I

Использование СУБД MySQL

В этой части...

1

Знакомство с СУБД MySQL и SQL

В этой главе...

Назначение СУБД MySQL
Пример базы данных
Основная терминология баз данных
Учебный курс по СУБД MySQL
Как работать с `mysql`
Что дальше?

Эта глава — введение в систему управления реляционными базами данных MySQL и язык структурированных запросов (SQL), который "понимает" СУБД MySQL. Здесь описаны основные термины и концепции, а также база данных, которая будет использована во всех примерах, приведенных в этой книге. В главе изложен обучающий материал, на примере которого можно научиться пользоваться СУБД MySQL.

Если вы — новичок в вопросах использования баз данных, если вы ничего не знаете о СУБД MySQL или языке SQL и вам нужно ознакомительное руководство, без колебаний начните чтение книги с этой главы. Читатели, которые уже имеют опыт работы с СУБД MySQL или с любой другой СУБД, могут пропустить этот материал. Однако всем будет полезно ознакомиться с разделом "Пример базы данных". Этот материал будет использоваться на протяжении всей книги.

Назначение СУБД MySQL

В этом разделе приведены примеры использования СУБД MySQL. Читатель получит представление о том, что может СУБД MySQL и как ее использовать. Если вам все это уже известно, можно смело перейти к разделу "Пример базы данных".

По сути *система баз данных* представляет собой способ манипулирования информационными списками. При этом информация может поступать из различных источников. Например, это могут быть данные, полученные в ходе исследований, деловые записи, заказы потребителей, спортивная статистка, отчеты о продажах, отчеты об ошибках или оценки учащихся. Несмотря на то что СУБД могут содержать широкий диапазон информации, вы не будете использовать СУБД только ради нее самой. Хорошим примером является список продуктов, из которого постепенно вычеркиваются купленные продукты, а потом этот список удаляется. Вряд ли вы воспользуетесь СУБД в этих целях. Даже в том случае, если у вас есть переносной ПК, вы воспользуетесь программой Блокнот, а не возможностями СУБД.

СУБД проявляет себя как мощный инструмент в том случае, когда информация, которую необходимо организовать и которой уже необходимо манипулировать, становится объемной и сложной, вследствие чего записи становятся непонятными и трудно обрабатываемыми вручную. Конечно, базы данных могут использоваться большими корпорациями, обрабатывающими миллионы транзакций в день. Но СУБД может также потребоваться для немасштабных операций, которые обрабатывает один человек в личных целях. Можно привести множество примеров ситуаций, когда СУБД будет использоваться даже в том случае, когда объемы информации не возросли до огромных размеров. Некоторые из таких ситуаций описаны ниже.

- В вашем столярном бизнесе заняты несколько работников. Нужно упорядочить записи о работниках и выплатах таким образом, чтобы иметь представление, кому и когда вы платили, получить возможность составлять отчеты о доходах в налоговые инстанции. Могут также понадобиться записи о нарядах, которые выполнял работник, и информация о том, кого планировали привлечь для выполнения этой работы.
- Вы содержите сеть складов запасных автомобильных частей. Необходимо получить оперативную информацию о том, на каком из складов лежит запасная часть, заказанная покупателем.
- Вы — продавец игрушек и зависите от колебания спроса на игрушки. Вам необходимо знать тенденции в продажах определенных товаров, чтобы иметь возможность оценить необходимость увеличения складских накоплений (для товаров, которые становятся более популярными) или уменьшения (нет нужды хранить на складе то, что уже не будет хорошо продаваться).
- Нужно проанализировать множество данных, полученных в ходе исследований за долгие годы, чтобы фраза "опубликуй или погибни" не стала эпитафией вашей карьеры. Вам необходимо "переварить" массу данных, чтобы получить итоговую информацию и более детальные выборки для подробного статистического анализа.

- Вы — известный докладчик, объезжающий страну с докладами в различных аудиториях: будь то выпускные церемонии, деловые встречи и т.д. Часто вспомнить, что вы говорили в той или иной аудитории, бывает довольно затруднительно. Поэтому потребовалось вести протоколы своих встреч, чтобы использовать их в своих планах на будущее. Это нужно для того, чтобы не повторяться в одной и той же аудитории. В этом случае запись о докладе, который вы прочитали в данной аудитории, поможет избежать повторений. Кроме того, можно оставить запись о том, насколько хорошо здесь был принят ваш доклад.
- Учитель ведет записи об успеваемости и посещаемости своих уроков. После каждого теста он записывает оценку, полученную каждым учащимся. Простым решением является запись оценок в журнале, но последующее использование этих записей затруднительно. При этом затруднительна сортировка учеников по баллам. Кроме того, учет посещаемости также становится большой проблемой.
- Вы работаете секретарем в организации и ведете список ее членов. (Организация может быть самой разной — профессиональное общество, клуб, театр, симфонический оркестр или атлетический клуб.) На основании постоянно корректируемого документа ежегодно издается список членов, который редактируется по мере изменения информации о них.

  Из-за ограничений, присущих этому методу, вы устали вести список вручную. Записи с трудом поддаются сортировке. Довольно проблематично выбрать только определенную часть каждой записи (например, перечень, состоящий только из фамилий и телефонов), или выбрать ту часть зарегистрированных членов организации, срок членства которых истек и которым следует немедленно возобновить свое членство.

  Кроме того, вы хотите избежать редактирования этого списка собственноручно, но бюджет организации не позволяет нанять кого-либо для выполнения этой работы. Вы что-то где-то слышали о “безбумажном офисе”, когда информация хранится в электронном виде, но не видели никакой выгоды от внедрения этой технологии для себя. Записи о членстве хранятся в электронном виде, но, по иронии, они хранятся в труднодоступной форме.

Такие сценарии применимы как для ситуаций с большими объемами информации, так и с малыми объемами. Всех их можно охарактеризовать как класс задач, которые в принципе можно выполнить вручную, хотя эффективнее они могут обрабатываться с помощью СУБД.

