            ptype = "scalar"
            val = param[name].value
        else:                                          # это список
            ptype = "list"
            val = []
            for item in param[name]:          # обойти MiniFieldStorage
                val.append (item.value)       # элементы для получения значений
            val = string.join (val, ",")      # преобразовать в строку для вывода
        items.append ("type=" + ptype + ", name=" + name + ", value=" + val)
    print make_unordered_list (items)
```

Python порождает исключение, если вы пытаетесь обратиться к параметру, не включенному в объект `FieldStorage`. Чтобы избежать этого, используйте `has_key()` для проверки существования параметра:

```
if param.has_key (name):
    print "parameter " + name + " exists"
else:
    print "parameter " + name + " does not exist"
```

У параметров с единственным значением есть и другие атрибуты, не только `value`. Например, у параметра, представляющего загружаемый файл, есть дополнительные атрибуты, с помощью которых можно получить содержимое файла. Подробнее об этом мы поговорим в рецепте 18.8.

Модуль `cgi` ожидает, что параметры URL будут разделены символами `&`. Если вы формируете гиперссылку на сценарий, используя модуль `cgi`, и URL содержит параметры, не разделяйте их символами `;`.

Java

Для страниц JSP доступ к параметрам запроса обеспечивает неявный объект `request` при помощи следующих методов:

`getParameterNames()`

Возвращает перечень объектов `String`, по одному для каждого имени параметра запроса.

`getParameterValues(String name)`

Возвращает массив объектов `String`, по одному для каждого значения, сопоставленного параметру, или `null`, если параметр не существует.

`getParameterValue(String name)`

Возвращает первое значение параметра или `null`, если параметр не существует.

Следующий пример показывает, как использовать эти методы для вывода параметров запроса:

```
<%@ page import="java.util.*" %>

<ul>
<%
```

```
    Enumeration e = request.getParameterNames ();
    while (e.hasMoreElements ())
    {
        String name = (String) e.nextElement ();
        // использовать массив, если параметр имеет несколько значений
        String[] val = request.getParameterValues (name);
        out.println ("<li> name: " + name + "; values:");
        for (int i = 0; i < val.length; i++)
            out.println (val[i]);
        out.println ("</li>");
    }
%>
</ul>
```

Параметры запроса также доступны внутри тегов JSTL посредством специальных переменных `param` и `paramValues`. Переменная `param[name]` возвращает первое значение указанного параметра и поэтому лучше всего подходит для параметров с единственным значением:

```
color value:
<c:out value="${param['color']}" />
```

Переменная `paramValues[name]` возвращает массив значений параметра, поэтому ее удобно использовать для параметров, которые могут иметь несколько значений:

```
accessory values:
<c:forEach var="val" items="${paramValues['accessories']}">
    <c:out value="${val}" />
</c:forEach>
```

Вы также можете обратиться к параметру, используя точечное обозначение, если имя параметра разрешено как имя свойства объекта:

```
color value:
<c:out value="${param.color}" />
accessory values:
<c:forEach var="val" items="${paramValues.accessories}">
    <c:out value="${val}" />
</c:forEach>
```

Для вывода списка объектов параметров с атрибутами `key` и `value` обработаем в цикле переменную `paramValues`:

```
<ul>
    <c:forEach var="p" items="${paramValues}">
        <li>
            name:
            <c:out value="${p.key}" />;
            values:
            <c:forEach var="val" items="${p.value}">
                <c:out value="${val}" />
            </c:forEach>
```

```
        </li>
    </c:forEach>
</ul>
```

Для создания URL, указывающего на страницы JSP и содержащего в конце параметры, следует разделять такие параметры символами &.

18.6. Проверка корректности ввода через Web

Задача

После извлечения параметров, полученных сценарием, разумно проверить их корректность.

Решение

Обработка ввода через Web – это одна из разновидностей импорта данных, поэтому после извлечения входных параметров вы можете проверить их корректность, используя приемы, описанные в главе 10.

Обсуждение

Одним из этапов обработки формы является извлечение ввода, полученного при отправке пользователем формы. Ввод также можно получить в виде параметров, указанных в конце URL. Вне зависимости от того, из какого источника получены входные данные, разумно проверить их перед сохранением в базе данных.

Когда клиент отправляет вам входные данные черезWeb, вы ничего не знаете о том, что отправлено. Если вы предлагаете пользователям форму для заполнения, обычно они, конечно, будут вести себя хорошо и вводить именно такие значения, каких вы ожидаете. Но злоумышленник может сохранить форму в файле, изменить этот файл так, чтобы разрешить форме то, чего вы совсем не ожидаете, затем загрузить файл в окно броузера и отправить измененную форму. Ваш сценарий обработки формы не почувствует разницы. Если вы создаете его только для обработки значений, введенных добропорядочными пользователями, то при получении неожидаемого ввода сценарий может работать некорректно или завершиться с ошибкой или даже сделать что-то ужасное с вашей базой данных (в рецепте 18.7 рассказано о том, что именно может случиться). Поэтому следует быть предусмотрительным и выполнить проверку корректности ввода через Web перед его использованием для формирования запросов к базе данных.

Предварительная проверка нужна не только для борьбы со злоумышленниками. Если вы просите заполнить поле, а пользователь забывает это сделать, нужно напомнить пользователю о необходимости ввода значения. Можно использовать простейшую проверку типа «Указан ли параметр?», а можно и что-нибудь посложнее. Типичными примерами операций определения корректности данных являются:

- Проверка формата содержимого, то есть получение уверенности в том, что значение выглядит как целое число или дата. При этом может потребоваться преобразование формата для обработки в MySQL (например, преобразование даты *MM/DD/YY* в формат ISO).
- Определение того, входит ли значение в допустимое множество. Возможно, значение должно присутствовать в определении столбца `ENUM` или `SET` или содержаться в справочной таблице.
- Удаление посторонних символов, таких как пробелы или дефисы, из телефонных номеров или номеров кредитных карт.

Некоторые из этих операций имеют мало общего с MySQL, если не считать того, что вы хотите, чтобы значения соответствовали типам столбцов, в которых их предполагается хранить или с которыми они будут сравниваться. Например, если собираетесь хранить значение в столбце типа `INT`, то можете сначала убедиться в том, что оно целое (используем Perl):

```
$val =~ /^\d+$/
    or die "Hey! '" . escapeHTML ($val) . "' is not an integer!\n";
```

Некоторые же проверки корректности тесно связаны с MySQL. Если значение поля будет храниться в столбце `ENUM`, вы можете убедиться в том, что значение входит в множество разрешенных значений столбца перечислимого типа, проверив определение столбца при помощи `SHOW COLUMNS`.

Я упомянул несколько видов проверки корректности ввода через Web, которые вы можете захотеть провести, и больше не буду об этом говорить. Эти, а также другие способы проверки корректности описаны в главе 10, которая ориентирована в основном на проверку пакетного ввода, но приведенные там методы работают и для веб-программирования. Ведь, по сути дела, обработка форм и параметров URL – это не что иное, как выполнение операции импорта данных.

18.7. Использование ввода через Web для формирования запросов

Задача

Вводу, полученному через Web, нельзя доверять и использовать его в запросе без принятия соответствующих мер.

Решение

Проведите «санитарную обработку» значений данных, используя заполнители или функцию заключения в кавычки.

Обсуждение

После того как значения входных параметров извлечены и проверены на корректность, их можно использовать для построения запросов. Это самая

простая часть процедуры, хотя и требуется принять некоторые меры предосторожности, чтобы не сделать ошибку, о которой придется сожалеть. Сначала давайте подумаем о том, что плохого может случиться, а затем посмотрим, как предотвратить такие проблемы.

Предположим, что у вас есть форма поиска, содержащая поле для ключевого слова, которое действует как внешний интерфейс для простой поисковой машины. Когда пользователь вводит ключевое слово, вы планируете использовать его для поиска соответствующих записей в таблице, формируя запрос так:

```
SELECT * FROM mytbl WHERE keyword = 'ключевое_слово'
```

Здесь *ключевое_слово* представляет значение, введенное пользователем. Если введено, например, значение `eggplant`, то результирующий запрос будет таким:

```
SELECT * FROM mytbl WHERE keyword = 'eggplant'
```

Запрос возвращает все соответствующие записи, вероятно, формируя небольшое результирующее множество. Но предположим, что мы имеем дело с коварным пользователем, который пытается испортить ваш сценарий, введя такое значение:

```
eggplant' OR 'x'='x
```

В данном случае запрос выглядит так:

```
SELECT * FROM mytbl WHERE keyword = 'eggplant' OR 'x'='x'
```

Этому запросу соответствуют все записи таблицы! Если таблица достаточно большая, то ввод можно рассматривать как атаку типа «отказ в обслуживании» на сценарий, так как он может привести к тому, что система будет тратить все ресурсы, предназначенные для обработки корректных запросов, на бесполезную работу. Возможны такие результаты:

- Чрезмерная нагрузка на сервер MySQL.
- Проблемы нехватки памяти сценария при попытке обработать результирующее множество, полученное от MySQL.
- Чрезмерное потребление пропускной способности сети при отправке результатов клиенту.

Если сценарий формирует предложение `DELETE`, то последствия такой подрывной деятельности могут быть еще печальнее – сценарий может выдать запрос, полностью опустошающий таблицу, при том, что вы планировали удалить всего одну запись.

Мораль в том, что предоставление веб-интерфеса к базе данных делает вас открытым для атак определенного рода. Однако проблемы можно предотвратить, всегда следуя одному простому правилу: «Никогда не помещать значения данных в строки запроса буквально». Используйте заполнители или функцию кодирования. Например, в Perl можно обработать входной параметр при помощи заполнителей так:

```
$keyword = param ("keyword");
$sth = $dbh->prepare ("SELECT * FROM mytbl WHERE keyword = ?");
$sth->execute ($keyword);
# ... извлечение результирующего множества ...
```

Или при помощи `quote()`:

```
$keyword = param ("keyword");
$keyword = $dbh->quote ($keyword);
$sth = $dbh->prepare ("SELECT * FROM mytbl WHERE keyword = $keyword");
$sth->execute ();
# ... извлечение результирующего множества ...
```

В любом случае, если пользователь вводит некорректное значение, запрос будет таким:

```
SELECT * FROM mytbl WHERE keyword = 'eggplant\' OR \'x\'=\'x'
```

Ввод становится безобидным, запрос не находит ни одной соответствующей условиям записи (вместо всех записей) – это определенно более удачный ответ тому, кто пытается вам навредить.

Применение заполнителей и способы кодирования для PHP, Python и Java аналогичны, они обсуждались в рецептах 2.6 и 2.7. Что касается страниц JSP, созданных при помощи библиотеки тегов JSTL, вы можете заключать значения входных параметров в кавычки, используя заполнители и тег `<sql:param>` (рецепт 16.3). Например, чтобы использовать значение параметра `keyword` формы в предложении `SELECT`, выполните:

```
<sql:query var="rs" dataSource="${conn}">
    SELECT * FROM mytbl WHERE keyword = ?
    <sql:param value="${param['keyword']}" />
</sql:query>
```

Заполнители и функции кодирования применяются только к значениям данных SQL. Они не занимаются обработкой Web-ввода, используемого для других типов элементов запроса, таких как имена баз данных, таблиц и столбцов. Если вы планируете вставлять такие значения в запрос, необходимо делать это буквально, поэтому предварительно следует проверить их. Например, если вы создаете такой запрос, необходимо проверить, разумное ли значение содержит `$tbl_name`:

```
SELECT * FROM $tbl_name;
```

Но что значит «разумное»? Если у вас нет таблиц, имена которых содержат странные символы, достаточно просто убедиться в том, что `$tbl_name` включает только буквенно-цифровые символы и символы подчеркивания. Или можно выполнить запрос `SHOW TABLES`, чтобы проверить, есть ли таблица с таким именем в базе данных. Это более надежно, но требуется дополнительный запрос.

Кроме того, методы использования заполнителей не ставят перед собой вопрос об интерпретации: если поле формы является необязательным, что следует сохранить в базе данных, если пользователь оставит его пустым? Вероятно, значение представляет пустую строку или должно восприниматься

как NULL. Для решения вопроса можно обратиться к метаданным столбца. Если столбец допускает использование NULL, то пустые значения интерпретируются как NULL. В противном случае пустое поле означает пустую строку.

Пробуйте взламывать ваши сценарии

В этом разделе акцент был сделан на предотвращении атак на ваши сценарии со стороны других пользователей. Но иногда бывает полезно попытаться представить себя на месте злоумышленника и подумать в направлении: «Как разрушить это приложение?». То есть можно ли передать сценарию какой-то ввод, который он не сможет обработать, что приведет к формированию некорректного запроса? Если вам удастся заставить сценарий вести себя некорректно, значит, это удастся и другим пользователям, случайно или намеренно. Относитесь ко входным данным осторожно и пишите свои приложения соответственно. Лучше перестраховаться, чем надеяться на «авось»!

См. также

Следующие разделы главы показывают, как встраивать веб-ввод в запросы. Рецепт 18.8 рассказывает о загрузке файлов в MySQL. Рецепт 18.9 приводит простое поисковое приложение, использующее ввод в качестве ключевых слов. Рецепты 18.10 и 18.11 обрабатывают параметры в URL.

18.8. Обработка загружаемых файлов

Задача

Вы хотите разрешить загрузку файлов на ваш веб-сервер и хранение их в базе данных.

Решение

Предложите пользователю веб-форму, включающую поле для файла. Когда пользователь отправит форму, извлеките файл и сохраните в MySQL.

Обсуждение

Особым видом ввода через Web является загружаемый файл. Файл отправляется в составе запроса POST, но обрабатывается не так, как остальные параметры POST, поскольку файл представлен несколькими элементами информации: содержимым, MIME-типом, исходным именем файла у клиента и именем временного хранения на хосте веб-сервера.

Для обработки загружаемого файла необходимо отправить пользователю специальную форму. Это верно для любого API, в котором создается форма. Однако после отправки формы пользователем все операции проверки и обработки загружаемого файла являются специфичными для каждого API.

Чтобы создать форму, позволяющую загружать файлы, необходимо указать метод POST в открывающем теге <form>, кроме того у него должен быть атрибут enctype (тип кодирования) со значением multipart/form-data:

```
<form method="POST" enctype="multipart/form-data" action="имя_сценария">
```

Если вы не укажете такое кодирование, форма будет отправлена с типом кодирования по умолчанию (application/x-www-form-urlencoded), и файл будет передан некорректно.

Для включения загружаемого файла в форму используйте элемент <input> типа file. Например, элемент, представляющий 60-символьное поле файла upload_file, должен быть таким:

```
<input type="file" name="upload_file" size="60" />
```

Броузер выводит его как текстовое поле ввода, в которое пользователь может вручную вводить имя. Он также предлагает кнопку Browse для выбора файла в стандартном системном диалоге поиска файла. Когда пользователь выбирает файл и отправляет форму, броузер кодирует содержимое файла для включения в результирующий запрос POST. В этот момент веб-сервер получает запрос и вызывает ваш сценарий для его обработки. Детали загрузки файла определяются конкретным используемым вами API, но в целом процедура всегда такова:

- Файл уже загружен и сохранен во временном каталоге ко времени начала работы сценария обработки. Все сценарии должны прочитать его. Временный файл будет доступен сценарию через дескриптор открытого файла, через временное имя файла или и так, и так. Размер файла можно получить при помощи файлового дескриптора. API может обеспечить доступ и к другой информации о файле, такой как его MIME-тип. (Только имейте в виду, что некоторые броузеры не отправляют значение MIME.)
- Загружаемые файлы автоматически удаляются веб-сервером при завершении сценария. Если вы хотите, чтобы содержимое файла сохранилось и после окончания работы сценария, сохраните его явно (например, в базе данных или каком-то другом каталоге файловой системы). При сохранении в файловой системе необходимо выбрать каталог, доступный веб-серверу.
- API может позволить вам управлять положением каталога временных файлов или максимальным размером загружаемых файлов. Переход к каталогу, доступному только веб-серверу, может несколько улучшить защиту от пользователей, имеющих учетные записи на хосте сервера.

В этом разделе описано, как создавать формы, включающие поле загрузки файла. Показано, как обрабатывать такие файлы, на примере сценария на Perl *post_image.pl*. Этот сценарий напоминает сценарий *store_image.pl*, написанный для загрузки изображений из командной строки (рецепт 17.6). Сценарий *post_image.pl* отличается тем, что позволяет сохранить изображения, полученные через Web; кроме того, он хранит изображения только в MySQL, а *store_image.pl* – в MySQL и файловой системе.

Также в этом разделе рассказано о получении информации из загружаемого файла при помощи PHP и Python. Я не буду приводить весь код отправки изображения, как это сделано для Perl, но в дистрибутиве `recipes` вы найдете полный текст реализаций *post_image.pl* для PHP и Python.

Perl

Используя модуль CGI.pm, вы можете задать сложное (multipart) кодирование формы несколькими способами. Все приведенные ниже предложения эквиваленты:

```
print start_form (-action => url (), -enctype => "multipart/form-data");
print start_form (-action => url (), -enctype => MULTIPART ());
print start_multipart_form (-action => url ());
```

Первое предложение задает тип кодирования буквально. Второе использует функцию the CGI.pm `MULTIPART()`, применить которую проще, чем вспоминать значение. Самым простым является третье предложение, так как функция `start_multipart_form()` сама добавляет параметр `enctype`. (Как и функция `start_form()`, `start_multipart_form()` использует по умолчанию метод запроса `POST`, поэтому нет необходимости в аргументе `method`.)

Рассмотрим простую форму, содержащую текстовое поле для присвоения изображению имени, файловое поле для выбора файла изображения и кнопку отправки:

```
print start_multipart_form (-action => url ()),
        "Image name:", br (),
        textfield (-name =>"image_name", -size => 60),
        br (), "Image file:", br (),
        filefield (-name =>"upload_file", -size => 60),
        br (), br (),
        submit (-name => "choice", -value => "Submit"),
        end_form ();
```

Когда пользователь отправит загруженный файл, приступаем к его обработке, извлекая значение параметра для поля файла:

```
$file = param ("upload_file");
```

Значение параметра загрузки файла является особенным, так как его можно использовать двумя способами. Можно интерпретировать его как дескриптор открытого файла для чтения содержимого файла или передать его в `uploadInfo()` для получения ссылки на хеш, предоставляющий сведения о файле, например, его MIME-тип. Рассмотрим сценарий *post_image.pl*, который создает форму и обрабатывает ее после отправки пользователем. При первом вызове *post_image.pl* генерирует форму с полем загрузки. На момент первого вызова никакой файл еще не загружен, так что сценарию больше делать нечего. Если пользователь передает файл изображения, сценарий принимает имя файла, читает его содержимое, определяет MIME-тип и сохраняет новую запись в таблице `image`. Для наглядности *post_image.pl* также выводит

всю информацию о загруженном файле, доступную посредством функции `uploadInfo()`**.**

```
#! /usr/bin/perl -w
# post_image.pl - позволяет пользователю загрузить файлы изображений
# при помощи запросов POST

use strict;
use lib qw(/usr/local/apache/lib/perl);
use CGI qw(:standard escapeHTML);
use Cookbook;

print header (), start_html (-title => "Post Image", -bgcolor => "white");

# Использовать multipart-кодирование, так как форма содержит поле загрузки файла.

print start_multipart_form (-action => url ()),
        "Image name:", br (),
        textfield (-name =>"image_name", -size => 60),
        br (), "Image file:", br (),
        filefield (-name =>"upload_file", -size => 60),
        br (), br (),
        submit (-name => "choice", -value => "Submit"),
        end_form ();

# Получить дескриптор файла изображения и имя, которое должно
# быть присвоено изображению.

my $image_file = param ("upload_file");
my $image_name = param ("image_name");

# Должно быть указано или ни одного параметра (если сценарий вызван впервые)
# или два (то есть форма должна быть заполнена). Если указан лишь один параметр,
# форма заполнена не полностью.

my $param_count = 0;
++$param_count if defined ($image_file) && $image_file ne "";
++$param_count if defined ($image_name) && $image_name ne "";

if ($param_count == 0)          # первый вызов
{
    print p ("No file was uploaded.");
}
elsif ($param_count == 1)       # незаполненная форма
{
    print p ("Please fill in BOTH fields and resubmit the form.");
}
else                            # файл загружен
{
my ($size, $data);

    # Если файл изображения загружен, вывести какую-нибудь информацию о нем
    # и сохранить его в базе данных.

    # Получить ссылку на хеш с данными о файле и вывести информацию
    # в формате "key=x, value=y".
```

```
    my $info_ref = uploadInfo ($image_file);
    print p ("Information about uploaded file:");
    foreach my $key (sort (keys (%{$info_ref})))
    {
        printf p ("key="
                  . escapeHTML ($key)
                  . ", value="
                  . escapeHTML ($info_ref->{$key}));
    }
    $size = (stat ($image_file))[7];   # получить размер файла
                                       # из файлового дескриптора
    print p ("File size: " . $size);

    binmode ($image_file);  # удобно для двоичных данных
    if (sysread ($image_file, $data, $size) != $size)
    {
        print p ("File contents could not be read.");
    }
    else
    {
        print p ("File contents were read without error.");

        # Получить MIME-тип, использовать общее умолчание, если не указан.

        my $mime_type = $info_ref->{'Content-Type'};
        $mime_type = "application/octet-stream" unless defined ($mime_type);

        # Сохранить изображение в таблице базы данных.  (Использовать REPLACE
        # для замены старых изображений с таким же именем.)

        my $dbh = Cookbook::connect ();
        $dbh->do ("REPLACE INTO image (name,type,data) VALUES(?,?,?)",
                  undef,
                  $image_name, $mime_type, $data);
        $dbh->disconnect ();
    }
}

print end_html ();

exit (0);
```

PHP

Для создания формы загрузки в PHP включите в нее поле файла. При желании можно также добавить перед полем файла скрытое поле с именем `MAX_FILE_SIZE` **и значением, определяющим желаемый максимальный допустимый размер загружаемого файла:**

```
<form method="POST" enctype="multipart/form-data"
    action="<?php print (get_self_path ()); ?>">
<input type="hidden" name="MAX_FILE_SIZE" value="4000000" />
Image name:<br />
<input type="text" name="image_name" size="60" />
```

```
<br />
Image file:<br />
<input type="file" name="upload_file" size="60" />
<br /><br />
<input type="submit" name="choice" value="Submit" />
</form>
```

Знайте, что MAX_FILE_SIZE носит только рекомендательный характер, и такое ограничение без труда можно обойти. Для того чтобы указать значение, которое нельзя превышать, используйте параметр конфигурации upload_max_filesize в инициализационном файле PHP. Кроме того, есть параметр file_uploads, определяющий принципиальную возможность загрузки файлов.

После отправки формы пользователем информацию о загруженном файле можно получить так:

- Начиная с версии PHP 4.1 данные загруженного файла из запроса POST помещаются в специальный массив $_FILES, который включает в себя запись для каждого загруженного файла. Каждая запись представляет собой массив из четырех элементов. Например, если форма содержит поле файла с именем upload_file и пользователь отправляет файл, информация о нем доступна в следующих переменных:

`$_FILES["upload_file"]["name"]`	*исходное имя файла на клиентском хосте*
`$_FILES["upload_file"]["tmp_name"]`	*временное имя файла на хосте сервера*
`$_FILES["upload_file"]["size"]`	*размер файла в байтах*
`$_FILES["upload_file"]["type"]`	*MIME-тип файла*

  Будьте внимательны, поскольку запись для поля загрузки может существовать, даже если пользователь не отправил файл. Тогда значение tmp_name будет пустой строкой или строкой none.

- В более ранних версиях PHP 4 информация о загружаемых файлах хранилась в массиве $HTTP_POST_FILES, записи которого были устроены так же, как и в $_FILES. Информация о файле, отправленном из поля upload_file, доступна в таких переменных:

`$HTTP_POST_FILES["upload_file"]["name"]`	*исходное имя файла на клиентском хосте*
`$HTTP_POST_FILES["upload_file"]["tmp_name"]`	*временное имя файла на хосте сервера*
`$HTTP_POST_FILES["upload_file"]["size"]`	*размер файла в байтах*
`$HTTP_POST_FILES["upload_file"]["type"]`	*MIME-тип файла*

- До версии PHP 4 обращение к информации о загружаемых файлах для поля upload_file осуществлялось через четыре переменные $HTTP_POST_VARS:

`$HTTP_POST_VARS["upload_file_name"]`	*исходное имя файла на клиентском хосте*
`$HTTP_POST_VARS["upload_file"]`	*временное имя файла на хосте сервера*
`$HTTP_POST_VARS["upload_file_size"]`	*размер файла в байтах*
`$HTTP_POST_VARS["upload_file_type"]`	*MIME-тип файла*

$_FILES — это суперглобальный массив (глобальный в любой области видимости). $HTTP_POST_FILES и $HTTP_POST_VARS должны объявляться с ключевым словом global при использовании не в глобальной области видимости, например, внутри функции.

Чтобы не терять время попусту, пытаясь определить, какой из массивов содержит информацию о загружаемых файлах, разумно написать функцию, которая делала бы всю работу. Функция get_upload_info() принимает аргумент, соответствующий имени поля загружаемого файла. Затем она просматривает массивы $_FILES, $HTTP_POST_FILES и $HTTP_POST_VARS и возвращает ассоциативный массив информации о файле или неустановленное значение, если информация недоступна. При успешном вызове ключами элементов массива будут "tmp_name", "name", "size" и "type" (то есть ключи совпадают с ключами записей в массивах $_FILES или $HTTP_POST_FILES).

