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.

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.