Каких преимуществ можно ожидать от использования такой СУБД, как СУБД MySQL? Это зависит от ваших конкретных задач и требований и, как видно из приведенных примеров, может сильно варьироваться. Давайте представим себе ситуацию, которая возникает достаточно часто, а потому очень показательна при использовании СУБД MySQL.

СУБД часто применяется для таких задач, для решения которых обычно используются картотеки. Действительно, базу данных можно представить в некотором роде в виде большой картотеки. Можно назвать несколько очень серьезных преимуществ ведения данных в электронном виде перед хранением информации вручную.

Преимущества применения СУБД MySQL описаны ниже.

- **Сокращение времени, необходимого для ведения записей.** В случае использования СУБД не требуется много времени на просмотр всей картотеки, чтобы добавить новую запись. Вы просто вводите ее в систему, не заботясь о месте размещения.
- **Сокращение времени, необходимого для поиска записей.** При поиске данных в СУБД нет необходимости последовательно просматривать все записи, чтобы найти интересующую. Предположим, что вы работаете в стоматологическом кабинете и вам надо разослать приглашения всем пациентам, которые забыли пройти ежегодный профилактический осмотр. Для этого достаточно сделать запрос к информационной системе. Конечно, это происходит не так, как при обычном общении с людьми, когда вы формулируете свой запрос примерно таким образом: "Найдите, пожалуйста, тех пациентов, которые не посещали нас на протяжении последних 6 месяцев". Работая с базой данных, вы вводите следующее:

  ```
  SELECT last_name, first_name, last_visit FROM patient WHERE
  last_visit < DATE_SUB(CURRENT_DATE, INTERVAL 6 MONTH);
  ```

  Возможно, вы никогда не встречали ничего подобного, но перспектива получить ответ через секунду или две вместо изнурительного многочасового просмотра записей должна показаться привлекательной. (В любом случае не стоит волноваться. Эта абракадабра скоро не будет казаться вам такой уж странной.)
- **Гибкость поиска.** Нет необходимости искать записи строго в соответствии с порядком, в котором они были записаны (по фамилии пациента, например). Информационной системе можно приказать расположить записи, отсортированные в любом порядке: по фамилии, названию страховой компании, дате последнего визита и т.д.
- **Гибкость формата вывода.** После того как необходимые записи найдены, копировать записи вручную не нужно. Можно сделать запрос информационной системе на вывод нужного списка. Иногда достаточно просто распечатать информацию. В других случаях вам может понадобиться воспользоваться этими данными в другой программе. (Например, после получения списка пациентов, которые забыли сделать ежегодный профилактический осмотр, эти данные можно переслать в текстовый редактор и уже на основании этой информации распечатать приглашения этим пациентам посетить ваш кабинет.) Предположим, что вам нужна только итоговая информация, такая как счетчик выбранных записей. И это совсем не обязательно делать вручную. Информационная система сгенерирует такой отчет сама.
- **Одновременный многопользовательский доступ к записям.** Предположим, что сразу два человека хотят просмотреть одну запись. При бумажном способе ведения дел второй кандидат всегда вынужден ждать, пока первый закончит просмотр бумаг. СУБД позволяет получить доступ к одной и той же записи одновременно.
- **Удаленный доступ и передача записей в электронном виде.** Бумажная технология ведения дел требует от вас быть там, где находятся сами бу-

маги, в противном случае кто-то должен скопировать и переслать их вам. Электронный способ ведения записей позволяет производить удаленный доступ к записям и передавать их в электронном виде. Предположим, что ваша стоматологическая фирма состоит из подразделений, удаленных территориально, электронный способ ведения учета позволяет получить доступ к вашим записям из удаленных офисов. Нет необходимости посылать копии записей о пациентах курьером, даже если кто-либо не имеет в своем распоряжении базы данных, аналогичной вашей, но имеет электронную почту. Ему можно будет послать содержимое базы данных в электронном виде.

Тот, кто имел опыт работы с СУБД, знает об этих преимуществах и может уже подумать не только о банальном замещении бумажной картотеки. СУБД теперь используются для предоставления услуг, о которых еще недавно не могло быть и речи. Примером тому может служить использование многими организациями баз данных в совокупности с доступом к Internet.

Предположим, что ваша компания имеет базу данных учета материальных ценностей, которой пользуются менеджеры при поступлении запросов от потребителей о наличии и стоимости товаров. Это достаточно традиционное использование баз данных, однако, если ваша компания может создать свой Web-узел, в перечне сервисов которого потребителям предоставляется возможность самим узнать о наличии товара на складе и его цене, это позволит потребителям самим получать нужную им информацию о товарах и способах их поставки. При этом потребитель получает информацию немедленно, не слушая раздражающую музыку в телефонной трубке. Такой магазин может работать круглые сутки. Для всех посетителей вашего Web-узла это будет означать, что им придется сделать на один звонок меньше.

Но базам данных можно найти еще лучшее применение. Запросы на поиск товара, сделанные через ваш Web-узел, могут служить источником информации не только для тех, кто их делает, но и для вас самих. Вы сможете получить ответ на очень важный вопрос: "Что нужно потребителю?" А результаты выполнения запросов потребителей покажут, сможете ли вы удовлетворить их запросы. Если у вас нет того, что им нужно в данный момент, ваш бизнес может прогореть. Поэтому есть смысл хранить информацию о спросе на товары и о имеющемся ассортименте. Имея под руками такую информацию, вы сможете эффективно управлять запасами товаров на складе и тем самым удовлетворять спрос покупателей.

Еще одним применением баз данных на Web-страницах может служить реклама на баннерах. У меня есть причины не любить их больше всех, но факт остается фактом. Они являются популярным применением СУБД MySQL, который может быть использован здесь для хранения рекламы и поиска ее Web-сервером для отображения. Кроме того, СУБД MySQL может хранить данные о том, как часто происходили обращения к данному роду деятельности, об отслеживании рекламы, которая сработала, сколько раз она запрашивалась, с каких узлов был произведен доступ, и т.д.

Итак, как же работает СУБД MySQL? Лучший способ узнать это — самостоятельно с ней поработать.