```
function get_upload_info ($name)
{
global $HTTP_POST_FILES, $HTTP_POST_VARS;

    unset ($unset);
    # Сначала ищем информацию в массиве PHP 4.1 $_FILES.
    # Проверяем элемент tmp_name, чтобы убедиться в том, что файл существует.
    # (Запись в $_FILES может присутствовать, даже если файл не был загружен.)
    if (isset ($_FILES))
    {
        if (isset ($_FILES[$name])
                && $_FILES[$name]["tmp_name"] != ""
                && $_FILES[$name]["tmp_name"] != "none")
            return ($_FILES[$name]);
        return (@$unset);
    }
    # Затем ищем информацию в массиве PHP 4 $HTTP_POST_FILES.
    if (isset ($HTTP_POST_FILES))
    {
        if (isset ($HTTP_POST_FILES[$name])
                && $HTTP_POST_FILES[$name]["tmp_name"] != ""
                && $HTTP_POST_FILES[$name]["tmp_name"] != "none")
            return ($HTTP_POST_FILES[$name]);
        return (@$unset);
    }
    # Ищем переменные загрузки PHP 3. Проверяем элемент_name,
    # так как на самом деле $HTTP_POST_VARS[$name] может не быть полем файла.
    if (isset ($HTTP_POST_VARS[$name])
        && isset ($HTTP_POST_VARS[$name . "_name"]))
    {
        # Сопоставляем элементам PHP 3 имена элементов PHP 4.
        $info = array ();
        $info["name"] = $HTTP_POST_VARS[$name . "_name"];
        $info["tmp_name"] = $HTTP_POST_VARS[$name];
        $info["size"] = $HTTP_POST_VARS[$name . "_size"];
        $info["type"] = $HTTP_POST_VARS[$name . "_type"];
```

```
            return ($info);
        }
        return (@$unset);
    }
```

Сценарий *post_image.php* использует эту функцию для получения информации об изображении и сохранения ее в MySQL.

Параметр конфигурации PHP `upload_tmp_dir` указывает место сохранения загруженных файлов. По умолчанию во многих системах это */tmp*, но при желании вы можете изменить настройки PHP, задав сохранение в каталоге, принадлежащем пользователю с идентификатором веб-сервера, соответственно, ограничив доступ к нему.

Python

Простую форму загрузки на Python можно написать так:

```
print "<form method=\"POST\" enctype=\"multipart/form-data\" action=\"%s\">" \
        % (os.environ["SCRIPT_NAME"])
print "Image name:<br />"
print "<input type=\"text\" name=\"image_name\", size=\"60\" />"
print "<br />"
print "Image file:<br />"
print "<input type=\"file\" name=\"upload_file\", size=\"60\" />"
print "<br /><br />"
print "<input type=\"submit\" name=\"choice\" value=\"Submit\" />"
print "</form>"
```

Когда пользователь отправляет форму, ее содержимое можно получить при помощи метода `FieldStorage()` модуля `cgi` (см. рецепт 18.5). Результирующий объект содержит элемент для каждого входного параметра. Информацию для загруженного из поля файла можно получить так:

```
form = cgi.FieldStorage ()
if form.has_key ("upload_file") and form["upload_file"].filename != "":
    image_file = form["upload_file"]
else:
    image_file = None
```

В большей части прочитанных мною документов сказано, что атрибут `file` объекта, соответствующего полю файла, установлен в значение «истина», если файл был загружен. К сожалению, атрибут `file` установлен в значение «истина», даже если пользователь отправляет форму, оставив поле файла пустым. Может случиться и так, что атрибут `type` установлен, при том что файл не был загружен (например, в `application/octet-stream`). Мой опыт подсказывает, что наиболее надежный способ определить, был ли файл загружен, – проверить атрибут `filename`:

```
form = cgi.FieldStorage ()
if form.has_key ("upload_file") and form["upload_file"].filename:
    print "<p>A file was uploaded</p>"
```

```
else:
    print "<p>A file was not uploaded</p>"
```

Считая, что файл был загружен, обращаемся к атрибуту параметра `value` для чтения файла и получения его содержимого:

```
data = form["upload_file"].value
```

Те, кому интересны подробности использования этой функции для получения информации об изображении и сохранения ее в MySQL, могут обратиться к сценарию *post_image.py*.

18.9. Выполнение поиска и получение результатов

Задача

Вы хотите реализовать веб-интерфейс для поиска.

Решение

Предложите пользователю форму, содержащую поля, в которых можно указать параметры поиска – ключевые слова. Используйте эти ключевые слова для формирования запроса, затем выведите результаты.

Обсуждение

Сценарий, реализующий веб-интерфейс поиска, делает удобной работу посетителей вашего сайта – им не нужно знать SQL, чтобы найти информацию в базе данных. Пользователи просто водят ключевые слова, описывающие интересующий их объект, а ваш сценарий формирует запросы на основе этих слов. Как правило, для этого применяется форма с одним или несколькими полями для ввода параметров поиска. Пользователь заполняет форму, отправляет ее и получает обратно новую страницу, содержащую записи, соответствующие параметрам.

Вы как разработчик сценария должны реализовать следующие операции:

- Генерирование формы и отправка ее пользователю.
- Интерпретация отправленной формы и создание запроса на основе ее содержимого. Соответствующим образом используются заполнители и кавычки для предотвращения разрушения сценария некорректным вводом.
- Отображение результата запроса. Это просто, если результирующее множество небольшое, или сложнее, если результат большой. В последнем случае вы можете захотеть представить найденные записи постранично, то есть вывод будет состоять из нескольких страниц, каждая из которых будет содержать подмножество результата запроса. Преимущество многостраничных выводов в том, что они сразу не вываливают на пользователя огромные объемы информации (поговорим об этом в рецепте 18.10).

В этом разделе приводится сценарий *search_state.pl*, реализующий самый простой интерфейс поиска: форму с одним полем для ввода ключевого слова, по которому строится запрос, возвращающий не более одной записи. Сценарий выполняет двусторонний поиск в таблице `states`. Если пользователь вводит название штата, сценарий выводит соответствующую аббревиатуру. И наоборот, если пользователь вводит аббревиатуру, сценарий ищет для нее имя:

```
#! /usr/bin/perl -w
# search_state.pl - простое приложение "поиска штата"

# Вывести форму с полем ввода и кнопкой отправки. Пользователь вводит в поле
# аббревиатуру или полное название штата и отправляет форму. Сценарий находит
# аббревиатуру и выводит полное имя или находит имя и выводит аббревиатуру.

use strict;
use lib qw(/usr/local/apache/lib/perl);
use CGI qw(:standard escapeHTML);
use Cookbook;

my $title = "State Name or Abbreviation Lookup";

print header (), start_html (-title => $title, -bgcolor => "white");

# Извлечь параметр keyword. Если его значение задано и не пусто, выполняется поиск.

my $keyword = param ("keyword");

if (defined ($keyword) && $keyword !~ /^\s*$/)
{
    my $dbh = Cookbook::connect ();
    my $found = 0;
    my $s;

    # Сначала пытаемся найти совпадающую с ключевым словом аббревиатуру;
    # в случае неудачи ищем совпадающее имя.
    $s = $dbh->selectrow_array ("SELECT name FROM states WHERE abbrev = ?",
                                undef, $keyword);
    if ($s)
    {
        ++$found;
        print p ("You entered the abbreviation: " . escapeHTML ($keyword));
        print p ("The corresponding state name is : " . escapeHTML ($s));
    }
    $s = $dbh->selectrow_array ("SELECT abbrev FROM states WHERE name = ?",
                                undef, $keyword);
    if ($s)
    {
        ++$found;
        print p ("You entered the state name: " . escapeHTML ($keyword));
        print p ("The corresponding abbreviation is : " . escapeHTML ($s));
    }
    if (!$found)
    {
        print p ("You entered the keyword: " . escapeHTML ($keyword));
```

```
        print p ("No match was found.");
    }

    $dbh->disconnect ();
}

print p (qq{
Enter a state name into the form and select Search, and I will show you
the corresponding abbreviation.
Or enter an abbreviation and I will show you the full name.
});

print start_form (-action => url ());

print "State: ";
print textfield (-name => "keyword", -size => 20);
print br (),
        submit (-name => "choice", -value => "Search"),
        end_form ();

print end_html ();

exit (0);
```

Сценарий сначала проверяет, задан ли параметр `keyword`. Если задан, то выдаются запросы, ищущие соответствие значению параметра в таблице `states` и выводящие результаты. Затем сценарий отображает форму для пользователя, чтобы тот мог выполнить новый поиск.

Опробовав сценарий, вы заметите, что значение поля `keyword` переходит из одного вызова сценария в следующий. Это объясняется тем, что CGI.pm инициализирует поля формы значениями из окружения сценария. Если вас не устраивает такое поведение сценария, и вы хотите, чтобы поле поиска при открытии формы всегда было пустым, явно укажите пустое значение и параметр `override` в вызове `textfield()`:

```
print textfield (-name => "keyword",
                 -value => "",
                 -override => 1,
                 -size => 20);
```

Или очистите значение параметра в окружении перед формированием поля:

```
param (-name => "keyword", -value => "");
print textfield (-name => "keyword", -size => 20);
```

18.10. Формирование ссылок на предыдущую и следующую страницы

Задача

Запрос находит так много записей, что вывод их на одной веб-странице выглядит устрашающе.

Решение

Разбейте вывод запроса на несколько страниц и предоставьте ссылки, позволяющие пользователю перемещаться со страницы на страницу.

Обсуждение

Если запрос находит большой объем записей, то просматривать их все на одной веб-странице становится неудобно. В подобных случаях разумно распределить результат по нескольким страницам. Постраничный вывод избавляет пользователя от необходимости воспринимать большой объем данных сразу, но его сложнее реализовать.

Постраничный вывод обычно используется при поиске для представления записей, соответствующих параметрам, введенным пользователем. Для простоты примеры этого раздела не будут иметь никакого поискового интерфейса. В них будут реализовываться постраничный вывод, представляющий на одной странице 10 строк, полученных следующим запросом:

```
SELECT name, abbrev, statehood, pop  FROM states ORDER BY name;
```

В MySQL не составляет труда выбрать часть результирующего множества: добавьте инструкцию `LIMIT`, указывающую, какие записи вам нужны. `LIMIT` может принимать два аргумента: количество записей, которые следует пропустить от начала результирующего множества, и количество выбираемых записей. Запрос, выбирающий часть таблицы `states`, будет таким:

```
SELECT name, abbrev, statehood, pop  FROM states ORDER BY name
LIMIT skip,select;
```

Возникает два вопроса. Во-первых, необходимо определить соответствующие значения *skip* и *select*. Во-вторых, следует сформировать ссылки, указывающие на другие страницы результата запроса. Можно выводить страницу, приводя ссылки только на предыдущую и следующую страницы. Для этого необходимо знать, существуют ли записи, предшествующие или следующие за представленными на выведенной странице. Можно предлагать ссылки на все доступные страницы. Тогда пользователь сможет сразу переходить на любую страницу, а не только на предыдущую или следующую. Реализация такого способа требует знания общего количества записей результирующего множества и количества строк на странице, чтобы можно было получить число страниц вывода.

Постраничный вывод со ссылками на предыдущую и следующую страницы

Сценарий *state_pager1.pl* представляет записи таблицы `states` постранично, предлагая ссылки только на предыдущую и следующую страницы. Для указанной страницы необходимые ссылки определяются так:

- Ссылка на «предыдущую страницу» необходима, если в результирующем множестве содержатся записи, предшествующие отображенным на ука-

занной странице. Если текущая страница начинается с записи 1, таких записей нет.

- Ссылка на «следующую страницу» необходима, если в результирующем множестве существуют записи, следующие за представленными на текущей странице. Вы можете проверить наличие таких записей, выполнив запрос `SELECT COUNT(*)`, чтобы посмотреть, сколько всего записей найдено запросом. Есть и другой способ – выбрать на одну запись больше, чем нужно. Например, если вы выводите по 10 записей на странице, попробуйте выбрать 11. Если получите 11, значит, существует следующая страница. Если получено 10 или меньше записей, ее нет. Сценарий *state_pager1.pl* использует второй подход.

Чтобы определить текущее положение в результирующем множестве и количество отображаемых записей, *state_pager1.pl* ищет входные параметры `start` и `per_page`. При первом вызове сценария этих параметров нет, поэтому они инициализируются в 1 и 10 соответственно. После этого сценарий формирует ссылки «предыдущая страница» и «следующая страница», указывающие на него же, URL которых содержит необходимые параметры для выбора предыдущей или последующей части результирующего множества.

```
#! /usr/bin/perl -w
# state_pager1.pl - постраничное отображение штатов со ссылками
# на предыдущую/следующую страницы

use strict;
use lib qw(/usr/local/apache/lib/perl);
use CGI qw(:standard escape escapeHTML);
use Cookbook;

my $title = "Paged US State List";

my $page = header ()
            . start_html (-title => $title, -bgcolor => "white")
            . h3 ($title);

my $dbh = Cookbook::connect ();

# Проверка параметров, которые определяют, в каком именно месте вывода мы находимся.
# По умолчанию - начало результирующего множества, 10 записей на странице,
# если параметры отсутствуют или заданы некорректно.

my $start = param ("start");
$start = 1
    if !defined ($start) || $start !~ /^\d+$/ || $start < 1;

my $per_page = param ("per_page");
$per_page = 10
    if !defined ($per_page) || $per_page !~ /^\d+$/ || $per_page < 1;;

# Если start > 1, то необходима активная ссылка на предыдущую страницу.
# Чтобы определить, есть ли следующая страница, пытаемся выбрать на одну запись
# больше, чем требуется. Если удается, выводим только первые $per_page страниц,
# но добавляем активную ссылку на следующую страницу.
```

```
# Выбираем записи текущей страницы результирующего множества и пытаемся
# получить дополнительную запись. (Не для того, чтобы ее выводить, а просто
# для проверки существования следующей страницы.)

my $query = sprintf (
                "SELECT name, abbrev, statehood, pop
                FROM states
                ORDER BY name LIMIT %d,%d",
                $start - 1,         # количество пропускаемых записей
                $per_page + 1);     # количество выбираемых записей

my $tbl_ref = $dbh->selectall_arrayref ($query);

$dbh->disconnect ();

# Вывод результатов в виде таблицы HTML.
my @rows;
push (@rows, Tr (th (["Name", "Abbrevation", "Statehood", "Population"])));
for (my $i = 0; $i < $per_page && $i < @{$tbl_ref}; $i++)
{
    # получаем значения в строке $i
    my @cells = @{$tbl_ref->[$i]};  # получаем значения в строке $i
    # сопоставляем значениям HTML-закодированные значения или  
    # если значение null/пустое
    @cells = map {
                defined ($_) && $_ ne "" ? escapeHTML ($_) : " "
                } @cells;
    # добавляем ячейки в таблицу
    push (@rows, Tr (td (\@cells)));
}

$page .= table ({-border => 1}, @rows) . br ();

# Если мы не в начале результата запроса, выводим активную ссылку
# на предыдущую страницу, иначе выводим статический текст.

if ($start > 1)                 # активная ссылка
{
    my $url = sprintf ("%s?start=%d;per_page=%d",
                        url (),
                        $start - $per_page,
                        $per_page);
    $page .= "[" . a ({-href => $url}, "previous page") . "] ";
}
else                            # статический текст
{
    $page .= "[previous page]";
}

# Если мы получили дополнительную запись, выводим активную ссылку
# на следующую страницу, иначе выводим статический текст.

if (@{$tbl_ref} > $per_page)    # активная ссылка
{
    my $url = sprintf ("%s?start=%d;per_page=%d",
```

```
                        url (),
                        $start + $per_page,
                        $per_page);
    $page .= "[" . a ({-href => $url}, "next page") . "]";
}
else                            # статический текст
{
    $page .= "[next page]";
}

$page .= end_html ();

print $page;

exit (0);
```

Постраничный вывод со ссылками на каждую страницу

Сценарий *state_pager2.pl* во многом похож на *state_pager1.pl*, но в данном случае страница вывода содержит ссылки на все страницы результирующего множества запроса. Для предоставления таких ссылок необходимо знать, сколько всего строк в результате. Сценарий *state_pager2.pl* определяет это, выполняя запрос `SELECT COUNT(*)`. Поскольку известно общее количество строк, не нужно выбирать дополнительную запись при извлечении очередной порции результата для отображения.

Опустим части *state_pager2.pl*, повторяющие *state_pager1.pl*, и рассмотрим только извлечение записей и формирование ссылок:

```
# Определяем общее количество записей.

my $total_recs = $dbh->selectrow_array ("SELECT COUNT(*) FROM states");

# Выбираем записи текущей страницы результата.

my $query = sprintf (
                "SELECT name, abbrev, statehood, pop
                FROM states
                ORDER BY name LIMIT %d,%d",
                $start - 1,         # количество пропускаемых записей
                $per_page);         # количество выбираемых записей

my $tbl_ref = $dbh->selectall_arrayref ($query);

$dbh->disconnect ();

# Вывести результаты в виде таблицы HTML.
my @rows;
push (@rows, Tr (th (["Name", "Abbrevation", "Statehood", "Population"])));
for (my $i = 0; $i < @{$tbl_ref}; $i++)
{
    # получаем значения в строке $i
    my @cells = @{$tbl_ref->[$i]};
    # сопоставляем значениям HTML-закодированные значения или  
    # если значение null/пусто
```

```
    @cells = map {
                defined ($_) && $_ ne "" ? escapeHTML ($_) : " "
                } @cells;
    # добавляем ячейки в таблицу
    push (@rows, Tr (td (\@cells)));
}

$page .= table ({-border => 1}, @rows) . br ();

# Формируем ссылки на все страницы результирующего множества. Активны все
# ссылки, кроме ссылки на текущую страницу, которая выводится как статический
# текст. Формат видимого текста ссылки - "[m to n]", где m и n - номера первой
# и последней отображаемых на странице записей.

for (my $first = 1; $first <= $total_recs; $first += $per_page)
{
    my $last = $first + $per_page - 1;
    $last = $total_recs if $last > $total_recs;
    my $label = "$first to $last";
    my $link;

    if ($first != $start)   # активная ссылка
    {
        my $url = sprintf ("%s?start=%d;per_page=%d",
                           url (),
                           $first,
                           $per_page);
        $link = a ({-href => $url}, $label);
    }
    else                    # статический текст
    {
        $link = $label;
    }
    $page .= "[$link] ";
}
```

18.11. Сортировка результатов запроса по произвольному столбцу

Задача

Вы хотите вывести результат запроса на веб-странице в виде таблицы, позволяющей пользователю выбрать столбец, по которому будут упорядочены записи таблицы.

Решение

Сделайте заголовок каждого столбца гиперссылкой, выводящей таблицу, упорядоченную по соответствующему столбцу.

Обсуждение

Работающий веб-сценарий определяет, какое действие следует предпринять, запрашивая значения параметров окружения. Обычно такие параметры поступают от пользователя, но ничто не мешает сценарию самостоятельно добавить параметры в URL. Например, именно так один вызов сценария может пересылать информацию следующему вызову. Получается, что сценарий взаимодействует сам с собой при помощи URL, генерируемых для выполнения определенных действий. Этот прием можно применять для вывода результата запроса в таком виде, чтобы пользователь мог выбрать, по какому столбцу результата упорядочивать вывод. Заголовки столбцов превращаются в активные ссылки, которые заново выводят таблицу, отсортированную по указанному столбцу.

Примеры раздела будут работать с таблицей `mail`:

```
mysql> SELECT * FROM mail;
+---------------------+---------+---------+---------+---------+---------+
| t                   | srcuser | srchost | dstuser | dsthost | size    |
+---------------------+---------+---------+---------+---------+---------+
| 2001-05-11 10:15:08 | barb    | saturn  | tricia  | mars    |   58274 |
| 2001-05-12 12:48:13 | tricia  | mars    | gene    | venus   |  194925 |
| 2001-05-12 15:02:49 | phil    | mars    | phil    | saturn  |    1048 |
| 2001-05-13 13:59:18 | barb    | saturn  | tricia  | venus   |     271 |
| 2001-05-14 09:31:37 | gene    | venus   | barb    | mars    |    2291 |
| 2001-05-14 11:52:17 | phil    | mars    | tricia  | saturn  |    5781 |
| 2001-05-14 14:42:21 | barb    | venus   | barb    | venus   |   98151 |
| 2001-05-14 17:03:01 | tricia  | saturn  | phil    | venus   | 2394482 |
| 2001-05-15 07:17:48 | gene    | mars    | gene    | saturn  |    3824 |
| 2001-05-15 08:50:57 | phil    | venus   | phil    | venus   |     978 |
| 2001-05-15 10:25:52 | gene    | mars    | tricia  | saturn  |  998532 |
| 2001-05-15 17:35:31 | gene    | saturn  | gene    | mars    |    3856 |
| 2001-05-16 09:00:28 | gene    | venus   | barb    | mars    |     613 |
| 2001-05-16 23:04:19 | phil    | venus   | barb    | venus   |   10294 |
| 2001-05-17 12:49:23 | phil    | mars    | tricia  | saturn  |     873 |
| 2001-05-19 22:21:51 | gene    | saturn  | gene    | venus   |   23992 |
+---------------------+---------+---------+---------+---------+---------+
```

Для извлечения содержимого таблицы и его вывода в виде таблицы HTML можно применять приемы, описанные в рецепте 17.3. Мы будем применять те же способы, только немного изменим их, чтобы сформировать заголовки столбцов, по щелчку на которых таблица упорядочивается по соответствующему столбцу («click to sort»).

«Обычная» таблица HTML будет содержать строку заголовков столбцов, состоящую только из имен столбцов:

```
<tr>
  <th>t</th>
  <th>srcuser</th>
  <th>srchost</th>
  <th>dstuser</th>
```

```
  <th>dsthost</th>
  <th>size</th>
</tr>
```

Для того чтобы сделать заголовки столбцов активными ссылками, которые повторно вызывают сценарий для формирования вывода, упорядоченного по указанному столбцу, нужно создать такую строку заголовков:

```
<tr>
  <th><a href="имя_сценария?sort=t">t</a></th>
  <th><a href="имя_сценария?sort=srcuser">srcuser</a></th>
  <th><a href="имя_сценария?sort=srchost">srchost</a></th>
  <th><a href="имя_сценария?sort=dstuser">dstuser</a></th>
  <th><a href="имя_сценария?sort=dsthost">dsthost</a></th>
  <th><a href="имя_сценария?sort=size">size</a></th>
</tr>
```

Для построения таких заголовков сценарию необходимо знать имена столбцов таблицы, а также собственный URL. В рецептах 9.5 и 18.1 было показано, как получать эти сведения, используя метаданные запроса и информацию из окружения сценария. Например, сценарий PHP может генерировать строку заголовков для столбцов указанного запроса так:

```
$self_path = get_self_path ();
print ("<tr>\n");
for ($i = 0; $i < mysql_num_fields ($result_id); $i++)
{
    $col_name = mysql_field_name ($result_id, $i);
    printf ("<th><a href=\"%s?sort=%s\">%s</a></th>\n",
                $self_path,
                urlencode ($col_name),
                htmlspecialchars ($col_name));
}
print ("</tr>\n");
```

Сценарий *clicksort.php* реализует именно такой способ вывода. Он извлекает из окружения параметр `sort`, указывающий, по какому столбцу следует выполнить сортировку. Затем сценарий использует параметр для формирования подобного запроса:

```
SELECT * FROM $tbl_name ORDER BY $sort_col LIMIT 50
```

(Если параметр `sort` не указан, сценарий использует инструкцию `ORDER BY 1` для задания сортировки по умолчанию по первому столбцу.) Инструкция `LIMIT` используется исключительно для того, чтобы избежать больших объемов вывода для больших таблиц.

Сценарий выглядит так:

```
<?php
# clicksort.php - вывод результата запроса в виде таблицы HTML со столбцами,
# сортируемыми по щелчку на заголовке.

# Строки таблицы базы данных выводятся в виде таблицы HTML. Заголовки столбцов
```

```
# представлены гиперссылками, повторно вызывающими сценарий для вывода
# таблицы, упорядоченной по выбранному столбцу. Для больших таблиц
# вывод ограничен 50 строками.

include "Cookbook.php";
include "Cookbook_Webutils.php";

$title = "Table Display with Click-To-Sort Column Headings";

?>

<html>
<head>
<title><?php print ($title); ?></title>
</head>
<body bgcolor="white">

<?php
# ---------------------------------------------------------------------

$tbl_name = "mail";     # таблица для вывода; можно изменить

$conn_id = cookbook_connect ();

print ("<p>Table: " . htmlspecialchars ($tbl_name) . "</p>\n");
print ("<p>Click on a column name to sort the table by that column.</p>\n");

# Получить имя столбца для сортировки (необязательно). Если отсутствует,
# использовать столбец 1. Иначе выполнить простую проверку корректности имени;
# оно должно включать только буквенно-цифровые символы и подчеркивание.

$sort_col = get_param_val ("sort");     # имя столбца сортировки (необязательно)
if (!isset ($sort_col))
    $sort_col = "1";        # сортировка по первому столбцу
else if (!ereg ("^[0-9a-zA-Z_]+$", $sort_col))
    die (htmlspecialchars ("Column name $sort_col is invalid"));

# Создание запроса для выбора из указанной таблицы записей, возможно,
# упорядоченных по определенному столбцу. Вывод ограничен 50 строками
# во избежание вывода всего содержимого большой таблицы.

$query = "SELECT * FROM $tbl_name";
$query .= " ORDER BY $sort_col";
$query .= " LIMIT 50";

$result_id = mysql_query ($query, $conn_id);
if (!$result_id)
    die (htmlspecialchars (mysql_error ($conn_id)));

# Вывод результатов запроса в виде таблицы HTML.  Используем метаданные запроса
# для получения имен столбцов и вывода их в первой строке таблицы в виде
# гиперссылок, вызывающих повторное отображение таблицы, упорядоченной
# по соответствующему столбцу.

print ("<table border=\"1\">\n");
$self_path = get_self_path ();
print ("<tr>\n");
```

```
for ($i = 0; $i < mysql_num_fields ($result_id); $i++)
{
    $col_name = mysql_field_name ($result_id, $i);
    printf ("<th><a href=\"%s?sort=%s\">%s</a></th>\n",
                $self_path,
                urlencode ($col_name),
                htmlspecialchars ($col_name));
}
print ("</tr>\n");
while ($row = mysql_fetch_row ($result_id))
{
    print ("<tr>\n");
    for ($i = 0; $i < mysql_num_fields ($result_id); $i++)
    {
        # Кодировать значения, используя   для пустых ячеек.
        $val = $row[$i];
        if (isset ($val) && $val != "")
            $val = htmlspecialchars ($val);
        else
            $val = " ";
        printf ("<td>%s</td>\n", $val);
    }
    print ("</tr>\n");
}
mysql_free_result ($result_id);
print ("</table>\n");

mysql_close ($conn_id);

?>

</body>
</html>
```

В рецепте 18.7 упоминалось о том, что заполнители можно использовать только для значений данных, но не для идентификаторов, которыми и являются имена столбцов. Параметр `sort` – это имя столбца, поэтому его нельзя изменять при помощи заполнителей и функции кодирования. Вместо этого сценарий выполняет элементарную проверку на присутствие в имени только буквенно-цифровых символов и подчеркиваний. Этот простой тест работает для большей части имен таблиц (хотя и может дать сбой для таблицы с необычным именем). Такие же проверки выполняются для имен базы данных, индекса, столбца и псевдонима.

Можно проверить корректность имен столбцов другим способом – запустить запрос `SHOW COLUMNS` для определения того, какие столбцы действительно содержатся в таблице. Если столбец сортировки не входит в их число, то он не является разрешенным. Рассмотренный в разделе сценарий *clicksort.php* не делает этого. Однако дистрибутив `recipes` включает в себя аналогичный сценарий на Perl *clicksort.pl*, выполняющий подобные проверки. Обратитесь к нему, если вам нужна дополнительная информация.

Ячейки строк, следующих за строкой заголовков, содержат значения данных из таблицы базы данных, выведенных как статический текст. Пустые ячейки выводятся при помощи , поэтому они окружены такой же рамкой, как непустые значения (см. рецепт 17.3).

18.12. Счетчики посещаемости веб-страниц

Задача

Вы хотите узнать, сколько раз запрашивалась страница. Такую информацию можно использовать, например, для вывода счетчика посещаемости страницы. Этот же прием можно применять для регистрации других сведений, например, количества нажатий на каждый рекламный баннер.

Решение

Реализуйте счетчик нажатий (hit counter) для интересующей вас страницы.

Обсуждение

В этом разделе обсуждается учет обращений к странице при помощи счетчиков посещаемости. Счетчикам, отображающим то количество раз, которое веб-страница была запрошена, уже не придается столь важное значение, как раньше, вероятно, потому что авторы страниц поняли, что большую часть пользователей не заботит популярность страницы. Однако общая идея все еще применяется в различных обстоятельствах. Например, если вы выводите на странице рекламный баннер (рецепт 17.7), то продавцы могут платить вам за количество просмотров их рекламы. Вам необходимо вычислить количество обращений к каждому баннеру – адаптируйте к вашему конкретному случаю предлагаемые в разделе приемы.

Есть несколько способов создания страницы, отображающей счетчик ее просмотров. Самый простой – это хранение счетчика в файле. При запросе страницы вы открываете файл, считываете счетчик, увеличиваете его, записываете новое значение в файл и выводите на странице. Преимущество данного способа в простоте реализации, а недостаток в том, что для каждой страницы со счетчиком нужен соответствующий файл. Кроме того, он некорректно работает, когда два клиента одновременно запрашивают страницу, если только вы не используете какую-либо блокировку в процедуре доступа к файлу. Можно несколько уменьшить количество файлов, храня несколько счетчиков в одном файле, но тогда будет сложно обращаться к отдельным значениям внутри файла, а проблема одновременного доступа все равно не будет решена. На самом деле, будет только хуже, так как одновременное обращение нескольких клиентов к файлу, содержащему несколько счетчиков, вероятнее, чем к файлу с одним счетчиком. Так что вам придется реализовывать методы хранения и извлечения для обработки содержимого файла и протоколы установки блокировок, чтобы не допустить конфликта нескольких процессов. Но ведь обо всем этом может позаботиться MySQL! Хранение

счетчиков в базе данных позволяет сосредоточить их в одной таблице, SQL обеспечивает интерфейс для хранения и извлечения, а вопрос с блокировками отпадает сам собой, так как MySQL предоставляет последовательный доступ к таблице, поэтому клиенты не могут мешать друг другу. Более того, в зависимости от способа обработки счетчиков может появиться возможность обновлять счетчик и извлекать новое значение последовательности в одном запросе.

Предположим, что вы хотите хранить счетчики для нескольких страниц. Для этого создайте таблицу, содержащую по строке для каждой обрабатываемой страницы. То есть каждая страница должна иметь уникальный идентификатор, чтобы не перепутать соответствующие счетчики. Идентификаторы можно присваивать любым способом, но проще всего использовать путь страницы в вашем дереве документов. Языки веб-программирования обычно обеспечивают простой доступ к этому пути (в рецепте 18.1 мы уже говорили о том, как его получить). Создаем таблицу `hitcount`:

```
CREATE TABLE hitcount
(
    path    VARCHAR(255) BINARY NOT NULL,
    hits    BIGINT UNSIGNED NOT NULL,
    PRIMARY KEY (path)
);
```

При определении таблицы сделано несколько допущений:

- Ключевое слово `BINARY` в определении столбца `path` делает значения столбца чувствительными к регистру. Это целесообразно для веб-платформы, путевые имена которой чувствительны к регистру, как многие версии UNIX. Для Windows или файловой системы HFS+ под управлением Mac OS X файловые имена не чувствительны к регистру, поэтому можно убрать `BINARY` из определения.
- Столбец `path` имеет максимальную длину в 255 символов, то есть вы не можете задавать более длинные пути страниц. Если вы полагаете, что понадобятся более длинные значения, используйте тип `BLOB` или `TEXT` вместо `VARCHAR`. Но в этом случае индексирование все равно будет возможно только по первым 255 символам, поэтому лучше применить неуникальный индекс, а не `PRIMARY KEY`.
- Механизм работает с одним деревом документов, то есть для случаев, когда ваш веб-сервер используется для обслуживания страниц одного домена. Если вы вводите счетчики посещения на хосте, обслуживающем несколько виртуальных доменов, то можете добавить столбец с именем домена. Это значение доступно как значение `SERVER_NAME`, помещаемое Apache в окружение сценария. Тогда индекс таблицы счетчиков посещаемости будет включать и имя хоста, и путь к странице.

Общая идея, лежащая в основе ведения счетчиков посещаемости, заключается в приращении поля `hits` в записи страницы и извлечении обновленного значения. Это можно сделать при помощи двух таких запросов:

```
UPDATE hitcount SET hits = hits + 1 WHERE path = 'путь к странице';
SELECT hits FROM hitcount WHERE path = 'путь к странице';
```

К сожалению, значение, полученное таким способом, может быть некорректным. Если несколько клиентов запрашивают одну страницу одновременно, то несколько предложений UPDATE может быть создано в очень близкие моменты времени. Поэтому предложения SELECT совсем не обязательно извлекут соответствующее значение hits. Чтобы избежать ошибки, можно использовать транзакцию или заблокировать таблицу hitcount, но это замедлит работу. MySQL предлагает решение, позволяющее каждому клиенту извлекать собственный счетчик вне зависимости от того, сколько обновлений производится одновременно:

```
UPDATE hitcount SET hits = LAST_INSERT_ID(hits+1) WHERE path = 'путь к странице';
SELECT LAST_INSERT_ID();
```

Для обновления счетчика применяется функция LAST_INSERT_ID(*выражение*), о которой мы говорили в рецепте 11.6. Предложение UPDATE находит нужную запись и увеличивает для нее значение счетчика. Использование LAST_INSERT_ID(hits+1) вместо просто hits+1 указывает MySQL на то, что необходимо интерпретировать значение так, как если бы оно относилось к типу AUTO_INCREMENT. Поэтому в следующем запросе его можно извлечь при помощи LAST_INSERT_ID(). Функция LAST_INSERT_ID() возвращает значение, специфичное для соединения, поэтому вы всегда получаете значение, соответствующее обновлению, произведенному в рамках того же соединения. Кроме того, предложению SELECT не нужно обращаться к таблице, поэтому оно работает очень быстро. Можно еще повысить производительность, вообще избавившись от запроса SELECT, что возможно, если ваш API предлагает средство прямого доступа к последнему номеру последовательности. Например, в Perl можно обновить счетчик и получить новое значение в одном запросе:

```
$dbh->do (
    "UPDATE hitcount SET hits = LAST_INSERT_ID(hits+1) WHERE path = ?",
    undef, $page_path);
$hits = $dbh->{mysql_insertid};
```

Однако одна проблема все еще не решена. Что, если страница не входит в таблицу hitcount? Тогда предложение UPDATE не находит записи для изменения, и вы получаете для счетчика значение ноль. Можно потребовать, чтобы каждая страница, содержащая счетчик посещаемости, была зарегистрирована в таблице hitcount перед ее публикацией в Интернете. Более удобным способом является автоматическое создание записи для каждой страницы, которая еще не содержится в таблице. Тогда создатели страниц смогут помещать на них счетчики без предварительной подготовки. Чтобы еще более упростить использование счетчиков, поместим код в функцию, которая принимает в качестве аргумента путь к странице, обрабатывает внутри себя отсутствие записи о странице и возвращает счетчик. Принцип действия функции таков:

```
обновление счетчика
если обновление изменяет строку
```

```
    то извлечь новое значение счетчика
иначе
    вставить запись для страницы, установив счетчик в 1
```

Когда вы впервые запрашиваете счетчик для страницы, операция обновления не изменяет строки, так как информации о странице в таблице еще нет. Функция создает новый счетчик и возвращает значение 1. Для каждого последующего запроса обновление изменяет существующую запись, и функция возвращает соответствующие увеличивающиеся счетчики.

В Perl функция для счетчика посещаемости могла бы выглядеть так (аргументами являются дескриптор базы данных и путь к странице):

```
sub get_hit_count
{
my ($dbh, $page_path) = @_;

    my $rows = $dbh->do (
        "UPDATE hitcount SET hits = LAST_INSERT_ID(hits+1) WHERE path = ?",
        undef, $page_path);
    return ($dbh->{mysql_insertid}) if $rows > 0;       # счетчик увеличивается

    # Если страница не была зарегистрирована в таблице, сделать это
    # и установить счетчик в 1. Использовать IGNORE, если другой клиент
    # пытается сделать то же самое в то же время.