Пример базы данных

В этом разделе описывается тестовая база данных, которая будет использоваться в книге в дальнейшем. Приведены исходные тексты тестовых программ, которые помогут глубже понять принципы работы СУБД MySQL. Примеры созданы на базе двух сценариев.

- **Сценарий секретаря организации.** Нам нужен какой-то более определенный термин, чем "организация". Поэтому давайте воспользуемся такими характеристиками, как группа людей, объединенных интересом к истории Соединенных Штатов Америки (назовем ее за отсутствием лучшего названия "Исторической Лигой США"). Члены этого общества подтверждают свое членство, периодически выплачивая взносы. Взносы покрывают такие накладные расходы организации, как публикация журнала "Хроника прошлого США". Организация имеет свой небольшой, не очень хорошо разработанный Web-узел. Он содержит только информацию о том, что представляет собой "Историческая Лига США" (в дальнейшем — "Историческая Лига"), каков штат и как можно стать ее членом.
- **Сценарий учета успеваемости учащихся.** В процессе обучения вы ведете учет результатов опросов и тестов, записываете оценки и присваиваете степени. После этого определяются окончательные оценки, которые представляются руководству школы вместе с данными о посещаемости.

Теперь давайте оценим более детально эти сценарии, исходя из следующих критериев.

- Вам предстоит решить, для чего вам нужна база данных, т.е. какие цели вы ставите перед собой.
- Необходимо четко определить ту информацию, которой вы хотите наполнить базу данных, т.е. какие данные должны отслеживаться.

Возможно, такая последовательность критериев может показаться странной. Ведь очевидно, что до того как делать выборку информации из базы данных, ее нужно наполнить данными. Но способ использования вами базы данных зависит от конкретных задач. А задачи больше связаны с назначением базы данных, чем с тем, что вы внесли в нее. Наверняка вы не будете попусту тратить время и силы на ввод информации в базу данных, если не имеете намерений использовать эту информацию в дальнейшем.

"Историческая Лига"

Отправная точка этого сценария: вы как секретарь этой организации и ведете список членов в текстовом редакторе. Этого достаточно для распечатки списка, но совершенно недостаточно при любом другом использовании информации. У вас есть три цели, которые перечислены ниже.

- Вам необходимо получить возможность выводить информацию в разных форматах, получая при этом только необходимые данные. Одна цель — получить возможность распечатывать список каждый год — традиция, которой придер-

живается “Историческая Лига”. Можно придумать и другие применения информации, хранящейся в списке, например, получать список действительных членов “Исторической Лиги” для включения их имен в программу ежегодного банкета организации. Для выполнения этой задачи требуются различные наборы данных. В печатном списке используется все содержимое записи о члене “Исторической Лиги”. Для программы банкета достаточно только получить имена и фамилии членов организации (это не так просто в случае работы с текстовым редактором).

- Необходимо производить поиск в членском списке, руководствуясь самыми разнообразными критериями. Например, необходимо определить, кому из членов нужно вскоре обновить членство. Другая задача, также требующая поиска, связана со списком ключевых слов, которые вы определяете для каждого члена. Эти ключевые слова очерчивают те периоды истории США, которые входят в круг научных интересов данного члена (например, Гражданская война в США, Великая Депрессия, гражданские права или биография Томаса Джефферсона). Члены “Исторической Лиги” иногда интересуются тем, чем интересуются другие их коллеги, чтобы найти единомышленников.
- Вы хотите разместить список членов на Web-узле “Исторической Лиги”. Это будет полезно как для самих членов организации, так и для вас. Если вам удастся преобразовать список в Web-страницу с помощью автоматического процесса, интерактивный вариант списка всегда будет более актуальным, чем печатный. И если этот список еще будет снабжен функцией поиска, члены организации получат возможность просматривать информацию самостоятельно. Например, член “Исторической Лиги”, который хочет узнать, кто из других членов интересуется историей Гражданской войны в США, сможет сделать это, не ожидая, пока это сделает для него секретарь. Да и вам не потребуется тратить на это свое время.

Я отлично понимаю, что база данных не является верхом совершенства. Поэтому я далек от того, чтобы провозгласить, что базы данных стимулируют созидательное мышление. Тем не менее, когда вы прекратите думать об информации как о чем-то, требующем усилий (как это бывает при работе с обычными документами, созданными в текстовом редакторе), а будете думать о том, что можно достаточно просто манипулировать данными (как это должно быть в случае с СУБД MySQL), это поможет вам быстрее создавать новые подходы к использованию или отображению информации.

- Информация может быть выведена из базы данных на Web-узел “Исторической Лиги”. Это позволяет организовать ведение информации по-другому. Например, членам общества можно предоставить возможность самим редактировать информацию о себе в базе данных. Таким образом, полностью отпадает необходимость выполнять редактирование самостоятельно. Это позволит иметь в базе данных более актуальную информацию.
- В базе данных могут храниться адреса электронной почты членов организации. Ими можно воспользоваться для рассылки сообщений членам, которые не обновили свое членство. Эти сообщения могут содержать текущее со-

держимое их записей, просьбу обновить их, а также указания, как это сделать прямо на Web-узле.

- База данных может помочь вам сделать Web-узел более функциональным и полезным, даже без списка членов. "Историческая Лига" публикует журнал "Хроника прошлого США", в котором ведется детский раздел с викториной. Один из последних выпусков викторины был посвящен биографиям президентов США. Web-узел тоже может иметь детский раздел со своей викториной, работающий в режиме "on-line". Вероятно, что этот раздел может быть интерактивным, и информация для викторины будет заноситься прямо в базу данных. Таким образом, при генерации вопросов Web-сервер делает запрос базы данных случайным образом.

Отлично! С этого момента число пользователей базы данных позволит вам понять, что вы немного отклонились от предначертанной цели. После небольшого тайм-аута вернитесь с небес на землю и задайте себе пару вопросов, затрагивающих практические аспекты.

Не слишком ли много амбиций? Не слишком ли много требуется выполнить работы? Нет ничего проще, чем задумать что-либо и не сделать это. Ясовсем не хочу сказать, что все это просто реализовать. Тем не менее к концу этой книги вы уже сможете выполнить все то, что здесь было запланировано. Просто надо помнить, что ничего нельзя сделать сразу. Следует разделить работу на отдельные этапы. И постепенно, шаг за шагом, их осуществлять.

- **Способна ли СУБД MySQL делать все это?** Нет. Например, СУБД MySQL не обладает непосредственной способностью работать с Internet. Но если СУБД MySQL сама не обладает такими возможностями, для этого есть программное обеспечение, дополняющее и расширяющее ее возможности.

  Для написания сценариев доступа к базам данных СУБД MySQL мы воспользуемся языком написания сценариев Perl и модулем DBI (database interface — интерфейс базы данных). Язык Perl обладает отличными возможностями и позволяет выводить данные в различных форматах. Например, с помощью Perl список можно выдавать в формате Rich Text Format (RTF), который воспринимается любым текстовым редактором.

  Мы также можем воспользоваться языком написания сценариев PHP. Язык PHP полностью адаптирован к написанию Web-приложений и позволяет работать с базами данных. Это позволит генерировать запросы СУБД MySQL прямо с Web-страниц и генерировать новые страницы, содержащие результаты запросов к базе данных. Этот язык хорошо адаптирован для работы с Web-сервером Apache (самый популярный Web-сервер), упрощающим выборку данных и отображение результатов выборки.

  СУБД MySQL хорошо интегрируется с этим программным обеспечением и позволяет гибко комбинировать его по своему усмотрению в соответствии с поставленными задачами. Это интегрированный компонент, обладающий высокой степенью интеграции "все в одном".

- **И наконец, самый больной вопрос — сколько все это стоит?** Бюджет организации довольно ограничен. Но как ни удивительно, это не будет вам ничего сто-

ить. Если вы знакомы с основными базами данных, имеющимися на рынке, то должны знать, что они достаточно дороги. В отличие от них, СУБД MySQL обычно распространяется бесплатно. (Для ознакомления с условиями лицензирования обратитесь к руководству по СУБД MySQL.) Бесплатно распространяется и другое программное обеспечение (Perl, DBI, PHP, Apache).

Выбор операционной системы ложится целиком и полностью на вас. Все программное обеспечение работает под управлением ОС UNIX (это название я использую как общее для ОС BSD UNIX, Linux, Mac OS X и т.д.) и Windows. Для ОС UNIX и Windows существуют исключения, такие как консольные и пакетные сценарии.

Теперь давайте рассмотрим другой сценарий использования баз данных.

Проект "Учет успеваемости"

Отправным пунктом этого сценария является учитель, на которого возложены обязанности учета успеваемости учащихся. У вас существует намерение заменить ручной учет, который проводился с помощью журнала, на электронное ведение дел с помощью СУБД MySQL. В этом случае вы хотите извлечь из базы данных то же, что и из вашего журнала успеваемости.

- По результатам викторин и тестов записываются баллы, полученные учащимися. Баллы тестов имеют такое обозначение, чтобы с первого взгляда на них можно было определить оценку (A, B, C, D и F).
- В конце каждого учебного периода подсчитывается суммарный балл каждого учащегося. Затем они сортируются, и на их основании выставляются оценки. Суммы могут содержать взвешенные коэффициенты, так как вы, вероятно, захотите акцентировать внимание на результатах тестов, а не на результатах викторин.
- По завершении учебного периода вы представляете оценку посещаемости школьному руководству.

Целью этого проекта является попытка избежать сортировки и суммирования оценок успеваемости и посещаемости вручную. Другими словами, СУБД MySQL вам нужна для сортировки оценок и вычислений, необходимых для получения суммарных баллов учащихся и числа пропусков занятий по завершении учебного периода. Для этого необходимы список учащихся, баллы, полученные за каждую викторину или тест, даты пропуска занятий учащимися.

Каким образом пример базы данных можно использовать в конкретном случае

Вас не интересует ни пример с "Исторической Лигой", ни пример учета успеваемости/посещаемости? Ответом может быть то, что эти примеры не являются самоцелью. Это только средство, с помощью которого демонстрируются возможности СУБД MySQL и соответствующего программного обеспечения.

Теперь рассмотрим, как примеры запросов к базе данных применимы к конкретным задачам, которые вы собираетесь решать. Предположим, что вы рабо-

таете в стоматологическом кабинете. Автор уже ссылался на этот пример. В этой книге вы не найдете никаких примеров, связанных со стоматологией, но вы встретите много запросов, которые смогут вас заинтересовать. Например, поиск членов "Исторической Лиги", которые должны возобновить свое членство, аналогичен определению пациентов, пропустивших свой очередной профилактический осмотр. И, действительно, оба запроса базируются на датах. Так, если вы знаете, как написать запрос на обновление членства, вы легко сможете создать запрос о недисциплинированных пациентах.

Основная терминология баз данных

Вы, наверное, почувствовали, что, пролистав достаточно много страниц этого фолианта, все еще не окунулись в жаргон и техническую терминологию. Я не сказал ни слова о том, как в действительности выглядит "база данных". Коротко были определены цели применения базы данных. Однако это возможно только на этапе проектирования базы данных. Этому вопросу и посвящен данный раздел. Здесь описаны термины, встречающиеся в этой книге. К счастью, концепция реляционных баз данных достаточно проста. И действительно, многие преимущества реляционных баз данных заложены в самой простоте базовых концепций.

Структурная терминология

СУБД MySQL классифицируется как реляционная система управления базами данных (RDBMS — relational database management system). Эта аббревиатура разбивается на части следующим образом.

- База данных (БД) (DB в RDBMS) — это совокупность информации, разбитой простым, регулярным образом.
  - Совокупность данных в базе данных объединена в таблицы.
  - Таблица состоит из строк и столбцов.
  - Строка таблицы является ее записью.
  - Записи содержат несколько единиц информации, каждый столбец таблицы содержит одну такую единицу.
- Система управления (СУ) (MS — management system) является программным обеспечением, позволяющим вставлять, выбирать, модифицировать и удалять записи.
- Слово "реляционная" обозначает популярную разновидность СУБД, в которых отслеживается "соответствие" записей в одной таблице на "соответствие" записей в другой таблице. Мощь реляционных СУБД заключается в их способности выбирать соответствующие данные из этих таблиц и создавать ответы на вопросы, которые нельзя получить только из одной такой таблицы.