    $dbh->do ("INSERT IGNORE INTO hitcount (path,hits) VALUES(?,1)",
                    undef, $page_path);
    return (1);
}
```

Функция CGI.pm `script_name()` возвращает локальную часть URL, поэтому можно использовать `get_hit_count()` так:

```
my $hits = get_hit_count ($dbh, script_name ());
print p ("This page has been accessed $hits times.");
```

Счетный механизм включает в себя несколько запросов, а транзакции мы не используем, поэтому в алгоритме остается еще одно место – первый запрос страницы, в котором возможны проблемы с конкуренцией. Если несколько клиентов одновременно запрашивают страницу, которая еще не содержится в таблице `hitcount`, каждый из них может создать запрос `UPDATE`, обнаружить, что страница отсутствует и, в результате, создать запрос `INSERT` для регистрации страницы и инициализации счетчика. Алгоритм использует `INSERT IGNORE` для устранения ошибок, если одновременные вызовы сценария пытаются инициализировать счетчик для одной и той же страницы, но в результате все они получают счетчик 1. Стоит ли попробовать исправить положение при помощи транзакций или блокировки таблицы? Для счетчика посещаемости я бы ответил: «Нет». Небольшая потеря точности не является поводом для дополнительных расходов на обработку. В каком-то другом приложении, где точность важнее эффективности, следует предпочесть транзакции, чтобы избежать потерь при подсчете.

PHP-версия функции для счетчика посещаемости будет такой:

```
function get_hit_count ($conn_id, $page_path)
{
    $query = sprintf ("UPDATE hitcount SET hits = LAST_INSERT_ID(hits+1)
                       WHERE path = %s", sql_quote ($page_path));
    if (mysql_query ($query, $conn_id) && mysql_affected_rows ($conn_id) > 0)
        return (mysql_insert_id ($conn_id));

    # Если путь страницы не приведен в таблице, то зарегистрировать ее
    # и установить счетчик в 1. Использовать IGNORE, если другой клиент
    # пытается сделать то же самое в то же время.

    $query = sprintf ("INSERT IGNORE INTO hitcount (path,hits)
                           VALUES(%s,1)", sql_quote ($page_path));
    mysql_query ($query, $conn_id);
    return (1);
}
```

Для использования счетчика вызовите функцию `get_self_path()`, возвращающую путевое имя сценария (рецепт 18.1):

```
$self_path = get_self_path ();
$hits = get_hit_count ($conn_id, $self_path);
print ("<p>This page has been accessed $hits times.</p>\n");
```

В Python функция счетчика будет такой:

```
def get_hit_count (conn, page_path):
    cursor = conn.cursor ()
    cursor.execute ("""
            UPDATE hitcount SET hits = LAST_INSERT_ID(hits+1)
            WHERE path = %s
        """, (page_path,))
    if cursor.rowcount > 0:     # счетчик увеличивается
        count = cursor.insert_id ()
        cursor.close ()
        return (count)

# Если путь страницы не приведен в таблице, то зарегистрировать ее
# и установить счетчик в 1. Использовать IGNORE, если другой клиент
# пытается сделать то же самое в то же время.

    cursor.execute ("""
            INSERT IGNORE INTO hitcount (path,hits) VALUES(%s,1)
        """, (page_path,))
    cursor.close ()
    return (1)
```

И используется она так:

```
self_path = os.environ["SCRIPT_NAME"]
count = get_hit_count (conn, self_path)
print "<p>This page has been accessed %d times.</p>" % count
```

Дистрибутив recipes содержит сценарии, демонстрирующие работы счетчиков посещаемости на Perl, PHP и Python в каталоге *apache*. JSP-версия хранится в каталоге tomcat. Установите любые из них в вашем дереве веб-документов, вызовите несколько раз и посмотрите, как увеличивается счетчик. (Сначала необходимо создать таблицу hitcount, а также таблицу hitlog, описываемую в рецепте 18.13. Обе таблицы можно создать при помощи сценария *hits.sql* из каталога tables.)

18.13. Журнал доступа к веб-странице

Задача

Вы хотите знать о странице нечто большее, чем просто количество обращений к ней, например, время доступа и хост пользователя, запросившего страницу.

Решение

Ведите журнал доступа, а не просто счетчик посещаемости.

Обсуждение

Таблица hitcount хранит только счетчик посещаемости каждой зарегистрированной в ней страницы. Если вы хотите учитывать и другую информацию, используйте иной подход. Предположим, что вам нужно отслеживать клиентский хост и время обращения к странице для каждого запроса. В этом случае для каждой страницы нужен не просто счетчик, а журнал. Но можно продолжать хранить и счетчики, используя составной индекс, объединяющий путь страницы и столбец последовательности AUTO_INCREMENT:

```
CREATE TABLE hitlog
(
    path    VARCHAR(255) BINARY NOT NULL,
    hits    BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
    t       TIMESTAMP,
    host    VARCHAR(64),
    PRIMARY KEY (path,hits)
);
```

Для вставки новой записи используйте такой запрос:

```
INSERT INTO hitlog (path, host) VALUES(значение_пути,значение_хоста);
```

Например, для страницы JSP обращения можно регистрировать так:

```
<c:set var="host">
    <%= request.getRemoteHost () %>
</c:set>
<c:if test="${empty host}">
    <c:set var="host">
        <%= request.getRemoteAddr () %>
    </c:set>
```

```
</c:if>
<c:if test="${empty host}">
    <c:set var="host">
        UNKNOWN
    </c:set>
</c:if>

<sql:update dataSource="${conn}">
    INSERT INTO hitlog (path, host) VALUES(?,?)
    <sql:param><%= request.getRequestURI () %></sql:param>
    <sql:param value="${host}" />
</sql:update>
```

Таблица `hitlog` обладает рядом полезных свойств:

- Время обращения автоматически записывается в столбец `t` типа `TIMESTAMP` при добавлении новой записи.
- Столбец `path` связывается со столбцом `hits` типа `AUTO_INCREMENT`, благодаря чему значение счетчика для определенной страницы автоматически увеличивается при каждой вставке новой записи для такого пути. Счетчики ведутся отдельно для каждого отличного значения `path` (подробная информация о работе многостолбцовых последовательностей приведена в рецепте 11.14).
- Нет необходимости в проверке существования счетчика для определенной страницы, так как запись вставляется в таблицу при любом, а не только при первом обращении к странице.
- Если вы хотите определить текущие счетчики для каждой страницы, выберите записи для всех различных значений `path` с наибольшим значением `hits`:

```
SELECT path, MAX(hits) FROM hitlog GROUP BY path;
```

18.14. Ведение журнала Apache с помощью MySQL

Задача

Вы не хотите использовать MySQL для ведения журнала доступа к нескольким страницам, как это делалось в рецепте 18.13. Вы хотите записывать обращения ко всем страницам, при этом не хотите явно помещать на каждую страницу механизм протоколирования.

Решение

Сообщите Apache, что для протоколирования доступа ему следует использовать MySQL.

Обсуждение

Применение MySQL для Web не ограничивается формированием и обработкой страниц. Вы можете использовать MySQL для того, чтобы помочь работе

самого веб-сервера. Например, большая часть серверов Apache ведут журнал веб-запросов в файле. Но можно вместо этого отправлять записи журнала в программу, из которой они могут быть записаны куда угодно, например в базу данных. Если записи хранятся в базе данных, а не в файле, журнал становится более структурированным, и к нему можно применять методы анализа SQL. Средства анализа журнального файла могут создаваться для обеспечения некоторой гибкости, но зачастую все ограничивается выбором и фильтрацией итогов. Сложно просить у программы вывести информацию, для предоставления которой она не создавалась. Если же записи журнала хранятся в таблице, гибкость увеличивается. Хотите получить какой-то определенный отчет? Напишите соответствующие предложения SQL. Для вывода отчета в указанном формате выдавайте запросы в API и пользуйтесь возможностями вашего языка программирования по организации вывода.

Разнеся формирование записи журнала и ее хранения в разные процессы, вы обеспечиваете дополнительную гибкость. Появляется возможность, например, отправки журналов с нескольких веб-серверов на один сервер MySQL или отправки разных журналов, созданных одним веб-сервером, разным серверам MySQL.

В этом разделе рассказано о настройке протоколирования веб-запросов Apache в MySQL и рассмотрено несколько полезных итоговых запросов.

Настройка протоколирования в базе данных

Протоколирование Apache управляется директивами конфигурационного файла *httpd.conf*. Например, стандартно используются директивы `LogFormat` и `CustomLog`:

```
LogFormat "%h %l %u %t \"%r\" %>s %b" common
CustomLog /usr/local/apache/logs/access_log common
```

Строка `LogFormat` определяет формат записей журнала и дает ему мнемоническое имя `common`. Директива `CustomLog` указывает на то, что строки в таком формате должны записываться в файл *access_log* каталога Apache *logs*. Для ведения протокола в MySQL вместо файла используйте следующую процедуру:[1]

- Определите, какие значения вы хотите хранить, и создайте таблицу с соответствующими столбцами.
- Напишите программу, которая бы читала строки журнала Apache и записывала их в базу данных.
- Задайте строку `LogFormat`, указывающую, как записывать строки журнала в формате, необходимом программе, и директиву `CustomLog`, которая сообщает Apache, что журнал следует записывать в программе, а не в файле.

[1] Если вы используете вместо `LogFormat` и `CustomLog` директиву протоколирования `TransferLog`, вам придется соответствующим образом изменить приведенные рекомендации.

Предположим, что вы хотите записывать дату и время каждого запроса, хост запрашивающего клиента, метод запроса и путь URL, код состояния, количество переданных байтов и пользовательского агента (обычно название броузера или программы-паука). Таблицу, содержащую столбцы для всех этих значений, можно создать так:

```
CREATE TABLE httpdlog
(
    dt      DATETIME NOT NULL,              # дата запроса
    host    VARCHAR(255) NOT NULL,          # хост клиента
    method  VARCHAR(4) NOT NULL,            # метод запроса (GET, PUT, и т. д.)
    url     VARCHAR(255) BINARY NOT NULL,   # URL
    status  INT NOT NULL,                   # статус запроса
    size    INT,                            # количество переданных байтов
    agent   VARCHAR(255)                    # пользовательский агент
);
```

Большинство строковых столбцов относятся к типу `VARCHAR` и не чувствительны к регистру. Исключением является `url`, объявленный как двоичная строка, что соответствует серверу, работающему в системе, имена файлов которой чувствительны к регистру. Если вы используете сервер, для которого регистр URL не имеет значения, то можете убрать из определения слово `BINARY`.

В приведенное выше определение таблицы `httpdlog` не входит ни один индекс. Необходимо их добавить, так как в противном случае все итоговые запросы будут выполняться все медленнее при увеличении объема таблицы. Выбор столбцов для индексирования зависит от того, какие запросы вы планируете выбирать для исследования содержимого таблицы. Например, для запросов, анализирующих распределение значений клиентских хостов, индекс для столбца `host` будет предпочтительнее.

Теперь нужно написать программу обработки строк журнала, сформированного Apache и вставки их в таблицу `httpdlog`. Сценарий *httpdlog.pl* устанавливает соединение с сервером MySQL, затем в цикле считывает строки ввода. Он разбирает каждую строку на значения столбцов и вставляет результат в базу данных. Когда Apache завершает работу, канал к программе протоколирования закрывается. Тогда *httpdlog.pl* видит на входе конец файла и завершает цикл, отключается от MySQL и заканчивает работу.

```
#! /usr/bin/perl -w
# httpdlog.pl - протоколирование запросов Apache в таблице httpdlog

use strict;
use lib qw(/usr/local/apache/lib/perl);
use Cookbook;

my $dbh = Cookbook::connect ();
my $sth = $dbh->prepare (qq{
        INSERT INTO httpdlog (dt,host,method,url,status,size,agent)
        VALUES (?,?,?,?,?,?,?)
    });
```

```
while (<>)  # цикл чтения ввода
{
    chomp;
    my ($dt, $host, $method, $url, $status, $size, $agent)
                                                = split (/\t/, $_);
    # заменить "-" на NULL в некоторых столбцах
    $size = undef if $size eq "-";
    $agent = undef if $agent eq "-";
    $sth->execute ($dt, $host, $method, $url, $status, $size, $agent);
}

$dbh->disconnect ();
exit (0);
```

Установите сценарий *httpdlog.pl* в каталог, где Apache сможет его обнаружить. В моей системе корневым каталогом Apache является */usr/local/apache*, поэтому разумно назначить каталогом установки */usr/local/apache/bin*. Путь к данному каталогу нам скоро понадобится – при создании директивы `CustomLog`, указывающей Apache на необходимость протоколирования в программе.

Сценарий *httpdlog.pl* считает, что строки ввода содержат значения столбцов `httpdlog`, разделенные символами табуляции (для простоты разбиения строк ввода), так что Apache должен записывать журнал именно в таком формате. Спецификаторы поля `LogFormat` для вывода соответствующих значений таковы:

`%{%Y-%m-%d %H:%M:%S}`

Дата и время запроса в формате даты MySQL `DATETIME`.

`%h`

Хост запрашивающего клиента.

`%m`

Метод запроса (`GET`, `POST` и т. д.).

`%U`

Путь URL.

`%>s`

Код состояния.

`%b`

Количество переданных байт.

`%{User-Agent}i`

Пользовательский агент.

Для определения формата протоколирования с именем `mysql`, выводящего перечисленные значения, разделенные табуляциями, добавим в файл *httpd.conf* следующую директиву `LogFormat`:

```
LogFormat "%{%Y-%m-%d %H:%M:%S}t\t%h\t%m\t%U\t%>s\t%b\t%{User-Agent}i" mysql
```

Уже почти все готово. У нас есть таблица, читающая ее программа и формат `mysql` для создания записей журнала. Остается лишь сообщить Apache о том, что следует передавать записи в сценарий *httpdlog.pl*. Однако пока вы не уверены в корректности формата вывода и в том, что программа может соответствующим образом обрабатывать записи журнала, преждевременно сообщать Apache о ведении журнала в программе. Для упрощения тестирования и отладки выведем протокол Apache в формате `mysql` в файл. Тогда, посмотрев на файл, вы сможете проверить формат вывода и использовать его как ввод для *httpdlog.pl*, чтобы проконтролировать корректность работы программы. Укажем Apache на необходимость записи строк в формате `mysql` в файл *test_log* каталога протоколов при помощи директивы `CustomLog`:

```
CustomLog /usr/local/apache/logs/test_log mysql
```

Перезапускаем Apache, чтобы новые директивы протоколирования вступили в силу. После того как ваш веб-сервер получит несколько запросов, посмотрите на файл *test_log*. Проверьте, совпадает ли его содержимое с ожидаемым, затем передайте в *httpdlog.pl*. Если вы находитесь в каталоге Apache *logs*, и каталоги *bin* и *logs* расположены внутри корневого каталога Apache, то команда выглядит так:

```
% ../bin/httpdlog.pl test_log
```

После завершения работы *httpdlog.pl* исследуйте таблицу `httpdlog`, чтобы проверить, все ли в ней хорошо. Если вас все устраивает, сообщите Apache об отправке записей журнала непосредственно в *httpdlog.pl*, изменив директиву `CustomLog` так:

```
CustomLog "|/usr/local/apache/bin/httpdlog.pl" mysql
```

Символ | в начале имени файла указывает Apache на то, что *httpdlog.pl* – это программа, а не файл. Перезапустите Apache, и новые записи будут появляться таблице `httpdlog` по мере обращения посетителей к вашему сайту.

Ничто из сделанного к этому моменту не меняет протоколирования, осуществлявшегося изначально. Например, если раньше вы вели протокол в файле *access_log*, ничего не изменилось. То есть Apache будет отправлять записи и в исходный файл журнала, и в MySQL. Если вы этого и хотели, отлично, Apache не интересует, в скольких направлениях ведется протоколирование. Но в этом случае используется больше дискового пространства. Для отмены протоколирования в файле закомментируйте исходную директиву `CustomLog`, поместив перед ней символ `#`, затем перезапустите Apache.

Анализ файла журнала

Теперь Apache ведет протокол в базе данных, но что можно сделать с этой информацией? Все зависит от того, какие сведения вам интересны. Приведем ряд вопросов, на которые без труда отвечает MySQL:

- Сколько записей содержится в журнале запросов?

```
SELECT COUNT(*) FROM httpdlog;
```

- Со скольких различных хостов отправлялись запросы?

```
SELECT COUNT(DISTINCT host) FROM httpdlog;
```

- Сколько разных страниц запросили клиенты?

```
SELECT COUNT(DISTINCT url) FROM httpdlog;
```

- Какие страницы входят в десятку наиболее популярных?

```
SELECT url, COUNT(*) AS count FROM httpdlog
GROUP BY url ORDER BY count DESC LIMIT 10;
```

- Сколько запросов было получено для этих бесполезных файлов *favicon.ico*, которые любят проверять некоторые броузеры?

```
SELECT COUNT(*) FROM httpdlog WHERE url LIKE '%/favicon.ico%';
```

- Какой диапазон дат охватывает журнал?

```
SELECT MIN(dt), MAX(dt) FROM httpdlog;
```

- Сколько запросов было получено в каждый из дней?

```
SELECT FROM_DAYS(TO_DAYS(dt)) AS day, COUNT(*) FROM httpdlog GROUP BY day;
```

  Ответ на этот вопросы требует выделения составляющей времени дня из значений `dt` для группировки запросов по указанной дате. Запрос использует функции `TO_DAYS()` и `FROM_DAYS()` для преобразования значений `DATETIME` в `DATE`. Однако если вы собираетесь создать множество запросов, использующих только составляющую даты значений `dt`, будет эффективнее создать таблицу `httpdlog` с отдельными столбцами `DATE` и `TIME`, изменить директиву `LogFormat` для вывода даты и времени в виде отдельных значений и соответствующим образом изменить *httpdlog.pl*. Тогда можно будет работать непосредственно с датами запросов, не отбрасывая время. Для еще большего повышения производительности можно индексировать столбец дат.

- Как запросы распределяются по времени суток?

```
SELECT HOUR(dt) AS hour, COUNT(*) FROM httpdlog GROUP BY hour;
```

- Каково среднее количество запросов, получаемых за день?

```
SELECT COUNT(*)/(TO_DAYS(MAX(dt)) - TO_DAYS(MIN(dt)) + 1) FROM httpdlog;
```

  Числитель – общее количество запросов в таблице. Знаменатель – количество дней, для которых есть записи.

- Какой самый длинный URL сохранен в таблице?

```
SELECT MAX(LENGTH(url)) FROM httpdlog;
```

  Если столбец `url` определен как `VARCHAR(255)`, а запрос выводит значение `255`, то вероятно, некоторые значения URL оказались слишком длинными для столбца и были усечены с конца. Во избежание подобных ситуаций можно преобразовать столбец в `BLOB` или `TEXT` (в зависимости от того, интересует ли вас чувствительность значений к регистру). Например, если вы

хотите хранить чувствительные к регистру значения длиной до 65 535 символов, измените столбец `url` так:

```
ALTER TABLE httpdlog MODIFY url BLOB NOT NULL;
```

- Каково общее количество отправленных байт и среднее количество байт для запроса?

```
SELECT
    COUNT(size) AS requests,
    SUM(size) AS bytes,
    AVG(size) AS 'bytes/request'
FROM httpdlog;
```

Запрос использует `COUNT(size)`, а не `COUNT(*)` для подсчета только запросов с значением `size` не-`NULL`. (Если клиент запрашивает страницу дважды, сервер может ответить на второй запрос отправкой заголовка, сообщающего о том, что страница не изменилась с момента последнего посещения. В такой ситуации запись журнала для запроса будет содержать значение `NULL` в столбце `size`.)

- Каким был трафик для каждой из разновидностей файлов (по расширению, например *.html*, *.jpg* или *.php*)?

```
SELECT
    SUBSTRING_INDEX(SUBSTRING_INDEX(url,'?',1),'.',-1) AS extension,
    COUNT(size) AS requests,
    SUM(size) AS bytes,
    AVG(size) AS 'bytes/request'
FROM httpdlog
WHERE url LIKE '%.%'
GROUP BY extension;
```

Инструкция `WHERE` выбирает только значения `url`, включающие точку, чтобы исключить из рассмотрения путевые имена без расширения имени файла. Для извлечения расширения из списка столбцов вывода внутренний вызов `SUBSTRING_INDEX()` удаляет строку параметров, находящуюся в конце URL (если она присутствует), и оставляет все остальное. (Тем самым значение типа `/cgi-bin/script.pl?id=43` превращается в `/cgi-bin/script.pl`. Если у значения не было параметрической составляющей, `SUBSTRING_INDEX()` возвращает всю строку.) Внешний вызов `SUBSTRING_INDEX()` удаляет все вплоть до самой правой точки результата (включая ее), оставляя только собственно расширение.

Еще о протоколировании

Я выбрал простой способ связывания Apache с MySQL – создание небольшого сценария, взаимодействующего с MySQL и сообщающего Apache о необходимости ведения журнала не в файле, а в сценарии. Этот способ хорошо работает, если все запросы регистрируются в одном файле, но, естественно, подходит не для любой конфигурации Apache. Например, если в файле *httpd.conf* определены виртуальные серверы, вы можете указать для каждого из них

отдельные директивы CustomLog. Для протоколирования их всех в MySQL можно изменить каждую директиву, указав на запись в *httpdlog.pl*, но тогда для каждого виртуального сервера будет работать свой процесс протоколирования. Возникает два вопроса:

- Как сопоставить записи журнала с соответствующим виртуальным сервером? Можно создать отдельную таблицу для каждого сервера и изменить сценарий *httpdlog.pl* так, чтобы он принимал аргумент, определяющий, какую из таблиц следует использовать. Другой способ заключается в добавление столбца virt_host в таблицу httpdlog и изменении *httpdlog.pl* так, чтобы он принимал аргумент имени хоста, указывающий имя сервера для записи в столбец virt_host.
- Действительно ли нужно столько работающих процессов *httpdlog.pl*? Если у вас есть несколько виртуальных серверов, то можно подумать об использовании модуля протоколирования, устанавливаемого непосредственно в Apache. Некоторые из них разрешают мультиплексное протоколирование для нескольких виртуальных хостов в рамках одного соединения с сервером базы данных, что снижает объем потребляемых ресурсов.

Ведение журнала не в файле, а в базе данных позволяет использовать для его анализа всю мощь MySQL, но при этом не избавляет от необходимости думать об организации пространства для хранения. Веб-серверы могут использоваться весьма активно, а записи журнала занимают место вне зависимости от того, в файле или в базе данных они хранятся. Одним из способов экономии пространства является удаление устаревших записей. Например, чтобы удалить из журнала записи более чем годичной давности, периодически выполняйте такой запрос:

```
DELETE FROM httpdlog WHERE dt < DATE_SUB(NOW(),INTERVAL 1 YEAR);
```

Еще одна возможность заключается в архивировании старых записей в сжимаемых таблицах. (Для этого необходимо работать с таблицами MyISAM, которые можно сжимать при помощи функции myisampack.) Например, при переходе от сентября 2001 года к октябрю того же года вы знаете, что сентябрьских записей Apache больше не сгенерирует, так что их можно перенести в другую таблицу, которая останется неизменной. Создайте таблицу httpdlog_2001_09 с такой же структурой, как у httpdlog (включая индексы). Затем перенесите сентябрьские записи журнала из httpdlog в httpdlog_2001_09, используя следующие запросы:

```
INSERT INTO httpdlog_2001_09
    SELECT * FROM httpdlog
    WHERE dt >= '2001-09-01' AND dt < '2001-10-01';
DELETE FROM httpdlog
    WHERE dt >= '2001-09-01' AND dt < '2001-10-01';
```

Наконец, запустите myisampack для httpdlog_2001_09 с тем, чтобы сжать ее и сделать доступной только для чтения.

Недостатком этого подхода можно назвать распределение записей журнала по множеству таблиц. Если вы хотите рассматривать таблицы как единую

сущность, чтобы можно было выполнять запрос ко всему множеству записей журнала, создайте таблицу MERGE, которая будет включать их все. Предположим, что есть набор таблиц, содержащий текущую таблицу и таблицы для промежутка времени с сентября 2001 по апрель 2002 года. Предложение, создающее таблицу MERGE, будет выглядеть так:

```
CREATE TABLE httpdlog_all
(
    dt      DATETIME NOT NULL,              # дата запроса
    host    VARCHAR(255) NOT NULL,          # хост клиента
    method  VARCHAR(4) NOT NULL,            # метод запроса(GET, PUT и т. д.)
    url     VARCHAR(255) BINARY NOT NULL,   # URL
    status  INT NOT NULL,                   # статус запроса
    size    INT,                            # количество переданных байтов
    agent   VARCHAR(255)                    # пользовательский агент
)
TYPE = MERGE
UNION = (httpdlog, httpdlog_2001_09, httpdlog_2001_10, httpdlog_2001_11,
httpdlog_2001_12, httpdlog_2002_01, httpdlog_2002_02, httpdlog_2002_03,
httpdlog_2002_04);
```

Инструкция `UNION` должна указывать имена всех таблиц, включаемых в таблицу `MERGE`. Обратите внимание на то, что вам придется удалять и заново создавать определение `httpdlog_all` при каждом формировании новой статической таблицы журнала для очередного месяца. (Кроме того, если вы добавляете индекс, нужно добавить его во все отдельные таблицы, а затем пересоздать таблицу MERGE, включив и в нее определение индекса.)

Отчеты, получаемые по таблице `httpdlog_all`, будут принимать в расчет все записи журнала. Для составления ежемесячных отчетов просто обратитесь к соответствующей отдельной таблице.

Что касается объема дискового пространства, расходуемого на протоколирование веб-активности, помните о том, что если для сервера MySQL включено протоколирование запросов, то каждый запрос будет записан в таблицу `httpdlog`, а также в журнал запросов. То есть может случиться так, что дисковое пространство будет исчезать быстрее, чем вы думали, поэтому разумно применять для сервера MySQL какой-либо способ сокращения объема журнала (установку строка действия записи или ротацию журналов).

19

Управление веб-сеансами с помощью MySQL

19.0. Введение

Во многих веб-приложениях процесс взаимодействия с пользователем состоит из ряда последовательных запросов, поэтому возникает необходимость хранения промежуточной информации. Последовательность взаимосвязанных запросов называется сеансом (session). Сеансы полезны в таких задачах, как регистрация пользователей с последующим отслеживанием выполняемых ими запросов, управление многоэтапными процедурами онлайн-заказов, поэтапное получение информации от пользователя (когда очередной вопрос выбирается в зависимости от содержания предыдущих ответов), хранение пользовательских настроек от посещения к посещению. К сожалению, в протоколе HTTP отсутствует понятие состояния. Это означает, что веб-серверы рассматривают каждый запрос изолированно от всех прочих – если только не предпринять специальные меры.

В этой главе рассказывается, как сохранять информацию на протяжении нескольких запросов в тех приложениях, где очередной запрос должен учитывать результаты предшествующих. Предлагаемые здесь подходы легко обобщаются для самых разнообразных веб-приложений, требующих хранения данных о своем состоянии.

Проблемы управления сеансами

Некоторые методы управления сеансами основываются на информации, хранимой на клиенте. Один из способов хранения данных на клиентской стороне предполагает использование cookies, которые представляют собой информацию, передаваемую от сервера клиенту и обратно в заголовках специальных запросов и ответов на них. В начале сеанса приложение создает и посылает клиенту данные, содержащие начальную информацию, подлежащую сохранению. Клиент возвращает эти данные серверу всякий раз, когда посылает ему очередной запрос, и по этим данным серверное приложение определяет принадлежность запроса текущему сеансу. На каждом из этапов

сеанса приложение определяет по данным cookies состояние клиента. Для изменения состояния сеанса приложение отправляет клиенту новый файл cookies, содержащий обновленную информацию, для замены старого. Такой механизм обеспечивает хранение данных на протяжении нескольких запросов и в то же время позволяет приложению обновлять их по мере необходимости. Cookies просты в использовании, но имеют ряд недостатков. Например, клиент может изменить их содержимое и таким образом вызвать нежелательные изменения в работе приложения. Эта уязвимость характерна и для других способов идентификации сеансов, основанных на хранении данных на стороне клиента.

Альтернативой хранению данных на стороне клиента может быть запоминание состояния сеанса на серверной стороне. При таком подходе информация о действиях клиента хранится где-то на сервере: в файле, разделяемой памяти или базе данных. На клиентской стороне хранится только уникальный идентификатор, сгенерированный сервером и отправленный им клиенту в начале сеанса. Клиент посылает это значение на сервер в каждом последующем запросе, чтобы сервер мог сопоставить его соответствующему сеансу. Обычно идентификатор сеанса пересылается с помощью cookies или включается в URL запроса. (Последний способ подходит для тех клиентов, у которых cookies запрещены.) Сервер извлекает идентификатор из cookies или из URL.

Хранить данные сеанса на сервере безопаснее, чем на клиенте, так как состояние сеанса полностью контролируется приложением. Единственное значение, имеющееся у клиента, – идентификатор сеанса, поэтому клиент не может изменить его характеристики без разрешения сервера. Правда, у клиента остается возможность вернуть измененный идентификатор, но если идентификаторы уникальны и выбираются из достаточно большого диапазона, очень мала вероятность того, что злоумышленник сможет угадать правильное значение ID другого сеанса.[1]

При использовании серверных методов управления сеансами их состояние обычно хранится в постоянной памяти, такой как файл или база данных. Хранение в базе данных отличается от хранения в файлах тем, что позволяет избавиться от неразберихи, создаваемой многочисленными файлами сеансов, более того, один и тот же сервер MySQL может хранить данные о сеансах нескольких веб-серверов. Если вам это подходит, то, применяя описанные здесь способы, вы сможете с помощью MySQL управлять сеансами в своих приложениях. В этой главе на примере трех из используемых нами API показано, как с помощью базы данных реализовать управление сеансами на сервере:[2]

---

[1] Если вас беспокоит, что другие клиенты, просматривая сетевой трафик, могут перехватить актуальные идентификаторы сеансов, – используйте безопасное соединение, например, SSL. Но эта тема выходит за пределы материала нашей книги.

[2] Примеры на Python отсутствуют, потому что мне не удалось найти автономный модуль для управления сеансами, подходящий для нашего обсуждения, а писать новый модуль с нуля мне не хотелось. Если вы пишете на Python приложения, требующие поддержки сеансов, обратите внимание на такие инструментальные средства, как Zope, WebWare или Albatross.

- В Perl имеется модуль `Apache::Session`, реализующий почти все, что необходимо для поддержки сеансов. Он может хранить данные сеансов в файлах или в любой из поддерживаемых им баз данных, включая MySQL, PostgreSQL и Oracle.
- В PHP встроенная поддержка сеансов появилась в версии 4. По умолчанию используются временные файлы, но гибкая реализация этого механизма позволяет приложению применять собственные средства хранения данных сеансов. Таким образом, можно подключить модуль, записывающий данные в MySQL.
- Для приложений на Java, выполняющихся в среде Tomcat, веб-сервер предоставляет собственную поддержку сеансов. Все, что нужно сделать, – это сконфигурировать сервер для хранения данных сеансов в MySQL. В прикладных программах не надо предпринимать специальных усилий для использования этой возможности, поэтому на прикладном уровне изменения не требуются.

Подходы к реализации поддержки сеансов в перечисленных API существенно различаются. В Perl поддержка на уровне языка отсутствует, поэтому в сценарий надо явно включать модуль `Apache::Session`. В PHP имеется встроенный менеджер сеансов. Сценарии могут пользоваться им без какой-либо предварительной подготовки, если применяемый по умолчанию метод хранения данных в файлах не вызывает возражений. Для реализации альтернативных методов (например, хранения данных сеансов в MySQL) приложение должно предоставить менеджеру сеансов собственные процедуры. Еще один подход реализован в Java-приложениях, выполняющихся в среде Tomcat, которая берет на себя многие функции управления сеансами, включая хранение их данных. Приложениям не нужно беспокоиться о том, где хранится информация.

Несмотря на различия в реализации, управление сеансами состоит из ряда общих задач:

- Определение того, предоставляет ли клиент идентификатор сеанса. Если нет, необходимо сгенерировать уникальное значение и отправить его клиенту. Некоторые менеджеры сеансов могут выполнять обмен идентификаторами между сервером и клиентом автоматически. Это относится к PHP, а также к Tomcat для программ на Java. Модуль Perl `Apache::Session` оставляет обмен идентификаторами на усмотрение разработчика.
- Сохранение данных сеанса для использования в последующих запросах и получение данных, записанных предшествующими запросами. Сюда входят все необходимые действия над данными сеанса: приращение счетчика, проверка запроса на регистрацию, обновление покупательской корзины и т. п.
- Закрытие сеанса, когда он больше не нужен. Некоторые менеджеры сеансов предусматривают их автоматическое закрытие после заданного периода бездействия. Сеансы могут быть закрыты и явно при получении запроса на окончание работы (например, когда клиент послал команду выхода).

В этом случае менеджер удаляет запись о сеансе и, возможно, дает команду клиенту на аналогичное действие. Если клиент посылает идентификатор сеанса с помощью cookies, приложение должно сообщить ему о необходимости удаления данного файла cookies. Иначе клиент может продолжать его использовать, несмотря на его неактуальность.

Еще одно общее свойство менеджеров сеансов заключается в том, что они почти не накладывают ограничений на содержание записей о сеансах. Как правило, в них можно хранить практически любые данные, будь то скаляры, массивы или объекты. Для облегчения записи и извлечения данных сеансов менеджеры обычно сериализуют данные (преобразуют их в кодированное скалярное строковое значение) при записи и десериализуют при извлечении. Как правило, вам не надо заботиться об операциях сериализации/десериализации при обращении к процедурам хранения. Достаточно убедиться в том, что для хранения сериализованных строк выделено достаточно места. С точки зрения реализации этого механизма на MySQL это означает, что столбцы должны иметь тип `BLOB` или `TEXT`.

В оставшейся части главы для каждого из API приведены сценарии, основанные на сеансах. Каждый их них выполняет две задачи: ведет счетчик запросов, полученных в течение текущего сеанса, и записывает временную метку для каждого запроса. Этим иллюстрируются запись и извлечение как скалярного значения (счетчика), так и нескалярного (массив, содержащий временную метку). Взаимодействие с пользователем минимизировано: повторная загрузка страницы приводит к отправке очередного запроса, что позволяет обойтись минимумом кода.

Приложения, работающие с сеансами, часто предоставляют пользователю возможность явного выхода и закрытия сеанса. В рассматриваемых сценариях реализована некоторая разновидность «выхода», основанная на неявном механизме: установлен лимит в 10 запросов за сеанс. При каждом обращении к сценарию он проверяет, достиг ли счетчик предельного значения, и, если это так, удаляет данные текущего сеанса. В результате при следующем запросе данные о сеансе отсутствуют, и сценарий создает новый сеанс.

Сценарии для Perl и PHP находятся в каталоге *apache* дистрибутива `recipes`, модуль поддержки сеансов для PHP расположен в каталоге *lib*, а примеры на JSP помещены в каталог *tomcat*. SQL-сценарии создания таблиц для хранения сеансов находятся в каталоге *tables*. Как и раньше, таблицы сеансов создаются в базе данных `cookbook` и доступны под той же учетной записью MySQL, что и остальные таблицы этой книги. Если вы не хотите смешивать управление сеансами и работу с остальными таблицами базы данных `cookbook`, подумайте о том, чтобы создать отдельную базу данных с собственной учетной записью. Особенно это касается сервера Tomcat, в котором управление сеансами выполняется вне уровня приложений. Вероятно, вы не захотите, чтобы Tomcat хранил свои данные в «вашей» базе данных, поэтому создайте для него отдельную.

19.1. Хранение сеансов в MySQL: приложения на Perl

Задача

Вы хотите встроить поддержку сеансов в сценарии на Perl.

Решение

Модуль `Apache::Session` предлагает простой способ работы с различными типами хранения, включая использование MySQL.

Обсуждение

Простой в использовании модуль `Apache::Session` предназначен для хранения информации о состоянии на протяжении нескольких веб-запросов. Вопреки своему названию он не связан с сервером Apache и может использоваться и в другом контексте, например, для запоминания состояния при последовательных вызовах сценария из командной строки. С другой стороны, в `Apache::Session` отсутствуют средства для работы с идентификаторами сеансов (отправка идентификатора клиенту в ответе на его первый запрос и получение его в последующих запросах). В приведенном ниже примере приложение использует cookies для передачи идентификатора сеанса в предположении, что их использование разрешено на клиенте.

Установка модуля Apache::Session

Если у вас нет модуля `Apache::Session`, то можно получить его из архива CPAN (по адресу *http://cpan.perl.org*). Установка не вызывает затруднений, хотя, возможно, вам придется предварительно установить несколько модулей, используемых `Apache::Session`. (В процессе установки `Apache::Session` сам сообщит вам, каких модулей не хватает.) После того, как все будет установлено, создайте таблицу, в которой будут храниться записи сеансов. Спецификация таблицы имеется в документации к модулю, для просмотра которой используйте команду:

```
% perldoc Apache::Session::Store::MySQL
```

Таблица может располагаться в любой базе данных (мы будем использовать `cookbook`), но должна называться `sessions` и иметь такую структуру:

```
CREATE TABLE sessions
(
    id          CHAR(32) NOT NULL,  # идентификатор сеанса
    a_session   BLOB,               # дата сеанса
    PRIMARY KEY (id)
);
```

В столбце `id` хранятся сгенерированные модулем идентификаторы сеансов в формате 32-символьных строк, кодированных по алгоритму MD5. Столбец

a_session содержит дату сеанса в форме сериализованной строки. Модуль Apache::Session пользуется модулем Storable для сериализации и десериализации данных.

Интерфейс модуля Apache::Session

Чтобы использовать таблицу sessions в сценарии, включите в него модуль поддержки MySQL:

```
use Apache::Session::MySQL;
```

Информация о сеансах представлена в Apache::Session с помощью хеша. Механизм tie Perl используется для сопоставления операциям хеша методов хранения и извлечения, используемых менеджером хранения. Для открытия сеанса необходимо объявить переменную хеша и передать ее в tie. Другими аргументами tie являются имя модуля поддержки сеансов, идентификатор сеанса и информация об используемой базе данных. Есть два способа задания соединения с базой данных. Во-первых, можно передать ссылку на хеш, содержащий параметры соединения:

```
my %session;
tie %session,
    "Apache::Session::MySQL",
    $sess_id,
    {
        DataSource => "DBI:mysql:host=localhost;database=cookbook",
        UserName => "cbuser",
        Password => "cbpass",
        LockDataSource => "DBI:mysql:host=localhost;database=cookbook",
        LockUserName => "cbuser",
        LockPassword => "cbpass"
    };
```

Тогда Apache::Session использует параметры для установки собственного соединения с MySQL, которое закрывается при закрытии или уничтожении сеанса. Во-вторых, можно передать дескриптор уже открытого соединения с базой данных (в данном случае – $dbh):

```
my %session;
tie %session,
    "Apache::Session::MySQL",
    $sess_id,
    {
        Handle => $dbh,
        LockHandle => $dbh
    };
```

Если вы передаете дескриптор открытого соединения, то Apache::Session оставляет его открытым, когда вы завершаете или уничтожаете сеанс, считая, что дескриптор используется с другими целями где-то в сценарии. Когда работа завершена, вам следует самостоятельно закрыть соединение.

Аргумент $sess_id, передаваемый в tie, представляет идентификатор сеанса. Его значением должно быть или undef для начала нового сеанса, или идентификатор, соответствующий существующей записи о сеансе (тогда значение должно совпадать со столбцом id некоторой существующей записи таблицы sessions).

После того как сеанс открыт, можно получить доступ к его содержимому. Например, после открытия нового сеанса вы захотите определить его идентификатор, чтобы отправить его клиенту. Значение идентификатора сеанса можно получить так:

```
$sess_id = $session{_session_id};
```

Имена элементов хеша сеанса, которые начинаются с символа подчеркивания (например, _session_id), зарезервированы Apache::Session для внутреннего использования. Помня об этом ограничении, вы можете использовать любые придуманные вами имена для хранения значений сеанса. Например, вы можете поддерживать скалярное значение счетчика так (счетчик инициализируется для нового сеанса, увеличивается и извлекается для отображения):

```
$session{count} = 0 if !exists ($session{count});   # инициализация счетчика
++$session{count};                                  # увеличение счетчика
print "counter value: $session{count}\n";           # вывод значения
```

Для сохранения в записи о сеансе нескалярного значения, такого как массив или хеш, будем хранить ссылку на него:

```
$session{my_array} = \@my_array;
$session{my_hash} = \%my_hash;
```

В этом случае изменения, сделанные в @my_array или %my_hash перед закрытием сеанса, будут отражены в содержимом сеанса. Для сохранения в сеансе независимой копии массива или хеша, которая не будет меняться при изменении оригинала, создадим ссылку на нее так:

```
$session{my_array} = [ @my_array ];
$session{my_hash} = { %my_hash };
```

Для извлечения нескалярного значения разыменуем ссылку, хранящуюся в сеансе:

```
@my_array = @{$session{my_array}};
%my_hash = %{$session{my_hash}};
```

При завершении работы для закрытия сеанса передайте его в untie:

```
untie (%session);
```

Когда вы закрываете сеанс, Apache::Session сохраняет его в таблице sessions, если были сделаны какие-то изменения. После этого значения сеанса становятся недоступными, так что не закрывайте сеанс до тех пор, пока нет уверенности в том, что больше не нужно будет обращаться к нему.

Apache::Session замечает изменения на «верхнем уровне» значений записей сеанса, но может не обнаружить изменение элемента значения, хранящегося по ссылке (например, элемента массива). Если это создает проблемы, вы можете вынудить Apache::Session сохранить сеанс при закрытии, присвоив значение любому элементу верхнего уровня. Идентификатор сеанса всегда присутствует в хеше сеанса, так что его удобно использовать для инициации сохранения сеанса:

```
$session{_session_id} = $session{_session_id};
```

Открытый сеанс может быть завершен, а не закрыт. Тогда соответствующая запись удаляется из таблицы sessions, так что использовать ее становится невозможно:

```
tied (%session)->delete ();
```

Тестовое приложение

Сценарий *sess_track.pl* представляет полную (хотя и короткую) реализацию приложения, использующего сеанс. Он применяет Apache::Session для отслеживания количества запросов в сеансе и времени каждого запроса, обновляя и выводя информацию при каждом вызове. Сценарий *sess_track.pl* использует файл cookies с именем PERLSESSID для передачи идентификатора сеанса с помощью CGI.pm-интерфейса управления файлами cookies.[1]

```
#! /usr/bin/perl -w
# sess_track.pl - вывод количества запросов сеанса и временных меток

use strict;
use lib qw(/usr/local/apache/lib/perl);
use CGI qw(:standard);
use Cookbook;
use Apache::Session::MySQL;

my $title = "Perl Session Tracker";

my $dbh = Cookbook::connect ();          # соединение с MySQL
my $sess_id = cookie ("PERLSESSID");     # идентификатор сеанса
                                         # (undef для нового сеанса)
my %session;                             # хеш сеанса
my $cookies;                             # cookies для отправки клиенту

# открыть сеанс

tie %session, "Apache::Session::MySQL", $sess_id,
        {
            Handle => $dbh,
            LockHandle => $dbh
        };
```

[1] Для получения информации о поддержке cookies в CGI.pm выполните следующую команду и прочтите раздел, описывающий функцию cookie():

```
% perldoc CGI
```

```
if (!defined ($sess_id))                # это новый сеанс
{
    # получить новый идентификатор сеанса, инициализировать данные
    # сеанса, создать cookies для клиента
    $sess_id = $session{_session_id};
    $session{count} = 0;                # инициализировать счетчик
    $session{timestamp} = [ ];          # инициализировать массив временных меток
    $cookie = cookie (-name => "PERLSESSID", -value => $sess_id);
}

# увеличить счетчик и добавить текущую временную метку в массив

++$session{count};
push (@{$session{timestamp}}, scalar (localtime (time ())));

# сформировать содержимое тела страницы

my $page_body =
    p ("This session has been active for $session{count} requests.")
    . p ("The requests occurred at these times:")
    . ul (li ($session{timestamp}));

if ($session{count} < 10)   # закрыть (и сохранить) сеанс
{
    untie (%session);
}
else                        # удалить сеанс после 10 вызовов
{
    tied (%session)->delete ();
    # вернуть cookies в исходное состояние, чтобы указать броузеру
    # на необходимость очистки cookies сеанса
    $cookie = cookie (-name => "PERLSESSID",
                      -value => $sess_id,
                      -expires => "-1d");     # "истек вчера"
}

$dbh->disconnect ();

# сформировать страницу вывода

print
    header (-cookie => $cookie) # отправить cookies в заголовки (если определены)
    . start_html (-title => $title, -bgcolor => "white")
    . $page_body
    . end_html ();

exit (0);
```

Опробуйте сценарий, установив его в каталоге *cgi-bin* и запросив из броузера. Для повторного вызова используйте функцию обновления страницы, имеющуюся в броузере.

Сценарий *sess_track.pl* открывает сеанс и увеличивает счетчик до того, как приступить к формированию страницы. Это необходимо, поскольку клиенту нужно отправить файл cookies, содержащий имя сеанса и его идентификатор,

если сеанс новый. Все cookies отправляются как часть заголовка ответа, поэтому тело страницы невозможно вывести до тех пор, пока не отправлены заголовки.

Сценарий также генерирует ту часть тела страницы, которая использует данные сеанса, но сохраняет ее в переменной, а не выводит сразу же. Причина такого поведения в том, что приложение, когда ему надо закрыть сеанс, отправляет броузеру файл cookies, содержащий команду удаления предыдущего cookies, переданного ему ранее. Это тоже нужно сделать до отправки заголовков и счетчиков.

Срок хранения данных сеанса

Модуль `Apache::Session` требует наличия в таблице `sessions` только столбцов `id` и `a_session` и не отслеживает время истечения срока сеанса. Но модуль и не запрещает вам добавлять другие столбцы, так что вы можете включить в таблицу столбец `TIMESTAMP` для хранения времени последнего обновления каждого сеанса. Например, можно добавить в таблицу `sessions` столбец `t` типа `TIMESTAMP`, используя `ALTER TABLE`:

```
ALTER TABLE sessions ADD t TIMESTAMP NOT NULL;
```

Тогда вы сможете завершать сеансы, периодически выдавая запрос для очистки таблицы от старых записей. Приведем запрос, в котором срок хранения записей равен четырем часам:

```
DELETE FROM sessions WHERE t < DATE_SUB(NOW(),INTERVAL 4 HOUR);
```

Имейте в виду, что удаление записей сеанса может вызвать проблему: `tie` порождает исключение, если вы пытаетесь выполнить поиск записи сеанса, используя не-`undef`-идентификатор сеанса для несуществующей записи. То есть, например, если клиент передает идентификатор сеанса, срок жизни которого истек, ваш сценарий может завершиться с ошибкой. Можно открывать сеанс в блоке `eval`, чтобы отлавливать ошибки. Если возникает ошибка, создаем новую запись сеанса:

```
eval
{
    tie %session, "Apache::Session::MySQL", $sess_id,
            {
                Handle => $dbh,
                LockHandle => $dbh
            };
};
if ($@) # ошибка - старый сеанс недоступен, создать новый сеанс
{
    $sess_id = undef;
    tie %session, "Apache::Session::MySQL", $sess_id,
            {
                Handle => $dbh,
                LockHandle => $dbh
            };
}
```

19.2. Хранение сеансов в MySQL: менеджер сеансов PHP

Задача

Вам требуется хранилище сеансов для сценариев на PHP.

Решение

PHP 4 поддерживает управление сеансами. По умолчанию он использует для хранения временные файлы, но вы можете задать и хранение в MySQL.

Обсуждение

В PHP 4 есть собственный менеджер сеансов. В этом разделе показано, как его использовать и как расширить его возможности, реализовав модуль хранения данных сеанса в MySQL.[1] Если PHP сконфигурирован так, что оба параметра `track_vars` и `register_globals` установлены, то переменные сеанса будут доступны в сценарии как одноименные глобальные переменные. (Параметр `track_vars` автоматически включен в версии PHP 4.0.3 и выше; в более ранних версиях необходимо включить его явно.) Если параметр `register_globals` отключен, вам придется обращаться к переменным сеанса как к элементам глобального массива `$HTTP_SESSION_VARS` или суперглобального массива `$_SESSION`. Это не так удобно, как просто положиться на `register_globals`, но зато более надежно. (В рецепте 18.5 рассказано о глобальном и суперглобальном массивах PHP и о соображениях безопасности в связи с `register_globals`.)

Интерфейс управления сеансами PHP 4

Управление сеансами в PHP обеспечивается небольшим набором функций, описанных в руководстве по PHP. Перечислим те из них, которые имеют наибольший шанс пригодиться вам в вашей повседневной работе:

`session_start ()`

Открывает сеанс и извлекает переменные, ранее сохраненные в нем, делая их доступными в глобальном пространстве имен сценария. Например, переменная сеанса `x` становится доступна как `$SESSION["x"]` или `$HTTP_SESSION_VARS["x"]`. Если параметр `register_globals` включен, `x` доступна и как глобальная переменная `$x`.

`session_register (`*`var_name`*`)`

Регистрирует переменную в сеансе, устанавливая соответствие между записью сеанса и переменной вашего сценария. Например, зарегистрируем `$count`:

---

[1] PHP 3 не поддерживает сеансы. Пользователи PHP 3, которым требуется применять сеансы, могут использовать PHPLIB или другой пакет, включающий в себя менеджер сеансов.

```
session_register ("count");
```

Если вы как-то изменяете переменную в открытом сеансе, то новое значение будет сохранено в сеансе только при закрытии. Имейте в виду, что переменные регистрируются по имени, а не по значению или ссылке:

```
session_register ($count);        # неверно
session_register (&$count);       # неверно
```

Можно одновременно зарегистрировать несколько переменных, передав вместо одного имени массив, содержащий несколько имен:

```
session_register (array ("count", "timestamp"));
```

Регистрация переменной неявно начинает сеанс, то есть если сценарий вызывает `session_register()`, то ему не нужно предварительно вызывать `session_start()`. Однако `session_register()` имеет силу, только если включен параметр `register_globals`. Чтобы не зависеть от `register_globals`, следует явно вызывать `session_start()` и получать переменные сеанса из массива `$_SESSION` или `$HTTP_SESSION_VARS`.

`session_unregister (`*`var_name`*`)`

Отменяет регистрацию переменной, так что она не сохраняется в записи сеанса.

`session_write_close ()`

Записывает данные сеанса и закрывает его. Обычно эта функция не вызывается, так как PHP автоматически сохраняет открытый сеанс при завершении сценария. Явное сохранение и закрытие сеанса может понадобиться, если вы хотите изменить переменные сеанса так, чтобы изменения не отразились в данных сеанса. В этом случае вызовите данную функцию для закрытия сеанса перед выполнением изменений.

`session_destroy ()`

Удаляет сеанс и все связанные с ним данные.

`session_name ($name)`

Менеджер сеансов PHP узнает, какой сеанс использовать, по идентификатору сеанса. Он ищет идентификатор в глобальной переменной `$PHPSESSID`; в переменной `PHPSESSID` cookies, методах `GET` и `POST`; или в параметре URL в форме `PHPSESSID=`*`значение`*. (Если ни в одном из этих мест идентификатор не обнаружен, менеджер сеансов генерирует новый идентификатор и начинает новый сеанс.) Имя идентификатора по умолчанию – `PHPSESSID`, но вы можете изменить его. Для выполнения глобального изменения (в масштабе сайта) отредактируйте директиву конфигурации `session.name` в *php.ini*. Если же вы хотите внести изменения только для отдельного сценария, вызовите перед началом сеанса `session_name($name)`, где `$name` указывает имя используемого сеанса. Для определения имени идентификатора текущего сеанса вызовите `session_name()` без аргумента.

Рассмотрим пример простейшего применения сеанса – вывод счетчика запросов, полученных к данному моменту в рамках сеанса:

```
session_start ();
session_register ("count");
if (!isset ($count))
    $count = 0;
++$count;
printf ("This session has been active for %d requests.", $count);
```

Функция `session_start()` открывает сеанс и извлекает его содержимое в глобальное пространство имен сценария. (Для первого запроса это не делается, так как сеанс пуст.) Функция `session_register()` регистрирует переменную сеанса `count` для изменения соответствующей переменной PHP `$count` с отслеживанием в данных сеанса. При первом запросе в сеансе нет такой переменной, что выявляется проверкой `isset()`, инициализирующей счетчик. (При последующих запросах регистрация `count` будет приводить к тому, что `$count` получит значение, присвоенное в предыдущем запросе.) Затем значение счетчика увеличивается и выводится. При завершении сценария PHP неявно вызывает функцию `session_write_close()`, которая автоматически сохраняет новое значение счетчика в сеансе.

В примере использована функция `session_register()`, то есть считается, что параметр `register_globals` включен. Далее мы поговорим о том, как обойти это ограничение.

Применение пользовательского модуля хранения

Только что описанный интерфейс управления сеансами PHP не ссылается ни на какое хранилище данных. То есть в описании ничего не говорится о том, как же на самом деле сохраняется информация сеанса. По умолчанию PHP хранит данные сеанса во временных файлах, но интерфейс сеанса является расширяемым, так что можно определить другие модули хранения. Для того чтобы подменить метод хранения по умолчанию и задать хранение данных в MySQL, необходимо выполнить следующие операции:

- Создать таблицу для хранения записей сеансов и написать функции, реализующие модуль хранения. Это делается один раз, перед созданием любых сценариев, использующих новый модуль.
- Сообщить PHP о том, что вы предоставляете пользовательский менеджер хранения. Можно сделать это глобально в *php.ini* (один раз) или в отдельных сценариях (тогда придется выражать свое намерение в каждом сценарии).
- Зарегистрировать функции модуля хранения в каждом сценарии, который планирует использовать модуль.

Создание таблицы сеанса

Любому модулю хранения в MySQL необходима таблица базы данных, в которой будет храниться информация о сеансе. Создадим таблицу `php_session` с такими столбцами:

```
CREATE TABLE php_session
(
```

```
    id      CHAR(32) NOT NULL,
    data    BLOB,
    t       TIMESTAMP NOT NULL,
    PRIMARY KEY (id)
);
```

По своей структуре таблица очень похожа на таблицу `sessions`, используемую модулем Perl `Apache::Session`. Столбец `id` хранит идентификаторы сеансов – уникальные 32-символьные строки (они подозрительно напоминают идентификаторы `Apache:Session`, что неудивительно, поскольку PHP использует значения MD5, как и модуль Perl). Столбец `data` хранит информацию о сеансе. PHP сериализует данные сеанса в строку перед сохранением, так что в таблице `php_session` нужен только большой общий строковый столбец для получения результирующего сериализованного значения. Столбец `t` относится к типу `TIMESTAMP` и автоматически обновляется MySQL при каждом обновлении записи сеанса. Этот столбец не обязателен, но полезен для реализации «сбора мусора» («garbage collection») на основе времени последнего обновления каждого сеанса.

Для обработки содержимого таблицы `php_session` в том виде, в котором мы ее создали, достаточно небольшого набора запросов:

- Для извлечения данных сеанса используйте простой запрос по идентификатору сеанса:

  ```
  SELECT data FROM php_session WHERE id = 'идентификатор_сеанса';
  ```

- Для записи данных сеанса выполните `REPLACE` – будет обновлена существующая запись или создана новая, если запись не существовала ранее:

  ```
  REPLACE INTO php_session (id,data) VALUES('идентификатор_сеанса','данные_сеанса');
  ```

  `REPLACE` обновляет и временную метку записи при ее изменении или создании, что важно для сбора мусора.

  Некоторые менеджеры хранения используют `INSERT` с переходом к `UPDATE` в случае неудачи `INSERT` из-за существования записи с указанным идентификатором сеанса (или `UPDATE` с переходом к `INSERT` при неудаче `UPDATE` из-за того, что запись с таким идентификатором *не* существует). В MySQL нет необходимости в двух запросах, `REPLACE` выполняет поставленную задачу в одном запросе.

- Для уничтожения сеанса удалите соответствующую запись:

  ```
  DELETE FROM php_session WHERE id = 'идентификатор_сеанса';
  ```

- Сбор мусора выполняется посредством удаления старых записей. Следующий запрос удаляет записи, значение временной метки которых превышает *время_жизни_сеанса* в секундах:

  ```
  DELETE FROM php_session
  WHERE t < DATE_SUB(NOW(),INTERVAL время_жизни_сеанса SECOND);
  ```

Эти запросы являются основой функций, образующих наш модуль хранения в MySQL. Главное, что должен делать этот модуль, – открывать и закрывать соединения с MySQL и выдавать соответствующие запросы в нужное время.

Создание функций модуля хранения

Интерфейс пользовательского модуля хранения данных сеансов реализован как набор функций обработчика, которые вы регистрируете вместе с менеджером сеансов PHP, вызывая `session_set_save_handler()` и указывая в качестве аргументов имена функций обработчика в строковом формате:

```
session_set_save_handler (
    "mysql_sess_open",      # функция открытия сеанса
    "mysql_sess_close",     # функция закрытия сеанса
    "mysql_sess_read",      # функция чтения данных сеанса
    "mysql_sess_write",     # функция записи данных сеанса
    "mysql_sess_destroy",   # функция уничтожения сеанса
    "mysql_sess_gc"         # функция сбора мусора
);
```

Функции обработчика можно называть как угодно, их именами не обязательно должны быть `mysql_sess_open()`, `mysql_sess_close()` и т. п. Но в отличие от имен, сами функции должны отвечать указанным требованиям:

`mysql_sess_open ($save_path, $sess_name)`

Выполняет все действия, необходимые для начала сеанса. `$save_path` — это путь к каталогу хранения сеансов, используется только при сохранении в файлах. `$sess_name` указывает имя идентификатора сеанса (например, `PHPSESSID`). Для менеджера хранения в MySQL можно игнорировать оба аргумента. Функция должна возвращать `TRUE` или `FALSE` соответственно при успешном или неуспешном открытии сеанса.

`mysql_sess_close ()`

Закрывает сеанс, возвращая `TRUE` в случае успеха, `FALSE` — при неудаче.

`mysql_sess_read ($sess_id)`

Извлекает данные, связанные с идентификатором сеанса, и возвращает их в виде строки. Если такой сеанс не существует, функция должна возвращать пустую строку. В случае ошибки функция должна возвращать `FALSE`.

`mysql_sess_write ($sess_id, $sess_data)`

Сохраняет данные, связанные с идентификатором сеанса, возвращая `TRUE` в случае успеха, `FALSE` — при неудаче. PHP сам занимается сериализацией и десериализацией содержимого сеанса, поэтому функции чтения и записи имеют дело только с сериализованными строками.

`mysql_sess_destroy ($sess_id)`

Уничтожает сеанс и все связанные с ним данные, возвращая `TRUE` в случае успеха, `FALSE` — при неудаче. При хранении данных сеанса в MySQL уничтожение сеанса выражается в удалении записи, соответствующей идентификатору сеанса, из таблицы `php_session`.

`mysql_sess_gc ($gc_maxlife)`

Выполняет сбор мусора для удаления старых сеансов. Эта функция вызывается с определенной вероятностью. Когда PHP получает запрос на стра-

ницу, использующую сеансы, он вызывает утилиту сбора мусора с вероятностью, указанной директивой конфигурации `session.gc_probability` в файле *php.ini*. Например, если вероятность равна 1 (то есть 1%), PHP вызывает сборщика мусора приблизительно один раз на сто запросов. Если значение вероятности равно 100, сборщик вызывается для каждого запроса, что может привести к нежелательно большому увеличению нагрузки.

Аргумент `$gc_maxlife` – это максимальное время жизни сеанса в секундах. Более старые сеансы являются кандидатами на удаление. Функция должна возвращать `TRUE` в случае успеха, `FALSE` – при неудаче.

Функции обработчика регистрируются при помощи вызова `session_set_save_handler()`, выполняемого совместно с уведомлением PHP о том, что вы будете применять пользовательский модуль хранения. Метод хранения данных сеансов по умолчанию определяется конфигурационной директивой `session.save_handler`. Можно изменить метод глобально, отредактировав инициализационный файл *php.ini*, или только для отдельного сценария:

- Для глобального изменения метода хранения отредактируйте файл *php.ini*. Установка директивы по умолчанию задает использование файлового хранилища для сеансов:

```
session.save_handler = files;
```

Измените директиву, указав, что сеансы будут обрабатываться пользовательским механизмом:

```
session.save_handler = user;
```

Если вы используете PHP как модуль Apache, то после редактирования *php.ini* потребуется перезапустить Apache, чтобы изменения вступили в силу.

Проблема глобального изменения – оно подразумевает, что каждый сценарий PHP, использующий сеансы, предоставит собственные функции работы с хранилищем. Возможны неожиданные побочные эффекты для других создателей сценариев, не знающих об изменении. Например, другие разработчики, использующие веб-сервер, могут предпочитать хранить данные сеансов в файловом хранилище.

- Альтернативой глобальным изменениям является задание для отдельного сценария собственного метода хранения посредством вызова `ini_set()`:

```
ini_set ("session.save_handler", "user");
```

В отличие от изменения глобальной конфигурации вызов функции `ini_set()` не приводит к побочным эффектам. Созданный нами менеджер хранения использует `ini_set()`, так что хранение сеансов в базе данных будет производиться только для тех сценариев, которые это запросят.

Для упрощения доступа к альтернативному модулю хранения данных сеансов создадим библиотечный файл *Cookbook_Session.php*. Тогда единственное, что нужно будет сделать сценарию для использования библиотечного файла, – включить его в себя до начала сеанса. Файл будет приблизительно таким:

```
<?php
# Cookbook_Session.php - модуль хранения сеансов в MySQL

include_once "Cookbook.php";

# Определяем функции обработчика.

function mysql_sess_open ($save_path, $sess_name) ...
function mysql_sess_close () ...
function mysql_sess_read ($sess_id) ...
function mysql_sess_write ($sess_id, $sess_data) ...
function mysql_sess_destroy ($sess_id) ...
function mysql_sess_gc ($gc_maxlife) ...

# Инициализируем идентификатор соединения, выбираем пользовательский
# обработчик сеансов и регистрируем его функции.

$mysql_sess_conn_id = FALSE;
ini_set ("session.save_handler", "user");
session_set_save_handler (
                    "mysql_sess_open",
                    "mysql_sess_close",
                    "mysql_sess_read",
                    "mysql_sess_write",
                    "mysql_sess_destroy",
                    "mysql_sess_gc"
                );
?>
```

Библиотечный файл включает в себя *Cookbook.php*, поэтому для установки соединения с базой данных `cookbook` он может обращаться к функции соединения. Затем он определяет функции обработчика (поговорим о них подробнее чуть позже). Наконец, он инициализирует идентификатор соединения, сообщает PHP о том, что следует подготовиться к использованию пользовательского механизма хранения сеансов, и регистрирует функции обработчика. Тогда сценарий PHP, который хочет хранить данные сеансов в MySQL, выполняет все необходимые настройки, просто присоединяя файл *Cookbook_Session.php*:

```
include_once "Cookbook_Session.php";
```

Интерфейс, предоставляемый библиотечным файлом *Cookbook_Session.php*, определяет глобальную переменную идентификатора соединения с базой данных (`$mysql_sess_conn_id`), набор функций обработчика с именами `mysql_sess_open()`, `mysql_sess_close()` и т. д. Сценарии, использующие библиотеку, не должны применять эти глобальные имена в других целях.

Теперь давайте посмотрим, как реализована каждая функция обработчика.

Открытие сеанса. PHP передает этой функции два аргумента: путь сохранения и имя сеанса. Путь сохранения нужен файловым хранилищам, а имя сеанса нам знать необязательно, так что для нас оба аргумента не имеют значения и могут быть проигнорированы. Поэтому функция должна только открывать соединение с MySQL:

```
function mysql_sess_open ($save_path, $sess_name)
{
global $mysql_sess_conn_id;

    # открыть соединение с MySQL, если еще не установлено
    $mysql_sess_conn_id or $mysql_sess_conn_id = cookbook_connect ();
    return (TRUE);
}
```

Закрытие сеанса. **Обработчик закрытия проверяет, открыто ли соединение с MySQL, и закрывает его, если открыто:**

```
function mysql_sess_close ()
{
global $mysql_sess_conn_id;

    if ($mysql_sess_conn_id)        # закрыть соединение, если оно открыто
    {
        mysql_close ($mysql_sess_conn_id);
        $mysql_sess_conn_id = FALSE;
    }
    return (TRUE);
}
```

Чтение данных сеанса. **Функция** `mysql_sess_read()` **возвращает запись сеанса, найденную по ее идентификатору. Если запись не существует, возвращается пустая строка:**

```
function mysql_sess_read ($sess_id)
{
global $mysql_sess_conn_id;

    $sess_id = addslashes ($sess_id);
    $query = "SELECT data FROM php_session WHERE id = '$sess_id'";
    if ($res_id = mysql_query ($query, $mysql_sess_conn_id))
    {
        list ($data) = mysql_fetch_row ($res_id);
        mysql_free_result ($res_id);
        if (isset ($data))
            return ($data);
    }
    return ("");
}
```

Запись данных сеанса. **Функция** `mysql_sess_write()` **обновляет запись сеанса (или создает ее, если запись ранее не существовала):**

```
function mysql_sess_write ($sess_id, $sess_data)
{
global $mysql_sess_conn_id;

    $sess_id = addslashes ($sess_id);
    $sess_data = addslashes ($sess_data);
    $query = "REPLACE php_session (id, data) VALUES('$sess_id','$sess_data')";
    return (mysql_query ($query, $mysql_sess_conn_id));
}
```

Уничтожение сеанса. Когда сеанс больше не нужен, функция `mysql_sess_destroy()` удаляет соответствующую запись:

```
function mysql_sess_destroy ($sess_id)
{
global $mysql_sess_conn_id;

    $sess_id = addslashes ($sess_id);
    $query = "DELETE FROM php_session WHERE id = '$sess_id'";
    return (mysql_query ($query, $mysql_sess_conn_id));
}
```

Сбор мусора. Столбец `t` типа `TIMESTAMP` для каждой записи сеанса указывает, когда сеанс обновлялся в последний раз. Функция `mysql_sess_gc()` использует это значение при сборе мусора. Аргумент `$sess_maxlife` показывает, насколько старыми могут быть записи (в секундах). Более ранние записи считаются устаревшими и являются кандидатами на удаление. Удаляются записи сеансов, временная метка которых отличается от текущей более чем на разрешенную величину срока жизни:

```
function mysql_sess_gc ($sess_maxlife)
{
global $mysql_sess_conn_id;

    $query = sprintf ("DELETE FROM php_session
                       WHERE t < DATE_SUB(NOW(),INTERVAL %d SECOND)",
                       $sess_maxlife);
    mysql_query ($query, $mysql_sess_conn_id);
    return (TRUE);  # игнорировать ошибки
}
```

Использование модуля хранения

Установите файл *Cookbook_Session.php* в общий библиотечный каталог, доступный вашим сценариям. (Я помещаю библиотечные файлы PHP в каталог */usr/local/apache/lib/php*.) Для опробования модуля поместите предлагаемый ниже сценарий *sess_track.php* в дерево документов и вызовите его несколько раз, чтобы посмотреть, как изменяется информация (точнее, чтобы увидеть, меняется ли она; как вы вскоре узнаете, в определенных случаях сценарий не будет работать):

```
<?php
# sess_track.php - вывод количества запросов в сеансе и временных меток
# (считаем, что register_globals включен)

include_once "Cookbook_Session.php";
include_once "Cookbook_Webutils.php";   # необходимо для make_unordered_list()

$title = "PHP Session Tracker";

# Открыть сеанс и зарегистрировать переменные сеанса.

session_start ();
session_register ("count");
```

```
session_register ("timestamp");

# Если сеанс новый, то инициализировать переменные.

if (!isset ($count))
    $count = 0;
if (!isset ($timestamp))
    $timestamp = array ();

# Увеличить счетчик, добавить текущую временную метку в массив.

++$count;
$timestamp[] = date ("Y-m-d H:i:s T");

if ($count >= 10)   # очистить переменные сеанса после 10 вызовов
{
    session_unregister ("count");
    session_unregister ("timestamp");
}

# Сформировать страницу вывода.

?>
<html>
<head>
<title><?php print ($title); ?></title>
</head>
<body bgcolor="white">

<?php

printf ("<p>This session has been active for %d requests.</p>\n", $count);
print ("<p>The requests occurred at these times:</p>\n");
print make_unordered_list ($timestamp);

?>

</body>
</html>
```

Сценарий присоединяет библиотечный файл *Cookbook_Session.php* для работы с модулем хранения в MySQL, затем стандартным образом использует интерфейс менеджера сеансов PHP. Сначала он открывает сеанс, регистрирует переменные сеанса и инициализирует их для нового сеанса. Скалярная переменная `$count` устанавливается в ноль, а не скалярная переменная `$timestamp` – в пустой массив. Затем сценарий увеличивает счетчик, добавляет текущую временную метку в конец массива и формирует страницу вывода, отображающую счетчик и время обращения.

Если выполнено предельно допустимое количество вызовов – 10, то сценарий отменяет регистрацию переменных сеанса, в результате чего `$count` и `$timestamp` не сохраняются в записи сеанса. При следующем запросе сеанс начинается заново.

Сценарий *sess_track.php* явно не вызывает `session_write_close()`; PHP автоматически сохраняет сеанс при завершении сценария.

Страница вывода формируется только после обновления записи сеанса, так как может оказаться, что необходимо отправить клиенту cookies с идентификатором сеанса. Необходимо сделать это до начала формирования тела страницы, так как cookies отправляются в заголовках.

Проблема предложенного сценария *sess_track.php* в том, что он работает только в том случае, если установлен параметр конфигурации PHP `register_globals`. Тогда регистрация переменных сеанса `count` и `timestamp` приводит к тому, что их значения становятся доступны как глобальные переменные PHP `$count` и `$timestamp`. Однако если параметр `register_globals` не включен, то `session_register()` ничего не делает, и сценарий *sess_track.php* будет работать неправильно (счетчик всегда будет равен единице, будет выводиться только одна временная метка).

Этот момент очень важен, так как разработчики PHP не рекомендуют включать параметр `register_globals` из соображений безопасности. Это значит, что `session_register()` фактически выходит из употребления, и что существующие приложения, использующие эту функцию для работы с сеансами, будут выходить из строя по мере того, как все большее количество сайтов будет следовать рекомендациям отключения `register_globals`. Для того чтобы решить проблему и написать код, который работал бы вне зависимости от установки `register_globals`, необходимо получать переменные сеанса другим способом. Есть две возможности: использовать глобальный массив `$HTTP_SESSION_VARS` или (начиная с PHP 4.1) суперглобальный массив `$_SESSION`. Например, переменная сеанса `count` будет доступна как `$HTTP_SESSION_VARS["count"]` или `$_SESSION["count"]`.

Можно достаточно легко изменить сценарий *sess_track.php* так, чтобы он не полагался на установку `register_globals`, но позволял работать с простыми именами переменных для манипулирования переменными сеанса:

- Не используйте `session_register()`. Вместо этого скопируйте переменные сеанса из глобального массива непосредственно в переменные `$count` и `$timestamp`.
- После того как работа с переменными сеанса завершена, скопируйте их обратно в массив переменных сеанса. Выполните копирование до записи сеанса, если вы явно вызываете `session_write()`.

Этот подход требует от вас указания того, какой глобальный массив следует использовать для хранения переменных сеанса (что может зависеть от версии PHP). Вместо того чтобы делать это каждый раз при обращении к переменной сеанса, удобнее написать пару функций, которые все сделают сами:

```
function get_session_val ($name)
{
global $HTTP_SESSION_VARS;

    unset ($val);
    if (isset ($_SESSION[$name]))
        $val = $_SESSION[$name];
    else if (isset ($HTTP_SESSION_VARS[$name]))
```

```
        $val = $HTTP_SESSION_VARS[$name];
    return (@$val);
}

function set_session_val ($name, $val)
{
global $HTTP_SESSION_VARS;

    if (PHP_VERSION >= "4.1")
        $_SESSION[$name] = $val;
    $HTTP_SESSION_VARS[$name] = $val;
}
```

Эти функции включены в библиотечный файл *Cookbook_Webutils.php* наряду с функциями, получающими значения других веб-параметров (см. рецепт 18.5). Они хранятся в *Cookbook_Webutils.php*, а не в *Cookbook_Session.php*, так что вы сможете вызвать их, даже если не захотите использовать модуль хранения сеанса в MySQL, реализуемый *Cookbook_Session.php*.

Рассмотрим сценарий *sess_track2.php*, показывающий, как избежать применения `register_globals`, лишь слегка изменив основную логику:

```
<?php
# sess_track2.php - вывод количества запросов сеанса и временных меток
# (без использования register_globals)

include_once "Cookbook_Session.php";
include_once "Cookbook_Webutils.php";  # требуется для make_unordered_list(),
                                       # get_session_val(), set_session_val()

$title = "PHP Session Tracker";

# Открыть сеанс и извлечь значения сеанса.

session_start ();
$count = get_session_val ("count");
$timestamp = get_session_val ("timestamp");

# Если сеанс новый, инициализировать переменные.

if (!isset ($count))
    $count = 0;
if (!isset ($timestamp))
    $timestamp = array ();

# Увеличить счетчик, добавить текущую временную метку в массив.

++$count;
$timestamp[] = date ("Y-m-d H:i:s T");

if ($count < 10)    # сохранить измененные значения в массиве переменных сеанса
{
    set_session_val ("count", $count);
    set_session_val ("timestamp", $timestamp);
}
else                # очистить переменные сеанса после 10 вызовов
```

```
{
    session_unregister ("count");
    session_unregister ("timestamp");
}

# сформировать страницу

?>
<html>
<head>
<title><?php print ($title); ?></title>
</head>
<body bgcolor="white">

<?php

printf ("<p>This session has been active for %d requests.</p>\n", $count);
print ("<p>The requests occurred at these times:</p>\n");
print make_unordered_list ($timestamp);

?>

</body>
</html>
```

Сценарий *sess_track2.php* практически идентичен *sess_track.php*, есть всего два отличия:

- *sess_track.php* вызывает `session_start()` для открытия сеанса, но это необязательно, так как используется функция `session_register()`, которая неявно открывает сеанс. Сценарий *sess_track2.php* не использует `session_register()`. Он получает значения переменных напрямую из глобального хранилища переменных сеанса. В данном случае *необходимо* сначала вызывать функцию `session_start()` для явного открытия соединения.
- Если предельное количество запросов в сеансе (10) еще не достигнуто, то *sess_track2.php* явно сохраняет значения сеанса `$count` и `$timestamp` в глобальных массивах переменных сеанса, вызывая функцию `set_session_val()`.

19.3. Хранение сеансов в MySQL: Tomcat

Задача

Вы хотите хранить данные сеансов для сценариев на Java.

Решение

Tomcat самостоятельно осуществляет управление сеансами. По умолчанию в качестве хранилища используются временные файлы, но вы можете указать в конфигурационном файле Tomcat *server.xml* соответствующие параметры JDBC для хранения сеансов в MySQL.

Обсуждение

Описанные ранее механизмы сеансов Perl и PHP требуют, чтобы приложения явно указывали на то, что для хранения сеансов следует применять MySQL. В Perl сценарий должен сообщить, что он хочет использовать соответствующий модуль `Apache::Session`. В PHP 4 менеджер сеансов встроен в язык, но каждое приложение, которое хочет использовать модуль хранения в MySQL, должно его зарегистрировать.

В приложениях на Java, работающих под управлением Tomcat, все по-другому. Tomcat сам обрабатывает сеансы, и если вы хотите хранить данные сеанса в MySQL, то следует переконфигурировать Tomcat, а не приложения. Другими словами, веб-программы Java свободны от деталей, связанных с обработкой сеансов, которые в других языках присутствуют именно на уровне приложений. Например, идентификаторами сеансов заведует Tomcat, а не приложения. Tomcat проверяет, включена ли поддержка cookies, и модифицирует URL, добавляя в него идентификатор сеанса, если cookies недоступны. Разработчикам приложений не приходится думать о том, какой метод использовать, так как доступность идентификатора не зависит от способа его передачи.

Для иллюстрации независимости приложения от метода управления сеансом, используемого Tomcat, в этом разделе рассматривается простое приложение JSP, применяющее сеанс. Описано изменение конфигурации Tomcat для хранения данных сеанса в MySQL, а не в хранилище сеансов по умолчанию, при этом никаких изменений в самом приложении выполнять не нужно. Однако начать следует с разговора об интерфейсе сеансов.

Интерфейс сеансов для сервлетов и страниц JSP

Tomcat использует стандартный интерфейс сеансов, описанный в спецификации Java Servlet. Этот интерфейс может быть использован и сервлетами, и страницами JSP. В сервлете вы получаете доступ к сеансу, импортируя класс `javax.servlet.http.HttpSession` и вызывая метод `getSession()` вашего объекта `HttpRequest`:

```
import javax.servlet.http.*;
HttpSession session = request.getSession ();
```

Для страниц JSP поддержка сеансов включена по умолчанию, то есть эти предложения как будто уже выполнены к моменту начала исполнения страницы. Таким образом, сеанс неявно доступен через переменную `session`, которая уже установлена без вашего участия.

Полный интерфейс сеансов определен в разделе `HttpSession` спецификации Java Servlet (см. приложение C). Приведем некоторые наиболее характерные методы объектов сеансов:

`isNew ()`

Возвращает истину или ложь в зависимости от того, начинается ли сеанс текущим запросом.

`getAttribute (String attrName)`

Содержимое сеанса состоит из атрибутов, которые являются объектами, связанными с именами. Для доступа к атрибуту сеанса укажите его имя. Метод `getAttribute()` возвращает объект, связанный с указанным именем, или `null`, если объект с таким именем не существует.

`setAttribute (String attrName, Object obj)`

Добавляет объект в сеанс и связывает его с указанным именем.

`removeAttribute (String attrName)`

Удаляет атрибут с указанным именем из сеанса.

`invalidate ()`

Объявляет некорректным сеанс и все связанные с ним данные. Следующий запрос клиента начнет новый сеанс.

Пример JSP-приложения, использующего сеансы

Рассмотрим страницу JSP *sess_track.jsp*, которая ведет счетчик запросов сеанса и журнал времени запросов. Для более наглядной иллюстрации операций, связанных с сеансами, страница состоит в основном из встроенного кода Java, который напрямую использует интерфейс сеанса `HttpSession`:

```
<%-- sess_track.jsp - вывод количества запросов сеанса и временных меток --%>

<%@ page import="java.util.*" %>
<%
    // получить переменные сеанса, инициализируя их в случае отсутствия

    int count;
    Object obj = session.getAttribute ("count");
    if (obj == null)
        count = 0;
    else
        count = Integer.parseInt (obj.toString ());

    ArrayList timestamp = (ArrayList) session.getAttribute ("timestamp");
    if (timestamp == null)
        timestamp = new ArrayList ();

    // увеличить счетчик, добавить текущую временную метку в массив

    count = count + 1;
    timestamp.add (new Date ());

    if (count < 10)     // сохранить обновленные значения в объекте сеанса
    {
        session.setAttribute ("count", String.valueOf (count));
        session.setAttribute ("timestamp", timestamp);
    }
    else                // начать новый сеанс после 10 запросов
    {
        session.removeAttribute ("count");
```

```
        session.removeAttribute ("timestamp");
    }
%>

<html>
<head>
<title>JSP Session Tracker</title>
</head>
<body bgcolor="white">

<p>This session has been active for <%= count %> requests.</p>

<p>The requests occurred at these times:</p>
<ul>
<%
    for (int i = 0; i < timestamp.size (); i++)
        out.println ("<li>" + timestamp.get (i) + "</li>");
%>
</ul>

</body>
</html>
```

Вызовите *sess_track.jsp* несколько раз в броузере, чтобы посмотреть на то, как меняется вывод.

Метод `session.setAttribute()`, применявшийся в странице *sess_track.jsp*, помещает информацию в сеанс, с тем чтобы она в дальнейшем могла быть обнаружена при последующих вызовах сценария. Атрибуты сеанса могут совместно использоваться несколькими сценариями. Чтобы убедиться в этом, создайте копию *sess_track.jsp* и вызовите копию из броузера. Вы увидите, что страница обращается к тем же данным сеанса, что и *sess_track.jsp*.

Некоторые операции с сеансами, приведенные в *sess_track.jsp*, можно выполнить при помощи тегов JSTL, содержащих переменную `sessionScope` для обращения к неявному объекту сеанса JSP:

```
<%-- sess_track2.jsp - вывод количества запросов и временных меток сеанса --%>

<%@ page import="java.util.*" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>

<c:if test="${empty sessionScope.count}">
    <c:set var="count" scope="session" value="0" />
</c:if>
<c:set var="count" scope="session" value="${sessionScope.count+1}" />

<%
    ArrayList timestamp = (ArrayList) session.getAttribute ("timestamp");
    if (timestamp == null)
        timestamp = new ArrayList ();
    // добавить текущую временную метку в массив, сохранить результат в сеансе
    timestamp.add (new Date ());
    session.setAttribute ("timestamp", timestamp);
%>
```

```
<html>
<head>
<title>JSP Session Tracker 2</title>
</head>
<body bgcolor="white">

<p>This session has been active for
<c:out value="${sessionScope.count}" />
requests.</p>

<p>The requests occurred at these times:</p>
<ul>
<c:forEach var="t" items="${sessionScope.timestamp}">
    <li><c:out value="${t}" /></li>
</c:forEach>
</ul>

<%-- достигнут ли предел в 10 запросов? --%>

<c:if test="${sessionScope.count ge 10}">
    <c:remove var="count" scope="session" />
    <c:remove var="timestamp" scope="session" />
</c:if>

</body>
</html>
```

Как указать Tomcat сохранять записи сеанса в MySQL

Документация Tomcat, относящаяся к работе с сеансами, доступна по адресу:

http://jakarta.apache.org/tomcat/tomcat-4.0-doc/config/manager.html

В Tomcat есть собственный механизм хранения сеансов, используемый по умолчанию (во временных файлах). Для сохранения сеансов в MySQL при помощи JDBC выполните следующие операции:

1. Создайте таблицу для хранения записей сеансов.
2. Убедитесь в том, что у Tomcat есть доступ к нужному драйверу JDBC.
3. Измените конфигурационный файл Tomcat *server.xml*, задав использование постоянного менеджера сеансов для соответствующего контекста (контекстов) приложений.

Ни один из этих шагов не касается изменения самого приложения, это отражает тот факт, что Tomcat реализует поддержку сеансов выше уровня приложений.

Создание таблицы сеансов Tomcat

Tomcat сохраняет в таблице сеанса различные виды информации:

- Идентификатор сеанса. По умолчанию – 32-символьные значения MD5.
- Данные сеанса. Это сериализованная строка.
- Однобайтовый признак корректности сеанса.

- Максимально разрешенное время простоя в секундах в виде 32-битного целого.
- Время последнего обращения в виде 64-битного целого.

Этим условиям удовлетворяет таблица, которую следует создать прежде, чем переходить к следующему разделу:

```
CREATE TABLE tomcat_session
(
    id              CHAR(32) NOT NULL,
    data            BLOB,
    valid_session   CHAR(1) NOT NULL,
    max_inactive    INT NOT NULL,
    last_access     BIGINT NOT NULL,
    PRIMARY KEY (id)
);
```

Размещение драйвера JDBC там, где Tomcat сможет его найти

Поскольку Tomcat сам управляет сеансами, он должен иметь возможность доступа к драйверу JDBC, используемому для сохранения данных сеансов в базе данных. Принято хранить драйверы в каталоге *lib* дерева Tomcat, чтобы к ним могли обращаться все приложения. Но для того чтобы драйвер был доступен и серверу Tomcat, он должен размещаться в каталоге *common/lib*. (То есть если у вас драйвер MySQL Connector/J установлен в *lib*, переместите его в *common/lib*.) После перезапуска Tomcat сможет его использовать. За более подробной информацией можно обратиться к рецепту 16.3.

Изменение конфигурационного файла Tomcat

Чтобы сообщить Tomcat о том, что следует использовать таблицу `tomcat_session`, необходимо изменить файл *server.xml* в каталоге Tomcat *conf*. Для этого поместите элемент `<Manager>` в тело элемента `<Context>` каждого контекста приложения, который должен использовать MySQL-хранилище сеансов. (Если в контексте нет такого элемента, создайте его.) Для контекста приложения `mcb` элемент `<Context>` можно создать так:

```
<Context path="/mcb" docBase="mcb" debug="0" reloadable="true">
  <Manager
    className="org.apache.catalina.session.PersistentManager"
    debug="0"
    saveOnRestart="true"
    minIdleSwap="900"
    maxIdleSwap="1200"
    maxIdleBackup="600">
    <Store
      className="org.apache.catalina.session.JDBCStore"
      driverName="com.mysql.jdbc.Driver"
      connectionURL=
        "jdbc:mysql://localhost/cookbook?user=cbuser&amp;password=cbpass"
      sessionTable="tomcat_session"
```

```
        sessionIdCol="id"
        sessionDataCol="data"
        sessionValidCol="valid_session"
        sessionMaxInactiveCol="max_inactive"
        sessionLastAccessedCol="last_access"
      />
    </Manager>
  </Context>
```

Атрибуты элемента `<Manager>` определяют общие параметры, относящиеся к сеансу. Внутри тела элемента `<Manager>` атрибуты элемента `<Store>` определяют особенности драйвера JDBC. Дальше мы будем говорить об атрибутах, приведенных в примере, но существуют и другие, которые вы также можете использовать. Обратитесь за информацией к документации по управлению сеансами Tomcat.

Использованные в примере атрибуты `<Manager>` имеют следующие значения:

`className`

Указывает класс Java, реализующий постоянное хранилище данных. Должен иметь значение `org.apache.catalina.session.PersistentManager`.

`debug`

Задает уровень подробности протоколирования. Нулевое значение отменяет отладочный вывод; чем больше значение, тем подробнее протокол.

`saveOnRestart`

Позволяет сеансам приложений пережить перезагрузку сервера. Должен быть установлен в `true`, если вы хотите, чтобы Tomcat сохранял текущие сеансы при отключении (и заново загружал их при перезапуске).

`maxIdleBackup`

Указывает количество секунд, в течение которых неактивные сеансы могут сохраняться в MySQL. Установка в `-1` означает «никогда».

`minIdleSwap`

Задает количество секунд, в течение которых сеанс может находиться в состоянии ожидания, прежде чем он сможет быть свопированным (выгруженным из области памяти сервера и сохраненным в MySQL). Установка в `-1` означает «никогда».

`maxIdleSwap`

Задает количество секунд, в течение которых сеанс может находиться в состоянии ожидания и по истечении которого он должен быть свопирован. Установка в `-1` означает «никогда». Если параметр установлен, его значение должно превышать `minIdleSwap` и `maxIdleBackup`.

Внутри тела элемента `<Manager>` атрибуты элемента `<Store>` указывают, как осуществлять соединение с сервером базы данных, какую базу данных и таблицу использовать для хранения записей сеанса, и каковы имена столбцов таблицы:

`className`

Имя класса, реализующего интерфейс `org.apache.catalina.Store`. Для менеджеров хранения на основе JDBC этот атрибут должен иметь значение `org.apache.catalina.session.JDBCStore`.

`driverName`

Имя класса драйвера JDBC. Для драйвера MySQL Connector/J этот атрибут следует установить в значение `com.mysql.jdbc.Driver`.

`connectionURL`

Указывает, как подключаться к серверу базы данных. Следующий URL служит для соединения с сервером MySQL, работающим на локальном хосте, с именем пользователя `cbuser` и паролем `cbpass`:

```
jdbc:mysql://localhost/cookbook?user=cbuser&password=cbpass
```

Однако содержимое *server.xml* записывается на XML, поэтому символ `&`, разделяющий параметры соединения `user` и `password`, следует записать как объект `&`, а весь адрес станет таким:

```
jdbc:mysql://localhost/cookbook?user=cbuser&amp;password=cbpass
```

Когда Tomcat читает файл *server.xml*, программа синтаксического анализа преобразует `&` обратно в символ `&`, который и передается драйверу JDBC.

`sessionTable`

Называет таблицу для хранения записей сеанса. В нашем примере это описанная ранее таблица `tomcat_session`.

Оставшиеся атрибуты `<Store>` примера определяют имена столбцов таблицы сеансов. Это атрибуты `sessionIdCol`, `sessionDataCol`, `sessionValidCol`, `sessionMaxInactiveCol` и `sessionLastAccessedCol`, которые очевидным образом соответствуют столбцам таблицы `tomcat_session`.

После изменения файла *server.xml* перезапустите Tomcat. Затем вызовите несколько раз сценарий *sess_track.jsp* или *sess_track2.jsp* для инициации сеанса. Все они должны вести себя так же, как до изменения конфигурации Tomcat. После периода неактивности, равного значению атрибута `maxIdleBackup` элемента `<Manager>`, вы должны увидеть, как запись о сеансе появится в таблице `tomcat_session`. Если вы следите за журналом запросов MySQL, то должны заметить и сохранение сеансов в MySQL при отключении сервера Tomcat.

Изменение *server.xml* является глобальным, чем напоминает изменение значения `session.save_handler` в конфигурационном файле PHP *php.ini*. Однако, в отличие от PHP (где изменение глобального файла инициализации влияет на других разработчиков того же хоста таким образом, что у них может возникнуть необходимость изменения сценариев, применяющих сеансы), изменение конфигурации Tomcat для хранения сеансов при помощи JDBC остается абсолютно невидимым для сервлетов и страниц JSP. То есть вы можете вы-

полнять изменения, не беспокоясь о том, что другие разработчики, использующие тот же сервер Tomcat, обвинят вас в предумышленном злодеянии.

Истечение срока сеанса в Tomcat

Продолжительность сеанса по умолчанию равна 60 минутам. Чтобы явно определить продолжительность для менеджера сеансов, добавьте `maxInactiveInterval` в соответствующий элемент `<Manager>` файла *conf/server.xml* сервера. Для указания длительности в рамках определенного контекста приложения добавьте элемент `<session-config>` в файл *WEB-INF/web.xml* контекста. Например, установим продолжительность в 30 минут:

```
<session-config>
  <session-timeout>30</session-timeout>
</session-config>
```

Если вы изменяете *server.xml* или *web.xml*, перезапустите Tomcat.

Отслеживание сеансов в Tomcat

Ваши страницы JSP не должны предпринимать никаких действий для настройки сеансов или использования JDBC для хранения данных сеанса, но им может понадобиться убедиться в том, что сеансы корректно переходят от запроса к запросу. Это необходимо, если вы генерируете страницы, содержащие гиперссылки на другие страницы, участвующие в том же сеансе.

Tomcat автоматически генерирует идентификатор сеанса и следит за сеансом при помощи cookies, если получает от клиента cookies с идентификатором сеанса. Если у клиента отключена поддержка cookies, Tomcat следит за сеансом, включая его идентификатор в URL. Для вас не важно, какой именно способ использует Tomcat, необходимо лишь позаботиться о корректной передаче идентификатора сеанса в случае, если он передается в URL. То есть если вы создаете страницу, которая включает в себя ссылку на другую страницу, входящую в сеанс, то нельзя просто формировать путь к странице как:

```
To go to the next page,
<a href="nextpage.jsp">click here</a>.
```

Эта ссылка не содержит идентификатор сеанса. Если Tomcat отслеживает сеанс за счет перезаписи URL, то вы потеряете идентификатор, когда пользователь выберет такую ссылку. Поэтому следует передать ссылку в функцию `encodeURL()` для того, чтобы Tomcat добавил в URL идентификатор сеанса:

```
To go to the next page,
<a href="<%= response.encodeURL ("nextpage.jsp") %>">click here</a>.
```

Если Tomcat отслеживает сеанс по cookies, то `encodeURL()` вернет URL без изменений. Если же для слежения за сеансом используется URL, то `encodeURL()` автоматически добавит идентификатор сеанса в путь страницы, и он будет выглядеть как-то так:

```
mypage.jsp;jsessionid=xxxxxxxxxxxxxxxx
```

Необходимо формировать URL с использованием `encodeURL()` для ссылок в любых тегах, которые приводят пользователя на страницу в текущем сеансе. Речь идет о тегах `<a>`, `<form>`, `<frame>` и, возможно, даже о теге `<img>`, если по какой-то причине такие теги вызывают сценарий, формирующий изображения в рамках сеанса.

Вероятно, лучше всего завести привычку применять `encodeURL()` при создании URL в приложениях, использующих сеансы. Даже если вы думаете, что у всех работающих с приложением включена поддержка cookies, однажды это предположение может оказаться неверным.

Имя метода `java.net.URLEncoder.encode()` похоже на `encodeURL()`, но на этом сходство заканчивается. Метод преобразует специальные символы в нотацию `%xx` для обеспечения их безопасного использования в URL.

A

Получение программного обеспечения MySQL

Большинство программ и определений таблиц, рассмотренных в книге, доступны в электронном виде, благодаря чему вы избавлены от необходимости набирать тексты вручную. Разумеется, для выполнения примеров потребуется доступ к MySQL, а также соответствующий модуль интерфейса для выбранного вами языка программирования. В этом приложении рассказано, где можно получить необходимые программы.

Исходные тексты примеров и данные

В основе примеров этой книги лежат исходные тексты и тестовые данные, взятые из трех источников. Два из них (дистрибутивы `recipes` и `mcb-kjv`) доступны на веб-сайте книги «MySQL Cookbook» по адресу:

http://www.kitebird.com/mysql-cookbook/

Дистрибутив `recipes` – это основной источник примеров. Он доступен в двух форматах – в сжатом файле *tar* (*recipes.tar.gz*) и в ZIP-файле (*recipes.zip*). Каждый из архивов при распаковке создает каталог с именем `recipes`.

Кроме программ, приведенных в книге, дистрибутив `recipes` зачастую содержит и их реализации и на других языках. Например, рецепт, реализованный в книге на Python, на сайте может быть доступен в версиях на Perl, PHP или Java. Это упростит вашу задачу, если вы захотите переписать программу на другом языке. Я планирую обновлять этот дистрибутив время от времени, поэтому проверяйте иногда, не появились ли изменения. Если вы захотите предложить свой вариант реализации для включения в дистрибутив, напишите мне об этом по адресу *mysql-cookbook@kitebird.com*.

На сайте Kitebird имеется также дистрибутив `mcb-kjv`, содержащий текст Библии в версии King James Version (KJV) в формате, пригодном для импорта в MySQL. Он используется в главе 4 в качестве источника достаточно большого текста для примеров, демонстрирующих полнотекстовый поиск, и в некоторых других случаях. Этот дистрибутив не входит в `recipes` из-за своего

большого размера. Он доступен в виде сжатого файла *tar* (*mcb-kjv.tar.gz*) или ZIP-файла (*mcb-kjv.zip*), которые при распаковке создают каталог *mcb-kjv*.

Дистрибутив `mcb-kjv` получен из текста KJV, доступного на библейском сайте университета Биола (Biola) (*http://unbound.biola.edu*). Дистрибутив содержит примечания, описывающие его отличия от оригинального текста.

Текстовые файлы в `recipes` и `mcb-kjv` отформатированы с шагом табуляции в четыре символа. Если в вашем редакторе записи файла выровнены неправильно, попробуйте установить такой же шаг табуляции. При печати можете обработать файлы командой *expand -4*. Предположим, что обычно вы печатаете так:

```
% pr filename | lpr
```

Тогда попробуйте такой способ:

```
% expand -4 filename | pr | lpr
```

Третий источник информации, используемой в примерах, – это база данных бейсбольного архива. Если вы хотите выполнить примеры, основанные на данных оттуда, посетите веб-сайт архива:

http://baseball1.com/

В примерах использована версия 4.5 этой базы данных, доступная в форматах Access 2000, Access 97 и в текстовых файлах с разделителями-запятыми. Инструкции по загрузке в MySQL файлов такого формата находятся в каталоге *baseball1* дистрибутива `recipes` (см. также рецепт 10.36).

Получение MySQL и сопутствующего ПО

Если вы рассчитываете работать с уже установленным где-либо сервером MySQL, то вам достаточно просто иметь клиентское ПО MySQL на своей машине. Если же вы собираетесь запускать собственный сервер, то понадобится полный дистрибутив MySQL.

Для написания программ, взаимодействующих с MySQL, необходим API для соответствующего языка. Программные интерфейсы языков Perl, PHP и Python базируются на клиентской библиотеке C API, реализующей низкоуровневый клиент-серверный протокол. PHP включает файлы поддержки клиента MySQL. В драйверах JDBC для Java реализован собственный клиент-серверный протокол, поэтому они не зависят от клиентской C-библиотеки.

Возможно, вам не придется устанавливать клиентские программы самостоятельно – это могут сделать за вас другие. Обычно так бывает, когда вы подключены к интернет-провайдеру, предоставляющему веб-хостинг с поддержкой MySQL. В таких случаях библиотеки и заголовочные файлы MySQL устанавливает персонал провайдера.

MySQL

За дистрибутивом сервера MySQL обратитесь на его домашний сайт:

http://www.mysql.com/

Дистрибутивы включают инструкции по установке, которые имеются и в справочном руководство по MySQL. Это руководство доступно на сайте в электронном виде, имеется также печатное издание O'Reilly & Associates. Если вы планируете использовать соединения ODBC, вам потребуется MyODBC.

Если вам необходимо установить клиентскую C-библиотеку и заголовочные файлы, воспользуйтесь дистрибутивом в исходных текстах или двоичным дистрибутивом не-RPM для Unix. В Linux есть возможность установки MySQL из RPM-файлов, но имейте в виду, что клиентская библиотека и заголовочные файлы имеются только в RPM-файлах для разработчиков. (Существуют различные RPM-файлы для сервера, для стандартных клиентских программ и для библиотеки разработки с заголовочными файлами.) Если вы не установите RPM-файлы для разработчиков, то пополните ряды пользователей Linux, которые жалуются: «Я установил MySQL, но не могу найти библиотеки и заголовочные файлы – где они?».

Поддержка Perl

Общая информация о Perl доступна по адресу:

http://www.perl.org/

Программное обеспечение Perl можно получить из сетевого архива CPAN (Comprehensive Perl Archive Network):

http://cpan.perl.org/

Для создания Perl-программ, работающих с MySQL, необходим модуль DBI и специальный модуль DBD для MySQL, *DBD::mysql*.

Чтобы установить эти модули в UNIX, проще всего доверить дело самому Perl. Например, для установки DBI и *DBD::mysql* выполните следующие команды (вероятно, от имени пользователя `root`):

```
# perl -MCPAN -e shell
cpan> install DBI
cpan> install DBD::mysql
```

Если вы работаете в ActiveState Perl для Windows, то можете использовать команду *ppm*:

```
C:\> ppm
ppm> install DBI
ppm> install DBD::mysql
```

Для установки других модулей Perl, упомянутых в книге, также можно использовать оболочку CPAN или *ppm*.

Поддержка PHP

Дистрибутивы PHP и инструкции по установке находятся по адресу:

http://www.php.net/

Дистрибутив PHP в исходных текстах содержит клиентскую библиотеку MySQL, поэтому нет необходимости устанавливать ее отдельно. Если вы используете двоичный дистрибутив, убедитесь, что в него включена поддержка MySQL.

Поддержка Python

Дистрибутивы Python и инструкции по установке находятся по адресу:

http://www.python.org/

Модуль драйвера DB-API, MySQLdb, обеспечивающий поддержку MySQL, доступен по адресу:

http://sourceforge.net/projects/mysql-python/

Поддержка Java

Для компилирования и выполнения программ на Java вам понадобится компилятор. Это может быть *javac* или *jikes*. Во многих системах они уже установлены. Если это не так, вы можете найти компилятор в Java Software Development Kit (SDK). Если SDK в вашей системе не установлен, то его версии для Solaris, Linux и Windows вы найдете на сайте компании Sun, посвященном Java:

http://java.sun.com/js2e/

Интерфейс JDBC (Java Database Connectivity) для MySQL реализован в нескольких драйверах. Используемый в этой книге MySQL Connector/J (ранее называвшийся MM.MySQL) находится по адресу:

http://www.mysql.com/downloads/

Веб-серверы

В главах, посвященных веб-програмированию, сценарии на Perl, PHP и Python предполагают использование Apache, а сценарии на JavaServer Pages – Tomcat. Оба сервера, Apache и Tomcat, доступны на сайте Apache Software Group по адресам:

http://httpd.apache.org/
http://jakarta.apache.org/

На втором сайте имеется и реализация библиотеки тегов JSP Standard Tag Library, которая в этой книге использовалась при написании страниц JSP:

http://jakarta.apache.org/taglibs/

Прочие программы

В главе 10 упоминалось приложение для Windows DBTools, используемое при передаче данных в/из MySQL. Вы найдете DBTools на сайте:

http://dbtools.vila.bol.com.br/

Библиотека класса Java, `gnu.getopt.Getopt`, которая обсуждалась в рецепте 2.10, доступна по адресу:

http://www.urbanophile.com/arenn/hacking/download.html

Сайт проекта Jakarta, на котором хранится Tomcat, также обеспечивает доступ к библиотеке класса регулярных выражений, которая использовалась в книге для выполнения операций сравнения с образцом:

http://jakarta.apache.org/oro/

B

JSP и Tomcat для начинающих

В приложении описаны основы JSP-программирования (JavaServer Pages), о котором мы говорили в книге, начиная с главы 16. Необходимые базовые знания достаточно обширны, поэтому материалы представлены в специальном приложении, чтобы не перегружать ими главу и не нарушать ход повествования. Рассматриваются следующие аспекты:

- Краткий обзор технологий сервлетов и JSP
- Настройка сервера Tomcat
- Структура каталогов Tomcat
- Структура веб-приложений
- Элементы JSP-страницы

Дополнительную информацию о страницах JSP, сервлетах и сервере Tomcat вы найдете в источниках, указанных в приложении С.

Обзор сервлетов и страниц JavaServer

Технология сервлетов Java – это средство эффективного исполнения Java-программ в веб-окружении. Спецификация Java Servlet определяет правила поведения в этом окружении, которые можно кратко подытожить так:

- Сервлеты работают внутри контейнера сервлетов, который сам работает внутри веб-сервера или взаимодействует с ним. Контейнеры сервлетов также называют средой исполнения сервлетов (servlet engines).
- Контейнер сервлета получает запросы от веб-сервера и выполняет соответствующий сервлет для обработки запроса. Контейнер получает ответ от сервлета и передает его веб-серверу, который возвращает его клиенту. То есть контейнер сервлета обеспечивает связь между сервлетами и веб-сервером, под управлением которого они работают. Контейнер действует как среда исполнения сервлетов, которая устанавливает соответствие запросов клиентов обрабатывающим их сервлетам, а также при необходимости загружает, исполняет и выгружает сервлеты.

- Сервлеты взаимодействуют со своим контейнером согласно заданным правилам. Каждый сервлет должен реализовывать методы с известными именами, вызываемые в ответ на различные виды запросов. Например, методы `GET` и `POST` отправляются методам `doGet()` и `doPost()`.
- Сервлеты, которые могут запускаться контейнером, разбиты на логические группы, называемые «контекстами». (Контексты могут соответствовать, например, подкаталогам дерева документов, управляемого контейнером.) Контексты могут включать в себя не только сервлеты, но и другие виды ресурсов, например, HTML-страницы, изображения или конфигурационные файлы.
- Контекст является основой для «приложения», то есть группы связанных сервлетов, работающих вместе, без вмешательства других неродственных сервлетов. Сервлеты внутри одного контекста приложения могут совместно использовать информацию, в отличие от сервлетов из разных контекстов. Например, сервлет шлюза или входа в систему должен проверить мандат пользователя, который затем помещается в окружение контекста для того, чтобы другие сервлеты того же контекста могли использовать его как подтверждение корректной регистрации пользователя. Если сервлеты при исполнении обнаружат, что в окружении приложения нет соответствующих мандатов, они могут отправить сервлету шлюза запрос на регистрацию пользователя. Сервлеты другого контекста не имеют доступа к таким мандатам. Тем самым контексты обеспечивают меры безопасности, не допуская вторжения одного приложения в другое. Они также могут изолировать приложения от последствий аварийного завершения других приложений. Контейнер может поддерживать неповрежденные приложения в рабочем состоянии, пока неисправное приложение будет перезапускаться.
- Совместное использование информации сервлетами возможно на разных уровнях, что позволяет им работать вместе в рамках одного запроса или на протяжении нескольких запросов.

Посмотрим, как выглядит простой сервлет. Это программа на Java, реализующая класс `SimpleServlet`. У класса есть метод `doGet()`,вызываемый контейнером сервлета при получении запроса `GET` для сервлета. У него также есть метод `doPost()` для обработки возможного запроса `POST`, содержащий внутри себя просто вызов метода `doGet()`. `SimpleServlet` формирует небольшую HTML-страницу, содержащую статический текст, остающийся неизменным при каждом исполнении сервлета, и два динамических элемента (текущую дату и IP-адрес клиента), которые меняются с течением времени и от клиента к клиенту:

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SimpleServlet extends HttpServlet
{
```

```
    public void doGet (HttpServletRequest request,
                       HttpServletResponse response)
        throws IOException, ServletException
    {
        PrintWriter out = response.getWriter ();

        response.setContentType ("text/html");
        out.println ("<html>");
        out.println ("<head>");
        out.println ("<title>Simple Servlet</title>");
        out.println ("</head>");
        out.println ("<body bgcolor=\"white\">");
        out.println ("<p>Hello.</p>");
        out.println ("<p>The current date is "
                     + new Date ()
                     + ".</p>");
        out.println ("<p>Your IP address is "
                     + request.getRemoteAddr ()
                     + ".</p>");
        out.println ("</body>");
        out.println ("</html>");
    }

    public void doPost (HttpServletRequest request,
                       HttpServletResponse response)
        throws IOException, ServletException
    {
        doGet (request, response);
    }
}
```

Вы наверняка обратили внимание на то, что этот «простой» сервлет не так уж прост! Необходимо выполнить множество действий для импорта требуемых классов и создания методов `doGet()` и `doPost()`, предоставляющих стандартный интерфейс к контейнеру сервлета. Сравните сервлет со сценарием PHP, который делает то же самое гораздо лаконичнее:

```
<html>
<head>
<title>Simple PHP Page</title>
</head>
<body bgcolor="white">

<p>Hello.</p>
<p>The current date is <?php print (date ("D M d H:i:s T Y")); ?>.</p>
<p>Your IP address is <?php print ($_SERVER["REMOTE_ADDR"]); ?>.</p>

</body>
</html>
```

Очевидная разница между сервлетом Java и сценарием PHP является отражением одной из проблем сервлетов – объем повторяющейся дополнительной работы:

- Некоторый минимальный набор классов должен быть импортирован в каждый сервлет.
- Создание класса сервлета и метода `doGet()` или `doPost()` является достаточно стандартной операцией, зачастую изменяется лишь имя класса сервлета.
- Каждый фрагмент HTML формируется при помощи оператора вывода.

Две первые задачи можно решить, используя файл прототипа, который копируется при начале нового сервлета. С третьим вопросом (упаковкой каждой строки HTML в оператор вывода) не так легко разобраться. Вероятно, это самый трудоемкий и утомительный этап создания сервлета. Кроме того, возникает еще одна проблема: код сервлета может быть достаточно удобочитаемым как код Java, но разглядеть формируемую им структуру HTML бывает довольно сложно. Все дело в том, что вы фактически пытаетесь одновременно писать на двух языках (то есть пишете код Java, который пишет код HTML), что неблагоприятно сказывается на них обоих.

Страницы JSP – альтернатива сервлетам

Одной из причин изобретения технологии JavaServer Pages было желание освободиться от необходимости создания веб-страниц посредством операторов вывода. JSP применяет тот же подход, что и PHP: код HTML пишется буквально, без использования операторов вывода, а исполняемый код встраивается в страницу внутри специальных маркеров. Вот страница JSP, эквивалентная сервлету `SimpleServlet`, но гораздо более похожая на соответствующий сценарий PHP:

```
<html>
<head>
<title>Simple JSP Page</title>
</head>
<body bgcolor="white">

<p>Hello.</p>
<p>The current date is <%= new java.util.Date () %>.</p>
<p>Your IP address is <%= request.getRemoteAddr () %>.</p>

</body>
</html>
```

Страница JSP лаконичнее сервлета по нескольким причинам:

- Не нужно импортировать стандартный набор классов, необходимый для запуска сервлета.
- HTML записывается более естественным способом, без применения операторов вывода.
- Не нужны определения классов, а также методы `doGet()` и `doPost()`.
- Объекты `response` и `out` не нуждаются в объявлении, так как они уже созданы для вас и готовы к использованию как неявные объекты. На самом деле только что приведенная страница JSP вообще явно не ссылалась на `out`, так как ее конструкции формирования вывода пишут в `out` автоматически.

- Содержимое по умолчанию имеет тип `text/html`; не нужно задавать его явно.
- Код Java помещается непосредственно в сценарий внутри специальных маркеров. В рассмотренной странице используются маркеры `<%=` и `%>`, означающие «вычислить выражение и вывести результат». Существуют и другие маркеры, каждый из который имеет определенное назначение. Они кратко описаны далее в разделе «Элементы страниц JSP».

Когда контейнер сервлета получает запрос на JSP-страницу, он воспринимает страницу как шаблон, содержащий буквальный текст и встроенный исполняемый код, заключенный в специальные маркеры. Контейнер формирует из шаблона страницу, отправляемую клиенту. Текст остается неизменным, а исполняемый код заменяется формируемым им выводом, и объединенные результаты возвращаются клиенту в качестве ответа на запрос. По крайней мере, именно такова теория обработки JSP. Поговорим теперь о том, что происходит в действительности, когда контейнер обрабатывает запрос к JSP:

- Страница JSP транслируется в сервлет, то есть в эквивалентную Java-программу. Вхождения текста шаблона преобразуются в операторы вывода, которые отображают текст буквально. Вхождения кода помещаются в программу так, чтобы выполняться с ожидаемым эффектом. Все это заключается в оболочку, которая предоставляет уникальное имя класса и включает в себя предложения `import` для извлечения стандартного набора классов, необходимых для корректной работы сервлетов в веб-окружении.
- Контейнер компилирует сервлет для формирования исполняемого файла класса.
- Контейнер исполняет файл класса для генерирования страницы вывода, которая возвращается клиенту в качестве ответа на запрос.
- Контейнер также кэширует исполняемый класс, с тем чтобы при получении следующего запроса на JSP-страницу можно было сразу исполнять класс, пропустив этапы трансляции и компиляции. Если контейнер замечает, что страница JSP была изменена, при следующем запросе он отказывается от кэшированного класса и заново компилирует измененную страницу в новый исполняемый класс.

Что касается представления веб-страницы, то с помощью JSP они записываются естественнее, чем посредством сервлетов. Преимущество среды исполнения JSP заключается в автоматической компиляции после создания страницы в дереве документов или ее последующего изменения. Когда вы пишете сервлет, его необходимо компилировать после каждого изменения, выгружая старый и загружая новый. Может случиться так, что основные усилия будут уходить на обслуживание сервлетов в ущерб их функциональности. JSP смещает акценты: у вас появляется больше возможности думать о том, что делает страница, а не о том, как ее правильно скомпилировать и загрузить.

Отличия между сервлетами и страницами JSP не подразумевают необходимости выбора между технологиями. Контексты приложений в контейнере

сервлета могут содержать и те, и другие, а поскольку страницы JSP в любом случае преобразуются в сервлеты, все они могут общаться между собой.

Технология JSP похожа на некоторые другие, что упрощает миграцию от них на JSP. Например, подход JSP имеет много общего с технологией Active Server Pages (ASP) от Microsoft. Однако JSP не зависит от поставщика и платформы, в то время как страницы ASP являются запатентованными. Так что JSP – это привлекательный вариант для тех, кто хочет отказаться от ASP.

Пользовательские действия (custom actions) и библиотеки тегов

Сервлет очень похож на Java-программу, собственно, он и есть Java-программа. Использование JSP способствует разделению HTML (представления) и кода, вам не нужно формировать HTML внутри операторов вывода Java. Но JSP *не требует* отделения HTML от кода, так что если вы не будете соблюдать осторожность, в странице опять-таки может оказаться встроенный код Java.

Можно избежать включения кода Java в страницы JSP, используя такую функциональность JSP, как custom actions – пользовательские действия. Речь идет о специальных тегах, очень напоминающих теги HTML (они также записываются как элементы XML). Пользовательские действия позволяют определять теги, которые будут действовать от имени страницы, на которой они указаны. Например, тег `<sql:query>` может взаимодействовать с сервером базы данных для запуска запроса. Пользовательские действия обычно используются группами, которые называются библиотеками тегов и создаются так:

- Выполняемые тегами действия реализуются набором классов. Это обычные классы Java, созданные по специальным правилам, позволяющим контейнеру сервлета взаимодействовать с ними стандартным способом. (Правила, например, определяют, как атрибуты тега и содержимое его тела передаются классам обработчика тегов.) Обычно набор классов упаковывается в файл JAR.
- Библиотека включает в себя файл дескриптора TLD (Tag Library Descriptor – дескриптор библиотеки тегов), который указывает, какие теги сопоставлены каким классам. Это позволяет процессору JSP определить, какой класс следует вызывать для каждого пользовательского тега страницы JSP. Файл TLD также указывает, как ведет себя каждый тег, например, требуются ли ему атрибуты. Такая информация используется при преобразовании страницы для проверки корректности использования тегов библиотеки страницей JSP. Например, если тегу необходим определенный атрибут, а на странице тег используется без него, то процессор JSP может выявить проблему и выдать соответствующее сообщение об ошибке.