Приведем пример того, как реляционная база данных объединяет данные в таблицы и связывает данные из одной таблицы с данными из другой таблицы. Предположим, что вы сопровождаете Web-узел, представляющий сервис размещения баннеров. Вы поддерживаете контакт с компаниями, которые хотят

разместить свою рекламу. Эта реклама будет появляться при каждом посещении вашего Web-узла. Всякий раз, когда посетитель вашего узла попадает на одну из ваших страниц, вы вместе с ней посылаете встроенную рекламу на броузер посетителя и получаете от рекламируемой компании небольшой гонорар. Для отображения этой информации воспользуемся тремя таблицами (рис. 1.1). Первая таблица, `company`, содержит столбцы с именами компаний, номером, адресом и телефонным номером. Вторая таблица, `ad`, содержит номера реклам, компаний, которым принадлежат эти рекламы, сумму, которую вы берете с них за один доступ к их рекламе. Третья таблица, `hits`, регистрирует количество обращений к рекламе и дату, когда это произошло.

Ответы на некоторые вопросы можно получить, пользуясь информацией из одной таблицы. Для определения количества компаний, с которыми у вас имеются контракты, достаточно посчитать строки в таблице `company`. Аналогично, для подсчета количества обращений за определенный период времени достаточно таблицы `hit`. Другие вопросы будут посложнее, и для получения ответов на них необходимо опросить не одну, а сразу несколько таблиц. Например, для определения, сколько раз 14 июля было сделано обращение ко всем рекламам компании Pickles, Inc., вам потребуется просмотреть три таблицы.

1. По названию компании (Pickles, Inc.) в таблице `company` определить номер компании (14).
2. По номеру компании найти соответствующие ему записи в таблице `ad`, содержащие номера рекламы. В нашем примере мы видим две такие записи — 48 и 101.
3. С помощью номера рекламы для всех найденных в таблице `ad` записей определить соответствующие записи в таблице `hit`, попадающие в нужный диапазон дат, а затем подсчитать количество совпадений. Найдены три совпадения для рекламы под номером 48 и два совпадения для рекламы под номером 101.

На первый взгляд, все это непросто. Но это лишь основа, на которой зиждется совершенство реляционных баз данных. Здесь сложность несколько иллюзорна, ибо каждый только что описанный шаг содержит нечто большее, чем простые операции выборки: вы связываете одну таблицу с другой с подбором значений из строк одной таблицы со строками другой таблицы. Эта простая операция может быть использована различными способами для получения ответов на следующие вопросы. Сколько различных реклам имеет каждая компания? Рекламы какой компании наиболее популярны? Сколько дохода приносит каждая реклама? Каков полный гонорар, полученный от каждой компании за текущий период?

Теперь вы достаточно знаете теорию реляционных баз данных, чтобы понять оставшийся материал, изложенный в этой книге. Здесь мы не будем углубляться в изучение третьей нормализованной формы, диаграмм “сущность–связь” и других высоконаучных проблем. Если вы хотите вникнуть в эти вопросы, что само по себе уже страшно, я могу предложить вам обратиться к трудам таких столпов науки о базах данных, как Дейт (*C.J. Date*) или Кодд (*E.F. Codd.*)

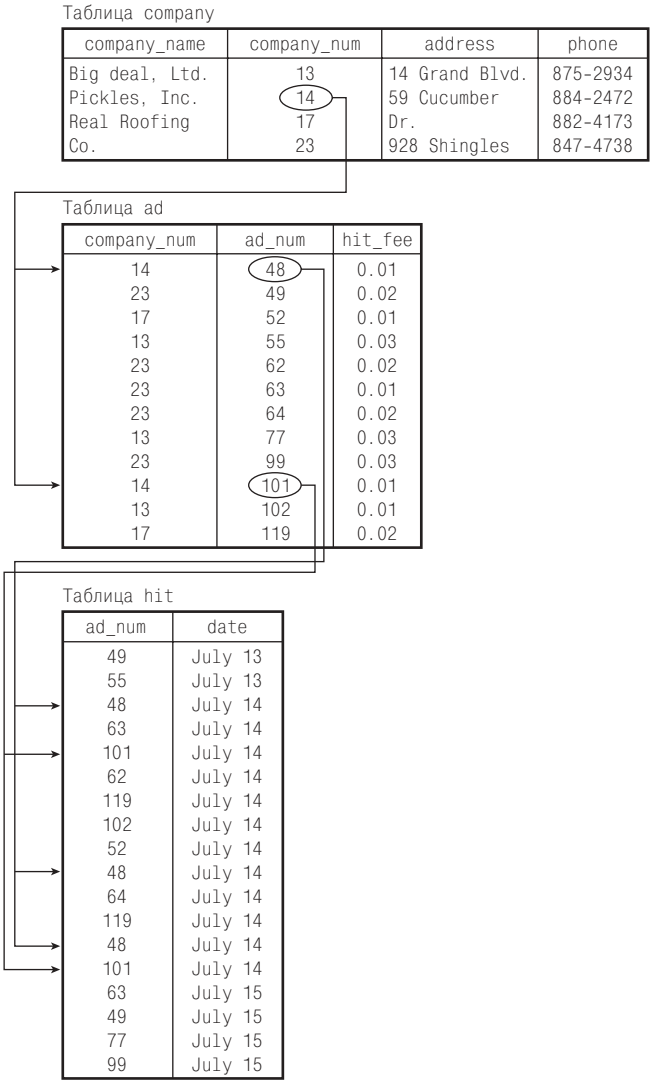

Рис. 1.1. Таблицы учета баннерной рекламы

Терминология языка запросов SQL

Для общения с СУБД MySQL применяется язык SQL (Stuctured Query Language – язык структурированных запросов). В настоящее время SQL является стандартом работы с базами данных, и все основные СУБД понимают его. SQL включает много разных типов операторов, разработанных для взаимодействия с базами данных.

Как и в случае с другими языками, SQL при первом знакомстве может показаться несколько странным. Например, создавая таблицу, необходимо указать СУБД MySQL, какую структуру она должна иметь. Вы и я можем представить себе

таблицу в виде диаграммы или рисунка, но СУБД этого не может. Таким образом, создать таблицу можно, только сказав СУБД MySQL что-то вроде следующего:

```
CREATE TABLE company
(
   company_name CHAR(30),
   company_num INT,
   address CHAR(30),
   phone CHAR(12)
);
```

Новичку такой оператор может показаться странным, но совсем не надо быть программистом, чтобы научиться эффективно применять его с SQL. Познакомившись с этим языком ближе, вы будете смотреть на оператор `CREATE TABLE` в другом свете: как на союзника, помогающего описывать информацию, а не как на колдовскую головоломку, состоящую из битов с байтами.

Терминология архитектуры СУБД MySQL

СУБД MySQL использует традиционную архитектуру клиент/сервер, поэтому, работая с СУБД MySQL, пользователь реально работает с двумя программами.

- Программа сервера базы данных, расположенная на компьютере, где хранится база данных. Она "прослушивает" запросы клиентов, поступающие по сети, и осуществляет доступ к содержимому базы данных для предоставления информации, которую запрашивают клиенты.
- Клиентская программа, которая осуществляет подключение к серверу и передает запросы на сервер.

Дистрибутив СУБД MySQL включает в себя несколько клиентских программ и сервер. Клиентские программы используются в соответствии с поставленными задачами. Одной из наиболее популярных и наиболее привлекательных клиентских программ является `mysql`. Это интерактивная программа, позволяющая создавать запросы и просматривать полученные результаты. Двумя административными программами являются `mysqldump`, которая позволяет делать дамп базы данных в файл и восстанавливать его обратно, и `mysqladmin`, позволяющая проверять состояние сервера и выполнять административные задачи наподобие отключения сервера. Кроме того, дистрибутив содержит другие клиентские программы.

Клиентские программы могут не подходить по своим функциональным возможностям для ваших целей, тогда вам может пригодиться библиотека для написания своей собственной программы-клиента СУБД MySQL. Эту библиотеку можно вызывать прямо из программ, написанных на языке C. В случае, если вы предпочитаете язык, отличающийся от C, доступно несколько других языков — Perl, PHP, Python, Java, C++.

Архитектура клиент/сервер СУБД MySQL имеет ряд преимуществ.

- Сервер обеспечивает контроль параллельного доступа, что не позволяет двум пользователям одновременно модифицировать одну и ту же запись. Все клиентские запросы проходят через сервер; таким образом, сервер знает, кто производит доступ к базе, что и когда он хочет сделать. При

одновременном доступе нескольких пользователей к одной и той же таблице базы данных им нет необходимости предварительно искать друг друга и договариваться между собой. Они просто делают свои запросы, а в остальном полностью полагаются на сервер.

- Нет необходимости регистрироваться на компьютере, где расположена база данных. СУБД MySQL может работать через Internet. Поэтому клиентская программа может быть запущена везде, где угодно, а клиент сам производит соединение с сервером через сеть. Расстояние не имеет значения, и к серверу можно подключиться из любой точки мира. Даже если сервер находится в Австралии, к нему можно подключиться с помощью переносного компьютера из Исландии. Значит ли это, что *любой человек* может подключиться к вашей базе данных через Internet? Конечно нет. СУБД MySQL снабжена гибкой системой защиты, которая позволяет иметь доступ к базе данных только тем, кто имеет на это право. Кроме того, этим пользователям должно быть разрешено делать только то, что они имеют право делать. Очевидно, что Салли из биллингового управления должна иметь право на чтение или модификацию записей, а Фил из отдела обслуживания — только на их просмотр. СУБД MySQL позволяет соответствующим образом определять их права.

В MySQL версии 4 есть другие варианты работы сервера. Вдобавок к обычному серверу `mysqld`, который используется в среде клиент/сервер, MySQL содержит сервер в виде библиотеки `libmysqld`, ссылки на которую вы можете поместить в программы, чтобы создать самостоятельные программы, работающие с MySQL. Это называется библиотекой встроенного сервера, поскольку он встраивается в автономные программы. Использование встроенного сервера отличается от варианта клиент/сервер, так как не требуется сеть. Таким образом, легче создать программы, которые могут распространяться сами по себе, с некоторыми поправками на их внешнюю рабочую среду. С другой стороны, этот метод должен быть использован только в ситуациях, когда такой программе единственной потребуется доступ к базам данных, управляемых сервером.

Различие между MySQL и `mysql`

Во избежание разночтений, хочу подчеркнуть, что MySQL относится ко всей СУБД MySQL, а "`mysql`" — это название конкретной клиентской программы. В разговорной речи это звучит одинаково, но в написании одно наименование пишется прописными буквами, а другое — строчными.

С одной стороны, MySQL произносится как “май–эс–кью–элл” (это известно из руководства по использованию MySQL). С другой стороны, SQL произносится как “сиквел”. Это зависит от того, кого вы спрашиваете. Я не настаиваю ни на одном из вариантов. Произносите это слово как вам угодно, но будьте готовы к тому, что кто-нибудь вас поправит.

Учебный курс по СУБД MySQL

Теперь вы обладаете базовыми знаниями, которые необходимы вам на этом этапе. Приступим к работе с СУБД MySQL!

В этом разделе вы познакомитесь с СУБД MySQL. Для этого здесь приведен учебный курс по СУБД MySQL. По мере ознакомления с учебным курсом вы соз-

дадите базу данных с таблицами, а потом попробуете поработать с ней, добавляя, удаляя и модифицируя информацию, а также осуществляя ее поиск. Кроме того, в процессе работы с учебной базой данных вы узнаете следующее.