Благодаря библиотекам тегов можно написать всю страницу, используя только теги, а не переходя от тегов к коду Java и обратно. Нотация больше напоминает JSP, чем Java, но эффект размещения на JSP-странице пользо-

вательского тега аналогичен вызову метода. Это объясняется тем, что тег на странице JSP отображается на вызов метода сервлета, в который преобразуется страница.

Чтобы проиллюстрировать разницу между использованием встроенного кода Java и библиотеки тегов, сравним два сценария, устанавливающих соединение с сервером MySQL и выводящих список таблиц базы данных `cookbook`. Первый сценарий применяет внутри страницы встроенный код Java:

```
<%@ page import="java.sql.*" %>

<html>
<head>
<title>Tables in cookbook Database</title>
</head>
<body bgcolor="white">

<p>Tables in cookbook database:</p>

<%
    Connection conn = null;
    String url = "jdbc:mysql://localhost/cookbook";
    String user = "cbuser";
    String password = "cbpass";

    Class.forName ("com.mysql.jdbc.Driver").newInstance ();
    conn = DriverManager.getConnection (url, user, password);

    Statement s = conn.createStatement ();
    s.executeQuery ("SHOW TABLES");
    ResultSet rs = s.getResultSet ();
    while (rs.next ())
        out.println (rs.getString (1) + "<br />");
    rs.close ();
    s.close ();
    conn.close ();
%>

</body>
</html>
```

То же самое можно сделать при помощи библиотеки тегов, такой как стандартная библиотека тегов JSP (JSTL). JSTL состоит из нескольких наборов тегов, сгруппированных по функциям. При помощи ее тегов ядра и баз данных можно преобразовать предыдущую страницу JSP так, чтобы избежать использования собственно кода Java:

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jstl/sql" prefix="sql" %>

<html>
<head>
<title>Tables in cookbook Database</title>
</head>
<body bgcolor="white">
```

```
<p>Tables in cookbook database:</p>

<sql:setDataSource var="conn"
    driver="com.mysql.jdbc.Driver" url="jdbc:mysql://localhost/cookbook"
    user="cbuser" password="cbpass" />

<sql:query var="rs" dataSource="${conn}">SHOW TABLES</sql:query>
<c:forEach var="row" items="${rs.rowsByIndex}">
    <c:out value="${row[0]}" /><br />
</c:forEach>

</body>
</html>
```

Директивы `taglib` определяют используемые страницей файлы TLD и указывают, что действия соответствующих наборов тегов будут идентифицироваться префиксами `c` и `sql`. (В сущности, префикс создает пространство имен для набора тегов.) Тег `<sql:dataSource>` задает параметры соединения с сервером MySQL, `<sql:query>` запускает запрос, `<c:forEach>` просматривает результат, а `<c:out>` добавляет имя каждой таблицы результата на страницу вывода. (Не будем сейчас говорить о деталях, теги JSTL подробно описаны в рецепте 16.3.)

Если есть вероятность того, что большинство JSP-страниц вашего контекста приложения будет соединяться с сервером базы данных одинаково, можно переместить тег `<sql:dataSource>` во включаемый файл. Назовем файл *jstl-mcb-setup.inc* и поместим его в каталог приложения *WEB-INF*,[1] тогда любая страница контекста приложения сможет установить соединение с сервером MySQL, обратившись к файлу при помощи директивы `include`. Изменим предыдущую страницу, использовав включаемый файл, – получится такой сценарий:

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jstl/sql" prefix="sql" %>
<%@ include file="/WEB-INF/jstl-mcb-setup.inc" %>

<html>
<head>
<title>Tables in cookbook Database</title>
</head>
<body bgcolor="white">

<p>Tables in cookbook database:</p>

<sql:query var="rs" dataSource="${conn}">SHOW TABLES</sql:query>
<c:forEach var="row" items="${rs.rowsByIndex}">
    <c:out value="${row[0]}" /><br />
</c:forEach>

</body>
</html>
```

[1] Приложения обычно используют свой каталог *WEB-INF* для частной информации определенного контекста. См. раздел «Структура веб-приложения».

Работая с библиотекой тегов, вы продолжаете использовать Java, так как действия тегов отображаются в вызовы классов Java. Но нотация следует правилам XML, поэтому кажется, что пишется не программный код, а, скорее, создаются элементы HTML-страницы. Если в вашей организации технологический процесс формирования веб-содержимого подразумевает разделение обязанностей, то пользовательские теги позволяют упаковывать динамически генерируемые элементы страницы так, чтобы дизайнерам и другим не-программистам было легче с ними работать. Им не придется разрабатывать и применять сами классы, реализующие действия тегов, этим занимаются программисты, создающие соответствующие тегам классы.

Настройка сервера Tomcat

В предыдущем разделе были приведены некоторые базовые сведения о сервлетах и страницах JSP, но ничего не говорилось о том, как заставить сервер исполнять их. Теперь же поговорим о том, как установить Tomcat, веб-сервер, умеющий работать с JSP. Сервер Tomcat входит в проект Jakarta Project, наряду с Apache являясь детищем Apache Software Foundation.

Как уже говорилось, сервлеты исполняются внутри контейнера, который является средой, взаимодействующей или встраивающейся в веб-сервер для обработки запросов страниц, выводимых в результате выполнения сервлетов. Некоторые контейнеры сервлетов могут работать автономно, выполняя одновременно функции и контейнера, и веб-сервера. Именно так работает Tomcat, поэтому, установив его, вы получите полнофункциональный сервер с возможностью обработки сервлетов. Фактически Tomcat создан в качестве образца реализации спецификаций сервлетов и страниц JSP, поэтому он также служит и средой исполнения страниц JSP, обеспечивая их трансляцию в сервлеты. Часть, отвечающая за контейнеры сервлетов, называется Catalina, а за обработку страниц JSP – Jasper.

«Контейнерную» часть Tomcat можно использовать совместно с другими веб-серверами. Например, можно установить такую связку из Apache и Tomcat, в которой Apache будет выступать в качестве внешнего интерфейса для Tomcat, передавая ему запросы к сервлетам и JSP-страницам, а все остальные обрабатывать самостоятельно. Информацию о такой организации совместной работы серверов Apache и Tomcat можно найти на сайте проекта Jakarta.

Для запуска сервера Tomcat необходимо следующее:

Инструментарий разработчика (Software Development Kit, SDK) Java. Нужен потому, что Tomcat должен компилировать сервлеты Java в процессе своей работы. Если вы, читая эту книгу, компилировали примеры на Java, то, скорее всего, SDK у вас уже установлен. Если нет, обратитесь к рецепту 2.1 за информацией о его получении и установке.

Собственно Tomcat. Дистрибутив Tomcat доступен на сайте проекта Jakarta *http://jakarta.apache.org*. Там же имеется изрядное количество документации для него. В частности, если вы еще не знакомы с Tomcat, я рекомендую

прочитать «Introduction» (Введение) и «Application Developer's Guide» (Руководство прикладного программиста).

Некоторое знание XML. Файлы конфигурации Tomcat представляют собой XML-документы, а многие элементы страниц JSP следуют синтаксическим правилам XML. Если вы не знаете язык XML, то обратитесь к примечанию «XML и XHTML в двух словах» в разделе 16.0, где XML сравнивается с HTML.

Установка сервера Tomcat

В настоящее время доступны версии 3.x и 4.x сервера Tomcat. В этом разделе описана установка Tomcat 4.x, который я и рекомендую устанавливать. Если вы работаете с версией 3.x, будьте готовы к отличиям. Например, структуры каталогов в этих двух версиях не совсем идентичны.

Для инсталляции Tomcat вы можете получить двоичный дистрибутив на сайте *jakarta.apache.org*. (Будем считать, что вы не захотите собирать дистрибутив из исходных текстов, так как это сложнее.) Дистрибутивы Tomcat доступны в файлах различных форматов, которые характеризуются расширением имени файла:

.tar.gz

Сжатый файл *tar*, обычно используемый для установок в UNIX.

.zip

Архив ZIP, который может применяться и в UNIX, и в Windows.

.rpm (RedHat Package Manager)

Эти файлы предназначены для Linux-систем.

.exe

Исполняемый инсталлятор, используется только в Windows.

Дистрибутив, упакованный в файл *tar* или ZIP, следует поместить в каталог, в котором планируется установить Tomcat, и распаковать его там соответствующей командой. Файлы RPM содержат внутри себя информацию о месте инсталляции, так что вы можете выполнять команду установки откуда угодно. Инсталлятор Windows типа *.exe* предлагает указать место установки Tomcat, так что и его можно запускать из любого каталога. Приведем команды, которые необходимы для инсталляции дистрибутивов любого вида. (Измените номера версий в именах файлов в соответствии с версией используемого дистрибутива.)

Для установки Tomcat из сжатого файла *tar* распакуйте его:

```
% tar zxf jakarta-tomcat-4.0.4.tar.gz
```

Если вы распаковываете дистрибутив `tar` в Mac OS X, используйте *gnutar* вместо *tar*. Доступны обе программы, но у *tar* могут возникнуть сложности с длинными именами файлов, а у *gnutar* таких проблем нет. В Solaris также может потребоваться использование GNU-совместимой версии *tar*.

Если ваша версия *tar* не понимает опцию *z*, выполните такую команду:

```
% gunzip jakarta-tomcat-4.0.4.tar.gz | tar xf -
```

Те, кто использует архив ZIP, могут распаковать его при помощи утилиты *jar* или любой другой программы, распознающей формат ZIP (например, приложения Windows *WinZip*). Используем *jar*:

```
% jar xf jakarta-tomcat-4.0.4.zip
```

Пользователи Linux имеют возможность установки из файлов RPM. Необходимо установить два RPM: первый – с программным обеспечением сервера, а второй – с приложениями. (Дистрибутивы в других форматах состоят из одного архивного файла, включающего и приложения.) Обычно команды установки RPM выполняются от имени `root`:

```
# rpm --install tomcat4-4.0.4-full.2jpp.noarch.rpm
# rpm --install tomcat4-webapps-4.0.4-full.2jpp.noarch.rpm
```

В зависимости от версии Tomcat вам может потребоваться предварительная установка других RPM-дистрибутивов, таких как *regex*, *servletapi* и *xerces*. Если это необходимо, *rpm* выведет соответствующее сообщение при инсталляции Tomcat.

После установки файлов RPM отредактируйте */etc/tomcat4/conf/tomcat4.conf*, присвоив переменной окружения `JAVA_HOME` значение пути к установке Java SDK.

Дистрибутив для Windows с расширением *.exe* является исполняемым. Запустите его и укажите, где хотите разместить Tomcat, когда инсталлятор задаст вам такой вопрос. Если вы выбрали именно этот инсталлятор, проверьте, установлен ли уже у вас Java SDK; инсталлятор помещает некоторые файлы в иерархию SDK, и при его отсутствии такая операция не будет выполнена. Для версий Windows, поддерживающих управление службами (Windows NT, 2000 или XP) инсталлятор *.exe* предлагает возможность установки Tomcat в качестве службы, чтобы он автоматически запускался при загрузке системы.

Большинство методов инсталляции создают каталог и распаковывают в него Tomcat. Каталогом верхнего уровня получившейся иерархии будет корневой каталог Tomcat. Я буду считать, что корневой каталог Tomcat – это */usr/local/jakarta-tomcat* в UNIX и *D:\jakarta-tomcat* в Windows. (В конце реальных имен каталогов, вероятно, будет указан номер версии.) Корневой каталог Tomcat содержит различные текстовые файлы с информацией, которая пригодится для решения общих или специфичных для платформы задач. Кроме того, в него входит ряд каталогов. Если вы хотите познакомиться с ними прямо сейчас, обратитесь к разделу «Структура каталогов Tomcat». Иначе переходим к следующему разделу «Запуск и останов Tomcat» и поговорим о работе Tomcat.

Имейте в виду, что если вы установили Tomcat из файлов RPM, то может оказаться, что дистрибутив «расползся», а не сосредоточен в одном каталоге.

Например, большинство составляющих можно обнаружить в */var/tomcat4*, а документацию – в каталоге */usr/doc/tomcat4*. Для того чтобы узнать, куда инсталлировано содержимое некоторого файла RPM, выполните такую команду:

```
% rpm -qpl rpmfile
```

(Это буква «l» в нижнем регистре, а не цифра «1».)

Запуск и останов Tomcat

Сервером Tomcat можно управлять вручную, а можно настроить его на автоматический запуск при старте системы. Давайте познакомимся с командами запуска и останова Tomcat, используемыми для вашей платформы, так как вполне вероятно, что вам придется выполнять эти операции очень часто (по крайней мере, при первоначальной настройке). Например, если вы изменяете конфигурационные файлы Tomcat или устанавливаете новое приложение, то необходимо перезапустить сервер, чтобы он заметил изменения.

Перед запуском Tomcat следует установить несколько переменных окружения. Переменная `JAVA_HOME` должна содержать путь к SDK, чтобы Tomcat мог его обнаружить, а `CATALINA_HOME` – путь к корневому каталогу Tomcat. (Tomcat 3.x использует `TOMCAT_HOME` вместо `CATALINA_HOME`, кроме того, может потребоваться установка `CLASSPATH`.)

Для запуска и останова Tomcat вручную в UNIX перейдите в каталог *bin* корневого каталога Tomcat. Используйте для управления Tomcat два сценария:

```
% ./startup.sh
% ./shutdown.sh
```

Для автоматического запуска Tomcat при загрузке системы найдите сценарий запуска */etc/rc.local* или */etc/rc.d/rc.local* (путь зависит от вашей операционной системы) и добавьте в него несколько строк:

```
export JAVA_HOME=/usr/local/java/jdk
export CATALINA_HOME=/usr/local/jakarta-tomcat
$CATALINA_HOME/bin/startup.sh &
```

Однако эти команды запустят Tomcat от имени пользователя `root`. Для запуска с учетной записью другого пользователя измените последнюю команду для вызова Tomcat через *su* и укажите имя пользователя:

```
su -c $CATALINA_HOME/bin/startup.sh имя_пользователя &
```

Если вы используете *su* для определения имени пользователя, то убедитесь в том, что дерево каталогов Tomcat доступно данному пользователю, или при попытке Tomcat обратиться к файлам вы столкнетесь с трудностями. Для этого можно, например, выполнить такую команду от имени `root` в корневом каталоге Tomcat:

```
# chown -R имя_пользователя .
```

Пользователи Linux, устанавливающие Tomcat из файлов RPM, увидят, что инсталляция создает сценарий *tomcat4* в каталоге */etc/rc.d/init.d*, который можно использовать для ручного или автоматического запуска. Если вы хотите запустить сценарий вручную, перейдите в этот каталог и выполните команды:

```
% ./tomcat4 start
% ./tomcat4 stop
```

Для автоматического запуска необходимо активизировать сценарий, выполнив следующую команду от имени `root`:

```
# chkconfig --add tomcat4
```

Инсталляция из RPM в Linux, помимо прочего, создает пользовательскую учетную запись с именем `tomcat4`, которая предназначена для работы с Tomcat.

Для пользователей Windows в каталоге *bin* имеется пара командных файлов, обеспечивающих ручное управление Tomcat:

```
C:\> startup.bat
C:\> shutdown.bat
```

Если вы установите Tomcat как службу для Windows-систем, поддерживающих управление службами (Windows NT, 2000 или XP), то сможете управлять им из консоли служб. Таким способом вы можете запускать и останавливать Tomcat, а также настроить его автоматический запуск при старте Windows. (Служба называется `Apache Tomcat`.)

Для опробования Tomcat запустите его, используя любую доступную на вашей платформе инструкцию. Затем запросите страницу по умолчанию в броузере. URL будет выглядеть примерно так:

http://tomcat.snake.net:8080/

Подставьте имя своего хоста и номер порта. Например, по умолчанию Tomcat обычно использует порт 8080, но если вы установили его из файлов RPM для Linux, то номером порта может быть 8180. Если броузер корректно получит страницу, то вы увидите логотип Tomcat и ссылки на примеры и документацию по теме. Полезно пройтись по ссылкам примеров и протестировать несколько страниц JSP, чтобы посмотреть, правильно ли они компилируются и исполняются.

Если окажется, что несмотря на установку переменной `JAVA_HOME` Tomcat не находит компилятор Java, то попробуйте изменить переменную окружения `PATH`, явно включив в нее каталог, содержащий компилятор. Обычно это каталог *bin* установки SDK. Вероятно, переменная `PATH` уже была установлена, тогда добавьте каталог *bin* в ее текущее значение:

```
export PATH=${PATH}:/usr/local/java/jdk/bin   (sh, bash, etc.)
setenv PATH ${PATH}:/usr/local/java/jdk/bin   (csh, tcsh, etc.)
set PATH=%PATH%;D:\jdk\bin                    (Windows)
```

Структура каталогов Tomcat

Для создания страниц JSP знакомство со структурой каталогов Tomcat обязательным не является. Но и вреда от этого точно не будет, так что переместитесь в корневой каталог Tomcat и посмотрите вокруг. Вы обнаружите множество стандартных каталогов, описанных далее, сгруппированных по функциям. Имейте в виду, что в вашей системе структура каталогов может быть несколько иной, особенно если вы работаете с Tomcat 3.x, а не 4.x. Даже для Tomcat 4.x несколько каталогов могут отсутствовать в дистрибутиве, а каталоги *logs* и *work* могут создаваться только при первом запуске Tomcat.

Каталоги приложений

С точки зрения разработчика приложений самой главной составляющей иерархии каталогов Tomcat является каталог *webapps*. У каждого контекста приложения есть собственный каталог, расположенный в каталоге *webapps* корневого каталога Tomcat.

Tomcat обрабатывает клиентские запросы, отображая их внутрь каталога *webapps*. Если запрос начинается с имени каталога, расположенного внутри *webapps*, Tomcat ищет соответствующую страницу в этом каталоге. Например, два следующих запроса будут обработаны при помощи страниц *index.html* и *test.jsp* каталога *mcb*:

http://tomcat.snake.net:8080/mcb/index.html
http://tomcat.snake.net:8080/mcb/test.jsp

Запросы, которые не начинаются с имени одного из подкаталогов *webapps*, обслуживаются специальным подкаталогом *ROOT*, в котором содержится контекст приложения по умолчанию.[1] В ответ на следующий запрос Tomcat отправит страницу *index.html* каталога *ROOT*:

http://tomcat.snake.net:8080/index.html

Приложения обычно упаковываются в файлы веб-архива (WAR), а Tomcat при запуске по умолчанию ищет файлы WAR, которые следует распаковать. То есть устанавливая приложение, вы копируете его WAR-файл в каталог *webapps*, перезапускаете Tomcat, и Tomcat распаковывает файл. Формат каталогов отдельных приложений описан в разделе «Структура веб-приложений».

Спецификация Java Servlet Specification определяет веб-приложение как набор сервлетов, HTML-страниц, классов и других ресурсов, которые образуют полноценное приложение на веб-сервере. Для Tomcat это, по существу, означает, что приложение соответствует подкаталогу *webapps*. Контейнеры сервлетов, такие как Tomcat, обеспечивают раздельное хранение контекстов, и практическим следствием такой структуры является то, что сценарии одного каталога приложения не могут внести беспорядок в каталог другого приложения.

---

[1] Каталог *Web-apps/ROOT* – это не то же самое, что корневой каталог Tomcat, являющийся каталогом верхнего уровня дерева Tomcat.

Каталоги конфигурации и управления

Два каталога содержат файлы конфигурации и управляющие файлы. Каталог *bin* хранит управляющие сценарии для запуска и останова Tomcat, а *conf* включает в себя файлы конфигурации Tomcat, записанные в виде XML-документов. Tomcat читает файлы конфигурации только при запуске. Если вы изменяете их, то следует перезапустить сервер, чтобы изменения вступили в силу.

Самый важный файл конфигурации – это *server.xml*, который управляет поведением Tomcat в целом. Файл *web.xml* задает настройки приложений по умолчанию. Этот файл используется в сочетании с собственным файлом *web.xml* приложения (если такой есть). Файл *tomcat-users.xml* определяет мандаты пользователей, имеющих доступ к защищенным функциям сервера, таким как приложение Manager, позволяющее управлять приложениями из броузера (см. раздел «Перезапуск приложений без перезапуска Tomcat»). Этот файл может быть заменен другими механизмами хранения пользовательской информации. Например, вы можете хранить записи пользователей Tomcat в MySQL. Обратитесь за инструкциями в каталог *tomcat* дистрибутива `recipes`.

Каталоги классов

Несколько каталогов Tomcat используется для файлов классов и библиотек. Они отличаются по функциям в зависимости от того, хотите ли вы их сделать видимыми для приложений, Tomcat, или для приложений и Tomcat:

classes

Эти файлы классов доступны приложениям, но не Tomcat.

common

Этот каталог имеет два подкаталога, *classes* и *lib*, для файлов классов и библиотек, которые должны быть доступны для приложений и Tomcat.

lib

Эти библиотеки классов видны приложениям, но не Tomcat.

server

Этот каталог имеет два подкаталога, *classes* и *lib*, для файлов классов и библиотек, которые должны быть видны Tomcat, но не приложениям.

Tomcat 3.x использует механизм загрузки классов, отличный от Tomcat 4.x, и имеет другую структуру каталога классов. В частности, в Tomcat 3.x нет каталога *common*.

Рабочие каталоги

Если рабочие каталоги не были созданы в рамках инсталляции сервера, Tomcat создает два таких каталога при первом запуске. Tomcat записывает файлы журнала в каталог *log*, а каталог *work* использует для временных файлов.

Файлы из каталога *logs* могут пригодиться при диагностике. Например, если есть подозрение на то, что Tomcat запущен некорректно, причина этого обычно занесена в один из файлов журнала (лог).

Когда Tomcat преобразует страницу JSP в сервлет и компилирует ее в исполняемый файл класса, он сохраняет получившиеся файлы *.java* и *.class* в каталоге *work*. (Когда вы только начинаете работу с JSP-страницами, может быть полезно обратиться к каталогу *work* и сравнить исходные страницы JSP с соответствующими сформированными Tomcat сервлетами.)

Перезапуск приложений без перезапуска Tomcat

Если вы изменяете страницу JSP, Tomcat автоматически перекомпилирует ее при следующем запросе этой страницы. Но если страница использует файл класса или JAR из каталога приложения *WEB-INF*, и вы обновляете один из них, то Tomcat естественным образом заметит изменения только при перезапуске.

Чтобы избежать перезапуска сервера при изменении приложения укажите для него в файле Tomcat *server.xml* элемент `<Context>` с атрибутом `reloadable`, установленным в `true`. Тогда Tomcat будет искать изменения не только в непосредственно запрашиваемых страницах JSP, но и в используемых страницами классах и библиотеках из каталога *WEB-INF*. Например, чтобы создать такой элемент `<Context>` для приложения *mcb*, можно добавить следующую строку в файл Tomcat *server.xml*:

```
<Context path="/mcb" docBase="mcb" debug="0" reloadable="true" />
```

Атрибуты элемента `<Context>` имеют следующее значение для Tomcat:

`path`

Указывает URL, сопоставленный страницам контекста приложения. Значение – это та часть URL, которая следует за именем хоста и номером порта.

`docBase`

Определяет расположение каталога контекста приложения относительно каталога *webapps* в дереве Tomcat.

`debug`

Устанавливает уровень отладки контекста. Значение ноль отменяет вывод отладочных сообщений; чем больше значение, тем более объемным будет вывод.

`reloadable`

Задает поведение Tomcat при рекомпиляции для случаев, когда клиент запрашивает страницу JSP, расположенную в контексте приложения. По умолчанию Tomcat перекомпилирует страницу только после того, как замечает изменение самой страницы. Установка атрибута `reloadable` в `true` сообщает Tomcat о том, что необходимо дополнительно проверять используемые классы и библиотеки каталога *WEB-INF* приложения.

После изменения *server.xml* для добавления элемента `<Context>` перезапустите Tomcat, и изменения вступят в силу.

Выполняемая Tomcat проверка изменений классов и библиотек может быть очень полезной на этапе разработки приложения за счет избавления от бесконечных перезапусков. Однако, как и следовало ожидать, автоматическая проверка классов требует значительных операционных расходов, в результате чего производительности значительно снижается. Поэтому такой подход лучше применять в системах разработки, но не в рабочих системах.

Есть другой способ заставить Tomcat распознавать изменения приложения без перезапуска всего сервера – использовать приложение Manager. Тогда вы сможете повторно загружать приложения по запросу из броузера без дополнительных расходов, связанных с установкой атрибута `reloadable`. Приложение Manager вызывается посредством указания пути */manager* в конце URL, по которому вы обращаетесь к своему серверу Tomcat. URL также включает в себя команду, которую вы хотите выполнить. Например, следующий запрос показывает, какие контексты работают в настоящее время:

http://tomcat.snake.net:8080/manager/list

Для остановки и перезапуска приложения *mcb* без перезагрузки Tomcat используйте такой URL:

http://tomcat.snake.net:8080/manager/reload?path=/mcb

Дополнительную информацию о приложении Manager и его командах вы можете получить по адресу:

http://jakarta.apache.org/tomcat/tomcat-4.0-doc/manager-howto.html

Этот документ может быть доступен и локально – выберите ссылку на документацию на домашней странице вашего сервера Tomcat. Обратите особое внимание на раздел, описывающий создание пользователя Tomcat с ролью `manager`, так как вам понадобятся имя и пароль для получения доступа к приложению Manager. По умолчанию пользовательские записи определятся в конфигурационном файле Tomcat *tomcat-users.xml*. Каталог *tomcat* дистрибутива `recipes` содержит сведения о хранении пользовательских записей Tomcat в MySQL.

Структура веб-приложения

Каждое веб-приложение соответствует одному контексту сервлетов и существует в виде набора ресурсов. Некоторые ресурсы видны пользователям, другие нет. Например, JSP-страницы приложения могут быть доступны клиентам, а файлы конфигурации, свойств и классов, используемые страницами JSP, могут быть скрыты. Расположение составляющих в иерархии приложения определяет их доступность клиентам. То есть помещая ресурс в определенный каталог, вы определяете его как общедоступный или частный.

Спецификация Java Servlet 2.3 определяет стандарт компоновки веб-приложения. Она задает для разработчиков приложений правила, указывающие,

куда что помещать и какие части приложения контейнер сделает доступными клиентам, а какие – скроет.

Каждое веб-приложение соответствует одному контексту сервлетов. В Tomcat они представлены подкаталогами каталога *webapps*, являющегося «родителем» всех веб-приложений. Внутри каталога приложения вы найдете подкаталог *WEB-INF* и, как правило, другие файлы: HTML-страницы, JSP-страницы или файлы изображений. Файлы, которые расположены в каталоге приложения верхнего уровня, являются общедоступными и могут запрашиваться клиентами. Особое значение имеет каталог *WEB-INF*. Его наличие сигнализирует Tomcat о том, что его родительский каталог на самом деле представляет собой приложение. То есть каталог *WEB-INF* – это единственная необходимая составляющая веб-приложения; он должен существовать, хотя бы пустым. Если *WEB-INF* не пуст, он обычно содержит конфигурационные файлы, классы и другую информацию, специфичную для данного приложения. Наиболее часто встречаются три его базовых компонента:

WEB-INF/web.xml
WEB-INF/classes
WEB-INF/lib

Файл *web.xml* представляет собой дескриптор установки веб-приложения. Он обеспечивает контейнеру стандартный способ распознавания того, как обрабатывать ресурсы, входящие в состав приложения. Дескриптор установки часто используется для определения поведения страниц JSP и сервлетов, настройки контроля доступа к защищенной информации, указания страниц сообщений об ошибках, которые должны использоваться в случае возникновения проблем, указания того, где можно найти библиотеки тегов.

Каталоги *classes* и *lib* внутри *WEB-INF* хранят библиотеки и файлы классов, а иногда также и другую информацию. Отдельные файлы классов попадают в *classes*, где используется структура каталогов, соответствующая иерархии классов. (Например, файл класса *MyClass.class*, реализующий класс `com.kitebird.jsp.MyClass`, будет сохранен в каталоге *classes/com/kitebird/jsp*.) Библиотеки классов, упакованные в файлы JAR, хранятся в каталоге *lib*. Tomcat автоматически просматривает каталоги *classes* и *lib* при обработке запросов страниц из приложения, что позволяет страницам использовать информацию, специфичную для приложения, без лишней суеты.

Особенность каталога *WEB-INF* еще и в том, что доступ к нему ограничен. Его содержимое доступно только сервлетам приложения и страницам JSP, обратиться к нему напрямую из броузера невозможно, так что в этом каталоге можно размещать информацию, которую не следует показывать пользователям. Например, вы можете хранить в *WEB-INF* файл свойств, содержащий параметры соединения с сервером базы данных. Если у вас есть приложение, разрешающее выгрузку файлов изображения на сервер одной страницей и последующую их загрузку другой страницей, поместите файлы изображений в каталог внутри *WEB-INF*, чтобы сделать их приватными. Так как Tomcat не предоставляет прямого доступа к содержимому *WEB-INF*, ваши страницы JSP могут реализовывать проверку доступа – определять,

кто имеет право выполнять операции с изображениями. (Простейшая линия поведения может заключаться в том, чтобы потребовать у клиентов ввода имени и пароля перед тем, как разрешать им выгружать страницы.) Достоинство каталога *WEB-INF* и в том, что он предоставляет хранилище приватных файлов, местоположение которого зафиксировано относительно корневого каталога приложения независимо от того, на какой машине устанавливается приложение.

Клиенты, которые идут на хитрость и, пытаясь получить доступ к приватным данным, выдают запросы, содержащие такие имена, как *Web-Inf*, обнаружат, что интерпретация этого имени чувствительна к регистру, причем даже в тех системах, в которых имена файлов не чувствительны к регистру (Windows или HFS+ в Mac OS X). В таких системах необходимо позаботиться о том, чтобы не создать каталог *WEB-INF* с именем типа *Web-Inf*, *web-inf* и т. д. Сама операционная система может считать такое имя совпадающим с *WEB-INF*, но Tomcat воспринимает их как различные. В результате ресурсы каталога будут недоступны вашим JSP-страницам. В Windows может потребоваться создание каталога *WEB-INF* из командной строки DOS. (Проводник Windows может не придерживаться регистра, использованного при создании или переименовании каталога, кроме того он необязательно отображает имена каталогов так же, как команда *DIR* для командной строки DOS.)

Мы описывали состав веб-приложения в терминах иерархии каталогов, так как это самый простой и понятный способ. Однако приложение не обязано быть представлено именно так. Обычно веб-приложения упаковываются в файл WAR, а расположение компонентов соответствует спецификации сервлета. Но некоторые контейнеры могут запускать приложение непосредственно из его файла WAR без распаковки. Более того, контейнер, который распаковывает файлы WAR, может извлекать их в любую выбранную им структуру файловой системы.

Tomcat применяет простейший подход – сохраняет приложение в файловой системе, используя структуру каталогов, повторяющую дерево каталогов, в котором файл был изначально создан. Давайте сравним структуру файла WAR с иерархией каталогов, создаваемых Tomcat при распаковке. Например, WAR-файл приложения *someapp* можно исследовать при помощи такой команды:

```
% jar tf someapp.war
```

Список путевых имен, выведенных командой, соответствует структуре каталога *someapp*, созданного Tomcat при распаковке файла внутри каталога *webapps*. Чтобы это проверить, выведем рекурсивно содержимое каталога *someapp*, используя одну из команд:

```
% ls -R someapp                    (UNIX)
C:\> dir /s someapp                (Windows)
```

Если вы вручную создаете контекст для приложения *myapp*, шаги должны быть такими (если вас интересует, какой должна быть результирующая иерархия приложения, см. каталог *tomcat/myapp* дистрибутива `recipes`):

- Перейдите в подкаталог *webapps* дерева каталогов Tomcat.
- Создайте в каталоге *webapps* каталог с тем же именем, что и приложение (*myapp*), затем перейдите в него.
- В каталоге *myapp* создайте каталог *WEB-INF*. Наличие этого каталога оповещает Tomcat о том, что *myapp* – это контекст приложения, поэтому его создание необходимо. Затем перезапустите Tomcat, чтобы он заметил появление нового приложения.
- Создайте небольшую тестовую страничку *page1.html* в каталоге *myapp*. Будем запрашивать ее из броузера, чтобы убедиться в том, что Tomcat отправляет страницы приложению. Пусть это будет простой файл HTML, нам сейчас не нужны трудности с встроенным кодом Java, библиотеками тегов и т. д.:

```
<html>
<head>
<title>Test Page</title>
</head>
<body bgcolor="white">
<p>
This is a test.
</p>
</body>
</html>
```

  Для обращения к странице используйте такой URL, подставив в него имя хоста вашего сервера и соответствующий номер порта:

  http://tomcat.snake.net:8080/myapp/page1.html

- Для того чтобы опробовать простую страницу JSP, создайте копию *page1.html* и назовите ее *page2.jsp*. Будет создана действительная страница JSP (хотя и не содержащая исполняемого кода), так что вы можете запросить ее и должны получить вывод, идентичный формируемому *page1.html*:

  http://tomcat.snake.net:8080/myapp/page2.jsp

- Скопируйте *page2.jsp* в *page3.jsp* и измените новую страницу так, чтобы она включала в себя встроенный код Java: добавьте пару строк, выводящих текущую дату и IP-адрес клиента:

```
<html>
<head>
<title>Test Page</title>
</head>
<body bgcolor="white">
<p>
This is a test.
The current date is <%= new java.util.Date() %>.
Your IP number is <%= request.getRemoteAddr () %>.
</p>
</body>
</html>
```

Метод `Date()` возвращает текущую дату, а `getRemoteAddr()` – IP-адрес клиента, полученный от объекта, сопоставленного клиентскому запросу. После завершения изменений запросите *page3.jsp* из броузера – вывод должен содержать текущую дату и IP-адрес хоста, с которого запрашивалась страница.

Теперь у вас есть простой контекст приложения, состоящий из трех страниц (одна из которых содержит исполняемый код) и пустого каталога *WEB-INF*. Для большинства приложений *WEB-INF* будет хранить файл *web.xml*, который служит файлом дескриптора установки веб-приложения и сообщает Tomcat, как сконфигурировано приложение. Если вы посмотрите на файлы *web.xml* других приложений, устанавливаемых в Tomcat, то обнаружите, что они могут быть очень сложными. Простейший файл дескриптора установки выглядит так:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web- Application 2.3//EN"
    "http://java.sun.com/dtd/Web-app_2_3.dtd">

<web-app>

</web-app>
```

Для добавления новой информации в файл *web.xml* поместите новые элементы между тегами `<web-app>` и `</web-app>`. В качестве простого примера вы можете добавить элемент `<welcome-file-list>` для определения списка файлов, которые Tomcat должен просматривать, когда клиент присылает запрос URL, заканчивающийся на *myapp* без указания конкретной страницы. Тот файл из списка, который Tomcat находит первым, становится страницей по умолчанию, которая и отправляется клиенту. Например, для того чтобы Tomcat считал допустимыми страницами по умолчанию *page3.jsp* и *index.html*, создайте такой файл *web.xml* в каталоге *WEB-INF*:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/Web-app_2_3.dtd">

<web-app>
  <welcome-file-list>
    <welcome-file>page3.jsp</welcome-file>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
</web-app>
```

Перезапустите Tomcat, чтобы он прочитал новую конфигурацию приложения, затем запустите запрос, в котором явно не указана страница:

http://tomcat.snake.net:8080/myapp/

Каталог *myapp* содержит страницу *page3.jsp*, указанную в списке страниц по умолчанию в файле *web.xml*, поэтому Tomcat должен выполнить *page3.jsp* и отправить результат в ваш броузер.

Элементы страниц JSP

В предыдущем разделе приложения были описаны некоторые общие характеристики страниц JSP. Теперь мы более подробно поговорим о типах используемых конструкций.

Страницы JSP – это шаблоны, состоящие из статических и динамических частей:

- Буквальный текст на странице JSP, который не заключен в специальные маркеры, является статическим – он отправляется клиенту «как есть», неизменным. JSP-примеры в книге формируют HTML-страницы, поэтому статические составляющие сценариев JSP написаны на HTML. Но вы можете создавать страницы, которые будут генерировать вывод и в другой форме – простой текст, XML или WML.
- Нестатические (динамические) части JSP-страниц состоят из кода, который должен быть вычислен. Код отличается от статического текста тем, что он заключается в специальные маркеры. Некоторые маркеры определяют директивы обработки страниц или скриптлеты. Директива предоставляет процессору JSP информацию о том, как обрабатывать страницу, а скриптлет – это мини-программа, которая вычисляется и заменяется своим выводом. Другие маркеры могут быть тегами, записанными как XML-элементы. Они сопоставляются классам, действующим как обработчики тегов для выполнения нужных операций.

В последующих разделах будут рассмотрены различные виды динамических элементов, которые могут содержать страницы JSP.

Элементы сценариев

Несколько наборов маркеров сценариев позволяют встраивать в JSP-страницу код Java или комментарии:

`<% ... %>`

Маркеры `<%` и `%>` указывают скриптлет, то есть встроенный код Java. Следующий скриптлет вызывает `print()` для записи значения в страницу вывода:

```
<% out.print (1+2); %>
```

`<%= ... %>`

Эти маркеры указывают выражение, которое должно быть вычислено. Результат добавляется на страницу вывода, что упрощает отображение значений без использования явных операторов вывода. Например, обе предложенные ниже конструкции выводят значение `3`, но второе написать проще:

```
<% out.print (1+2); %>
<%= 1+2 %>
```

`<%! ... %>`

Маркеры `<%!` и `%>` обеспечивают объявление переменных классов и методов.

```
<%-- ... --%>
```

Текст внутри таких маркеров интерпретируется как комментарий и игнорируется. Комментарии JSP полностью исчезают и не появляются в выводе, возвращаемом клиенту. Если вы пишете страницу JSP, выводящую HTML, и хотите, чтобы комментарии присутствовали на итоговой странице вывода, используйте комментарий HTML:

```
<%-- Такой комментарий не войдет в итоговую страницу вывода --%>
<!-- Такой комментарий войдет в итоговую страницу вывода -->
```

Когда JSP-страница преобразуется в сервлет, все элементы сценария входят в него как составные части. То есть переменную, объявленную в одном элементе, могут использовать другие элементы далее в странице. Это также означает, что если вы объявляете переменную в двух элементах, то полученный сервлет будет некорректным, и возникнет ошибка.

Маркеры `<% ... %>` и `<%! ... %>` можно использовать для объявления переменных, но результаты их применения различны. Переменная, объявленная внутри `<% ... %>`, является переменной объекта (или экземпляра), она инициализируется при каждом запросе страницы. Переменная, объявленная внутри `<%! ... %>`, – это переменная класса, которая инициализируется только в начале жизни страницы. Рассмотрим JSP-страницу *counter.jsp*, которая объявляет `counter1` как переменную объекта, а `counter2` – как переменную класса:

```
<%-- counter.jsp -использование для счетчика переменной объекта и класса --%>

<% int counter1 = 0; %>          <%-- переменная объекта --%>
<%! int counter2 = 0; %>         <%-- переменная класса --%>
<% counter1 = counter1 + 1; %>
<% counter2 = counter2 + 1; %>
<p>Counter 1 is <%= counter1 %></p>
<p>Counter 2 is <%= counter2 %></p>
```

Если вы устанавливаете страницу и запрашиваете ее несколько раз, то значение `counter1` будет равно 1 для каждого запроса. Значение `counter2` возрастает от запроса к запросу (даже если к странице обращаются разные клиенты) до тех пор, пока Tomcat не будет перезапущен.

В дополнение к объявляемым вами самостоятельно переменным страницы JSP имеют доступ к объектам, которые объявляются неявно для вас (см. «Неявные объекты JSP»).

Директивы JSP

Маркеры `<%@` и `%>` задают директиву JSP, предоставляющую процессору JSP информацию о том, какой тип вывода формирует страница, какие классы и библиотеки тегов ей необходимы и т. д.

```
<%@ page ... %>
```

Директивы `page` выводят различные виды информации, задаваемой парами *атрибут="значение"*, следующими за ключевым словом `page`. Следующая

директива указывает, что язык сценариев страницы – Java, а тип содержимого страницы вывода – `text/html`:

```
<%@ page language="java" contentType="text/html" %>
```

На самом деле эту конкретную директиву вообще не нужно указывать, так как `java` и `text/html` являются значениями по умолчанию для соответствующих атрибутов.

Если страница JSP формирует вывод не-HTML, не забудьте изменить тип содержимого по умолчанию. Например, если страница генерирует простой текст, используйте такую директиву:

```
<%@ page contentType="text/plain" %>
```

Атрибут `import` вызывает импорт классов Java. В обычной Java-программе вы используете предложение `import`, а в странице JSP вместо него работает директива `page`:

```
<%@ page import="java.util.Date" %>
<p>The date is <%= new Date () %>.</p>
```

Если вы только единожды ссылаетесь на какой-то класс, то, вероятно, удобнее не указывать директиву и просто сослаться на класс по его полному имени в момент использования:

```
<p>The date is <%= new java.util.Date () %>.</p>
```

`<%@ include ... %>`

Директива `include` включает содержимое файла в процесс трансляции страницы. То есть директива заменяется содержимым включаемого файла, который затем транслируется. Следующая директива приводит к включению файла *my-setup-stuff.inc* из каталога приложения *WEB-INF*:

```
<%@ include file="/WEB-INF/my-setup-stuff.inc" %>
```

Начальный символ / указывает имя относительно каталога приложения (контекстно-зависимый путь). Отсутствие начального символа / означает имя файла относительно местоположения страницы, содержащей директиву `include`.

Включаемые файлы обеспечивают совместное использование содержимого (как статического, так и динамического) набором JSP-страниц. Например, вы можете использовать их для задания стандартных заголовков или нижних колонтитулов для множества страниц JSP, или для выполнения кода общих операций, таких как установка соединения с сервером базы данных.

`<%@ taglib ... %>`

Директива `taglib` показывает, что страница использует пользовательские действия из указанной библиотеки тегов. Директива содержит атрибуты, сообщающие процессору JSP, как найти файл TLD библиотеки, и определяющие имя, которое далее на странице будет использоваться для тегов

данной библиотеки. Например, страница, использующая теги ядра и доступа к базе данных JSTL, может включать в себя такие директивы taglib:

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jstl/sql" prefix="sql" %>
```

Атрибут uri (Uniform Resource Identifier – уникальный идентификатор ресурса) уникально идентифицирует библиотеку тегов, так что процессор JSP может найти ее TLD-файл. Файл TLD определяет поведение (интерфейс) действий, так что процессор JSP может проверить, корректно ли страница использует библиотеку тегов. Общим правилом создания уникальных значений uri является использование строки, содержащей хост, с которого получена библиотека тегов. В результате значение uri напоминает URL, но это просто идентификатор, процессор JSP не будет обращаться к указанному хосту для извлечения файла дескриптора. Об интерпретации значений uri рассказано в разделе «Использование библиотеки тегов».

Атрибут prefix определяет, как будут вызываться теги из библиотеки. Приведенные ранее директивы указывали, что теги ядра и базы данных будут иметь вид <c:xxx> и <sql:xxx>. Например, можно использовать тег out библиотеки ядра для вывода значения так:

```
<c:out value="Hello, world." />
```

Можно запустить запрос при помощи тега базы данных query так:

```
<sql:query var="result" dataSource="${conn}">
    SHOW TABLES
</sql:query>
```

Элементы действий (action)

Теги элементов действий могут ссылаться на стандартные (предопределенные) действия JSP или на пользовательские (дополнительные) действия в библиотеке тегов. Названия тегов состоят из префикса и названия конкретного действия:

- Названия тегов с префиксом jsp обозначают предопределенные элементы действий. Например, <jsp: forward> пересылает текущий запрос на другую страницу. Такое действие доступно любой странице, запущенной в стандартном процессоре JSP.
- Пользовательские действия реализуются библиотеками тегов. Префикс названия тега должен соответствовать атрибуту prefix директивы taglib, уже указанной на странице, так что процессор JSP может определить, в какую библиотеку входит тег. Для того чтобы применять пользовательские теги, необходимо сначала установить библиотеку (см. «Использование библиотеки тегов»).

Действия записываются как элементы XML, и их синтаксис следует обычным правилам XML. Элемент с телом использует отдельные закрывающий и открывающий теги:

```
<c:if test="${x == 0}">
    x is zero
</c:if>
```

Если у тега нет тела, то можно объединить открывающий и закрывающий теги:

```
<jsp:forward page="some_other_page.jsp" />
```

Использование библиотеки тегов

Предположим, что у вас есть библиотека тегов, состоящая из JAR-файла *mytags.jar* и файла дескриптора библиотеки тегов *mytags.tld*. Для того чтобы библиотека стала доступной JSP-страницам определенного приложения, необходимо установить оба эти файла. Обычно JAR-файл помещают в каталог *WEB-INF/lib* приложения, а TLD-файл – в каталог *WEB-INF*.

Страница JSP, использующая библиотеку тегов, должна указать соответствующую директиву `taglib`, прежде чем использовать какое-либо из действий, предоставляемых библиотекой:

```
<%@ taglib uri="расположение-tld" prefix="идентификатор-taglib" %>
```

Атрибут `prefix` сообщает Tomcat о том, как вы будете ссылаться на теги библиотеки в оставшейся части страницы JSP. Если значение `prefix` указать как `mytags`, то далее в странице вы можете ссылаться на теги так:

```
<mytags:sometag attr1="attribute value 1" attr2="attribute value 2">
tag body
</mytags:sometag>
```

Значение `prefix` выбирается исключительно по вашему усмотрению, но далее вы должны единообразно использовать его на протяжении всей страницы, кроме того, нельзя одинаково называть две разные библиотеки тегов.

Атрибут `uri` указывает процессору JSP, как можно найти TLD-файл библиотеки тегов. Значение может быть прямым или косвенным:

- Вы можете указать значение `uri` непосредственно как путь к файлу TLD, который обычно помещается в каталог *WEB-INF*:

  ```
  <%@ taglib uri="/WEB-INF/mytags.tld" prefix="mytags" %>
  ```

  Начальный символ / указывает на то, что имя файла определяется относительно каталога приложения (контекстно-зависимый путь). Отсутствие начального символа / означает, что файл указан относительно местоположения страницы, содержащей директиву `taglib`.

  Если приложение использует множество библиотек тегов, то чтобы не засорять каталог *WEB-INF* файлами TLD, принято помещать их в подкаталог *tld* каталога *WEB-INF*. Тогда значение `uri` записывается так:

  ```
  <%@ taglib uri="/WEB-INF/tld/mytags.tld" prefix="mytags" %>
  ```

  Недостаток непосредственного указания пути к файлу TLD в том, что если выпускается новая версия библиотеки файлов, и файл TLD называет-

ся в ней по-другому, то придется изменять директиву `taglib` в каждой ссылающейся на файл странице JSP.

- Другим способом указания местоположения файла TLD является использование пути к JAR-файлу библиотеки тегов:

```
<%@ taglib uri="/WEB-INF/lib/mytags.jar" prefix="mytags" %>
```

Процессор JSP может таким образом найти TLD-файл, если его копия включена в JAR-файл как *META-INF/taglib.tld*. Однако проблема этого метода повторяет проблему предыдущего: если выходит новая версия библиотеки с другим именем JAR-файла, то приходится изменять директивы `taglib` для отдельных страниц JSP. Кроме того, данный способ не работает для контейнеров, которые не могут обнаружить TLD-файлы в JAR-файлах (так дело обстоит, например, со старыми версиями Tomcat).

- Есть и третий способ – будем указывать местоположение TLD-файла косвенно. Присвоим библиотеке символическое имя и добавим в файл приложения *web.xml* запись `<taglib>`, отображающую символическое имя в путевое имя TLD-файла. Затем на страницах JSP можно ссылаться на символическое имя. Предположим, что вы определяете такое символическое имя для библиотеки `mytags`:

http://terrific-tags.com/mytags

Запись `<taglib>` в файле *web.xml* должна указывать символическое имя и путь к соответствующему TLD-файлу. Если файл установлен в каталог *WEB-INF*, то запись может быть такой:

```
<taglib>
    <taglib-uri>http://terrific-tags.com/mytags</taglib-uri>
    <taglib-location>/WEB-INF/mytags.tld</taglib-location>
</taglib>
```

Если же файл хранится в *WEB-INF/tld*, изменяем запись:

```
<taglib>
    <taglib-uri>http://terrific-tags.com/mytags</taglib-uri>
    <taglib-location>/WEB-INF/tld/mytags.tld</taglib-location>
</taglib>
```

В любом случае вы ссылаетесь на библиотеку тегов в страницах JSP, используя символическое имя:

```
<%@ taglib uri="http://terrific-tags.com/mytags" prefix="mytags" %>
```

Использование символического TLD-имени приводит к возникновению некоторой неопределенности, но его значительное преимущество в том, что обеспечивается более стабильный способ ссылки на библиотеку тегов внутри JSP-страницы. Вы указываете фактическое местоположение файла TLD только в *web.xml*, а не в каждой отдельной странице JSP. Если выходит новая версия библиотеки тегов, где TLD-файл называется по-другому, то просто измените значение `<taglib-location>` в *web.xml* и перезапустите Tomcat, чтобы ваши страницы JSP могли работать с новой библиотекой. Нет необходимости изменять сами JSP-страницы.

Неявные объекты JSP

При запуске сервлета контейнер сервлета передает ему два аргумента, представляющие собой запрос и ответ, но вы можете объявить другие объекты самостоятельно. Например, можно использовать аргумент `response` для получения генерирующего вывод объекта так:

```
PrintWriter out = response.getWriter ();
```

Удобство JSP по сравнению с сервлетом заключается в наличии неявных объектов – то есть стандартных объектов, существующих как часть среды исполнения JSP. На любой из таких объектов можно ссылаться без предварительного декларирования. То есть в странице JSP объект `out` может считаться уже созданным и доступным для использования. Вот перечень некоторых наиболее полезных неявных объектов:

`pageContext`

Объект, создающий окружение страницы.

`request`

Объект, содержащий информацию о запросе, полученном от клиента, например, параметры, переданные в форме.

`response`

Ответ, формируемый для отправки клиенту. Его можно применять, например, для определения заголовков (header) ответа.

`out`

Объект вывода. Запись в этот объект при помощи методов `print()` или `println()` добавляет текст в страницу ответа.

`session`

Tomcat предоставляет доступ к сеансу, который может применяться для передачи информации от запроса к запросу. Появляется возможность писать приложения, которые взаимодействуют с пользователем, создавая для него видимость единой последовательности событий. Сеансы подробно описаны в главе 19.

`application`

Этот объект обеспечивает доступ к информации, совместно используемой на уровне приложения.

Области видимости в страницах JSP

Страницы JSP имеют доступ к различным уровням областей видимости (scope), которые могут использоваться для хранения информации разных уровней доступности. Существуют следующие уровни областей видимости:

Уровень страницы

Информация, доступная только текущей странице JSP.

Уровень запроса

Информация, доступная любой из JSP-страниц или сервлетов, обслуживающих текущий клиентский запрос. Одна страница может вызывать другую в процессе обработки запроса, и размещение инфомации в области видимости уровня запроса позволяет таким страницам взаимодействовать.

Уровень сеанса

Информация, доступная любой странице, обслуживающей запрос, входящий в определенный сеанс. Уровень сеанса может включать в себя несколько запросов одного клиента.

Уровень приложения

Информация, доступная любой странице, входящей в контекст приложения. Уровень приложения может распространяться на несколько запросов, сеансов или клиентов.

Один контекст ничего не знает о других, но страницы, отправляемые из одного контекста, могут разделять информацию друг с другом, регистрируя атрибуты (объекты) в одном из уровней выше уровня страницы.

Для перемещения информации на другой уровень используйте методы `setAttribute()` и `getAttribute()` неявного объекта, соответствующего данному уровню ((`pageContext`, `request`, `session` или `application`). Например, для того чтобы поместить строковое значение `tomcat.snake.net` в область видимости запроса в качестве атрибута `myhost`, используем объект `request`:

```
request.setAttribute ("myhost", "tomcat.snake.net");
```

`setAttribute()` хранит значение как `Object`. Для последующего извлечения значения выбираем его по имени при помощи метода `getAttribute()`, затем приводим обратно к строковому виду:

```
Object obj;
String host;
obj = request.getAttribute ("myhost");
host = obj.toString ();
```

При использовании с объектом `pageContext` методы `setAttribute()` и `getAttribute()` по умолчанию работают в контексте страницы. Иначе их можно вызывать с дополнительным параметром `PAGE_SCOPE`, `REQUEST_SCOPE`, `SESSION_SCOPE` или `APPLICATION_SCOPE` для явного определения области видимости. Следующие предложения получают тот же результат, что и только что рассмотренные:

```
pageContext.setAttribute ("myhost", "tomcat.snake.net",
                          pageContext.REQUEST_SCOPE);
obj = pageContext.getAttribute ("myhost", pageContext.REQUEST_SCOPE);
host = obj.toString ();
```

C

Справочная информация

В этом приложении собраны ссылки, которые могут пригодиться тем, кто хочет более подробно познакомиться с вопросами, затронутыми в книге.

Ресурсы MySQL

Майкл Вайдениус (Michael Widenius), Дэвид Эксмарк (David Axmark) и MySQL AB «MySQL Reference Manual» (Справочное руководство по MySQL), O'Reilly & Associates.

Поль Дюбуа (Paul DuBois) «MySQL», New Riders.[1]

Ответы на часто задаваемые вопросы по MySQL доступны по адресу *http://www.bitbybit.dk/mysqlfaq/*. Кроме того, на этом сайте поддерживается полезный указатель изменений и обновлений MySQL, который удобно использовать для определения того, доступна ли какая-то возможность в вашей версии.

Ресурсы Perl

Аллигатор Декарт (Alligator Descartes) и Тим Банс (Tim Bunce) «Programming the Perl DBI», O'Reilly & Associates.[2]

Линкольн Д. Стейн (Lincoln D. Stein) «Official Guide to Programming with CGI.pm» (Официальное руководство по программированию с помощью CGI.pm), Wiley Computer Publishing.

Поль Дюбуа (Paul DuBois) «MySQL and Perl for the Web», New Riders.[3]

Джеффри Янг (Geoffrey Young), Поль Линднер (Paul Lindner) и Рэнди Коубс (Randy Kobes) «mod_perl Developer's Cookbook» (Сборник рецептов для разработчика mod_perl), Sams Publishing.

---

[1] Поль Дюбуа «MySQL», 2-е издание, Вильямс, 2004.

[2] Аллигатор Декарт и Тим Банс «Программирование на Perl DBI», Символ-Плюс, 2000.

[3] Поль Дюбуа «Применение MySQL и Perl в Web-приложениях», Вильямс, 2002.

Дополнительные документы о DBI и CGI.pm можно получить из командной строки:

```
% perldoc DBI
% perldoc DBI::FAQ
% perldoc DBD::mysql
% perldoc CGI.pm
```

Или в Интернете:

http://dbi.perl.org/
http://stein.cshl.org/WWW/software/CGI/

Ресурсы PHP

Основной веб-сайт PHP расположен по адресу *http://www.php.net/*, там вы можете получить дистрибутивы и документацию PHP. Архив PEAR (PHP Extension and Add-on Repository) также имеет свой сайт – *http://pear.php.net/*. PEAR включает в себя модуль абстракции базы данных.

Ресурсы Python

Дистрибутивы и документация Python представлены на основном сайте Python по адресу *http://www.python.org/*.

Общая документация по интерфейсу доступа к базам данных DB-API доступна по адресу *http:// www.python.org/topics/database/*. Документация для MySQLdb, специальному драйверу DB-API для MySQL, хранится на сайте *http://sourceforge.net/projects/mysql-python/*.

Основным хранилищем исходных текстов кода Python служит сервер Vaults of Parnassus: *http:// www.vex.net/~x/parnassus/*.

Дэвид М. Бизли (David M. Beazley) «Python Essential Reference», New Riders.[1]

Ресурсы Java

Java-сайт компании Sun обеспечивает доступ к документации (включая спецификации) для JDBC, сервлетов, JSP и стандартной библиотеке тегов JSP:

- Общие сведения по JDBC: *http://java.sun.com/products/jdbc/index.html*
- Документация по JDBC: *http://java.sun.com/j2se/1.3/docs/guide/jdbc/index.html*
- Сервлеты Java: *http://java.sun.com/products/servlet/*
- Страницы JSP: *http://java.sun.com/products/jsp/*
- Библиотека JSTL: *http://java.sun.com/products/jsp/jstl/*

[1] Дэвид Бизли «Язык программирования Python. Справочник», Диасофт, 2000.

Джордж Риз (George Reese) «Database Programming with JDBC and Java» (Программирование для баз данных на Java и JDBC), O'Reilly & Associates.

Дэвид Флэнаган (David Flanagan) «Java Examples in a Nutshell», O'Reilly & Associates.[1]

Ханс Бергстен (Hans Bergsten) «JavaServer Pages», O'Reilly & Associates.

Дэвид Хармс (David Harms) «JSP, Servlets, and MySQL» (JSP, сервлеты и MySQL), M & T Books.

Саймон Браун и др. (Simon Brown, et al) «Professional JSP» (JSP для профессионалов), Wrox Press.

Шон Байерн (Shawn Bayern) «JSTL in Action» (JSTL в действии), Manning Publications.

Ресурсы Apache

Питер Уэйнрайт (Wainwright, Peter) «Professional Apache», Wrox Press.[2]

Другие ресурсы

Чак Муссиано (Chuck Musciano) и Билл Кеннеди (Bill Kennedy) «HTML & XHTML: The Definitive Guide», O'Reilly & Associates.[3]

Эрик Т. Рей (Erik T. Ray) «Learning XML», O'Reilly & Associates.[4]

Джеффри Э. Ф. Фридл (Jeffrey E. F. Friedl) «Mastering Regular Expressions», O'Reilly & Associates.[5]

---

1 Дэвид Флэнаган «Java в примерах. Справочник», 2-е издание, Символ-Плюс, 2003.

2 Питер Уэйнрайт «Apache для профессионалов», Лори, 2001.

3 Чак Муссиано и Билл Кеннеди «HTML и XHTML. Подробное руководство», Символ-Плюс, 2002.

4 Эрик Рей «Изучаем XML», Символ-Плюс, 2001.

5 Джеффри Фридл «Регулярные выражения», 2-е издание, Питер, 2003.

Алфавитный указатель

Специальные символы

A

B

C

D

E

F

G

H

I

M

N

O

P

T

U

V

W

X

Y

Z

А

Б

В

Г

Д

Е

Ж

З

И

К

Л

М

Н

О

П

P

C

T

У

Ф

Х

Ц

Ч

Э

Ю