- **Основы диалекта языка SQL, который применяется в работе с СУБД MySQL**. (Для тех, кто уже знаком с языком SQL по опыту работы с другой СУБД, также будет полезно просмотреть этот учебный материал, чтобы получить понятие о различиях, имеющихся в данной версии языка).

- **Основные принципы работы с сервером СУБД MySQL с помощью клиентской программы `mysql`.** Эта программа получает SQL-запросы от пользователя, посылает их на сервер для выполнения и возвращает полученный результат пользователю. Клиент `mysql` работает на любой платформе, имеющей поддержку СУБД MySQL, и предоставляет наиболее прямые средства взаимодействия с сервером. Поэтому логичнее начать наш учебный курс с сервера. В некоторых примерах используются клиенты `mysqlimport` и `mysqlshow`.

Назовем учебную базу данных `sampdb`. Однако может потребоваться воспользоваться другим именем для обозначения базы данных. Вероятно, что кто-то уже использует имя `sampdb` для обозначения своей базы данных, поэтому ваш администратор предоставит вам другое имя. В любом случае вы можете свободно заменить имя базы данных `sampdb` другим именем.

Таблицам рекомендуем давать имена, аналогичные тем, которые приведены в примерах. Это допустимо, даже если аналогичную учебную базу данных уже создало несколько пользователей вашей системы. В СУБД MySQL не имеет никакого значения то, что несколько пользователей применяют одноименные таблицы. Таблицы хранятся в различных базах данных, и СУБД MySQL не позволяет им пересекаться.

Пример тестовой базы данных

Этот обучающий материал в определенной мере имеет отношение к файлам из примера тестовой базы данных (также известным как дистрибутив sampdb). Это файлы, содержащие запросы и данные, которые могут оказаться полезными при установке учебной базы данных. О том, как можно получить тестовую базу данных, читатель узнает в приложении А, “Получение и инсталляция программного обеспечения”. После разархивирования примера будет автоматически создан каталог `sampdb`, содержащий все необходимые файлы. Я рекомендую сделать этот каталог текущим.

Предварительные требования

Для того чтобы выполнить примеры, приведенные в этой книге, должно быть удовлетворено несколько предварительных условий.

- У вас должно быть установлено программное обеспечение MySQL.
- Для того чтобы вы могли подключиться к серверу в СУБД MySQL, на вас должна быть заведена учетная запись.

- Наконец, вам понадобится база данных.

Требуемое программное обеспечение содержит клиента и сервер MySQL. Клиентские программы должны размещаться на компьютере, на котором вы будете работать. Сервер может размещаться на вашей машине, хотя это не обязательно. Пока у вас есть доступ к нему, вы можете подключаться откуда угодно. Для получения MySQL обратитесь к приложению А, "Получение и инсталляция программного обеспечения". Если у вас доступ к сети через поставщика Internet-услуг (ISP), узнайте, предоставляет ли он MySQL-хостинг. В противном случае обратитесь к приложению И, "Провайдеры услуг Internet", чтобы узнать, как выбрать подходящего поставщика.

После этого необходимо получить право на создание своей тестовой базы данных и ее таблиц. (Если у вас уже есть учетная запись MySQL, вы можете использовать ее, хотя, возможно, вы захотите создать отдельную запись для работы с материалами, представленными в этой книге.)

Чтобы настроить учетную запись MySQL, с помощью которой вы будете подключаться к серверу, необходимо подключиться к серверу. Обычно это делается привилегированным пользователем на сервере, на котором установлен сервер MySQL, и вам предоставляются права на создание новой учетной записи. Если вы установили MySQL на собственном компьютере и сервер работает, вы можете подключиться к нему и создать новую учетную запись администратора с именем `sampadm` и паролем `secret` (по всей книге можно изменить имя и пароль на нужные).

```
% mysql -p -u root
Enter password: ******
mysql> GRANT ALL ON sampdb.* TO 'sampadm'@'localhost' IDENTIFIED BY
'secret';
```

Команда `mysql` содержит ключ `-p` для запроса пароля привилегированного пользователя MySQL. Введите пароль там, где в примере вы видите ******. (Я полагаю, что вы уже указали пароль привилегированного пользователя и знаете его. Если же вы еще не указали его, нажмите клавишу <Enter> при запросе "Enter password:". Однако отсутствие пароля небезопасно, и вы должны указать его как можно скорей.)

Оператор `GRANT` действителен только при входе с машины, на которой работает сервер. Он позволяет подключиться к серверу с именем `sampadm` и паролем `secret` и предоставляет полный доступ к базе данных `sampdb`. Но он не создает базу данных. К этому мы вернемся позже.

Если вы не планируете подключаться с компьютера, на котором работает сервер, измените `localhost` на имя машины, с которой вы будете работать. Например, если вы будете подключаться к серверу с адреса `asp.snake.net`, утверждение `GRANT` должно выглядеть следующим образом:

```
mysql> GRANT ALL ON sampdb.* TO 'sampadm'@'asp.snake.net' IDENTIFIED
BY 'secret';
```

Если у вас нет контроля над сервером, попросите администратора MySQL создать для вас учетную запись. Затем измените имя, пароль и имя базы данных, которые назначил администратор, на `sampadm`, `secret` и `sampdb`. Дополнительную информацию об утверждении `GRANT`, настройке учетных записей MySQL и изменении паролей можно найти в главе 11, "Общее администрирование MySQL".

Установка и завершение связи с сервером

Для подключения к серверу вызовите программу `mysql` из своей оболочки (т.е. из подсказки ОС UNIX или из DOS-консоли под управлением ОС Windows). Вот эта команда:

```
% mysql options
```

Для отображения подсказки операционной системы воспользуемся обозначением `'%'`. Это одна из стандартных подсказок ОС UNIX, другой такой подсказкой является подсказка `'$'`. В ОС Windows подсказка будет иметь вид `'C:\>'`.

Часть параметров командной строки `mysql` может быть пустой, но, вероятнее всего, эта строка будет иметь вид

```
% mysql -h host_name -p -u user_name
```

Вряд ли вам потребуется указывать все ключи при запуске клиентской программы `mysql`, но, возможно, вам придется изменить хотя бы имя и пароль.

- `-h` *`host_name`* (альтернативная форма: `- -host=`*`host_name`*).

  Указывает имя серверного узла, к которому вы хотите подключиться. Если сервер работает на том же компьютере, что и клиент, этот параметр можно смело опустить.

- `-u` *`user_name`* (альтернативная форма: `- -user=` *`user_name`*).

  Указывает имя пользователя, зарегистрированное в СУБД MySQL. Если имена пользователей ОС UNIX и СУБД MySQL совпадают, этот ключ тоже можно опустить. При этом клиентская программа `mysql` будет использовать ваше регистрационное имя в качестве имени пользователя СУБД MySQL.

  При работе в Windows, имя пользователя по умолчанию будет ODBC. Возможно, это вам не очень пригодится. Существует возможность указать свое имя в командной строке или установить свою среду по умолчанию, указав имя пользователя `sampadm` в команде `set`:

  ```
  C:\> set USER=sampadm
  ```

  Если вы пропишете эту команду в файл AUTOEXEC.BAT, она будет срабатывать при каждой загрузке и вам не придется задавать ее в запросе.

- `–p` (альтернативная форма: `- -password`).

  Этот параметр подсказывает СУБД MySQL, что нужно запросить ваш пароль. Например:

  ```
  % mysql –h host_name –u user_name –p
  Enter password:
  ```

  Когда пользователь получает подсказку `Enter password:`, он должен вводить пароль. (Пароль традиционно не выводится на экран на тот случай, если кто-то заглядывает через ваше плечо.) Отметим, что ваш пароль в СУБД MySQL не обязательно совпадает с паролем в ОС UNIX или ОС Windows.

  При полном отсутствии ключа `–p` `mysql` решает, что пароль вам не нужен и не выводит подсказки.

  Также можно указать пароль прямо в командной строке как `-pyour_pass` (альтернативная форма: `-password=your_pass`). Но из соображений

безопасности лучше этого не делать, хотя бы потому, что пароль будет виден окружающим. Если вы все же решили указать пароль в командной строке, учтите, что между ключом `-p` и паролем нет пробела. Это часто является причиной путаницы, так как ключ `-p` отличается от ключей `-h` и `-u`, после которых всегда стоит пробел.

Например, предположим, что имя пользователя и пароль в СУБД MySQL соответственно `sampadmin` и `secret` и что пользователь хочет подключиться к серверу, запущенному на том же компьютере, на котором он работает. Для этого необходимо задать следующую команду `mysql`:

```
% mysql -p -u sampadm
Enter password: ******
```

После ввода команды `mysql` ответит подсказкой `Enter password:`. Пользователь вводит пароль, который отображается по мере ввода `******` (здесь был введен пароль `secret`).

Если пароль правильный, `mysql` ответит подсказкой `mysql>`. Это свидетельствует о том, что СУБД MySQL ожидает ввода новых запросов. В целом последовательность команд для подключения к серверу СУБД выглядит следующим образом:

```
% mysql -p -u sampadm
Enter password: ******
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 7575 to server version: 4.0.4-log
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql>
```

Для подключения к серверу, работающему на каком-либо другом компьютере, дополнительно требуется с помощью ключа `–h` указать имя узла. Предположим, что узел имеет имя `cobra.snake.net`, тогда команда будет иметь вид

```
% mysql -h cobra.snake.net -p -u sampadm
```

В большинстве из приведенных здесь примеров демонстрируется применение ключей `–h`, `–u` и `–p`. Это делается с одной целью – научить вас правильно использовать эти ключи.

Установленное соединение с сервером можно прервать в любой момент командой `QUIT`:

```
mysql> QUIT
Bye
```

Кроме того, завершить сеанс можно с помощью комбинации клавиш <Ctrl+D> (по крайней мере в ОС UNIX).

Первоначально новичок может подумать, что система безопасности СУБД MySQL будет только раздражать и затруднять работу с СУБД. (Для создания базы данных и получения доступа к ней необходимо получить разрешение и при каждом подключении к серверу вводить свое имя и пароль.) Однако после того, как вы начнете создавать и использовать свои собственные записи, ситуация кардинально изменится. Тогда вы уже сможете оценить то, что СУБД MySQL не позволяет другим пользователям просматривать (или, что еще хуже, уничтожать!) ваши данные.

Существуют способы настроить учетные записи таким образом, что вам не придется каждый раз вводить параметры соединения. Они обсуждаются в раз-

деле “Как работать с mysql” в этой главе. Самый простой способ упростить процесс соединения — сохранить настройки в отдельном конфигурационном файле. Вы можете перейти к указанному выше разделу, чтобы узнать, как создать такой файл.

Ввод запросов

Запросы к базе данных можно вводить только после подключения к серверу. В этом разделе рассказывается о работе с клиентской программой `mysql`.

Для ввода запроса в `mysql` достаточно его напечатать, ввести в конце точку с запятой и нажать клавишу <Enter>. После ввода запроса `mysql` передает его серверу на выполнение. После обработки запроса сервер передает результат клиенту `mysql`. Клиент отображает полученный результат.

Вот пример простейшего запроса текущих даты и времени:

```
mysql> SELECT NOW();
+---------------------+
| NOW()               |
+---------------------+
| 2002-09-01 13:54:24 |
+---------------------+
1 row in set (0.00 sec)
```

`mysql` отображает результат запроса и строку, которая показывает, из скольких строк состоит результат и сколько времени затрачено на выполнение запроса. В следующих примерах я не буду показывать эту строку. Программа `mysql` воспринимает точку с запятой как окончание запроса, поэтому по мере необходимости запросы могут занимать несколько строк:

```
mysql> SELECT NOW(),
    -> USER(),
    -> VERSION()
    -> ;
+---------------------+-------------------+----------------+
| NOW()               | USER()            |      VERSION() |
+---------------------+-------------------+----------------+
| 2002-09-01 13:54:37 | sampadm@localhost | 4.0.4-beta-log |
+---------------------+-------------------+----------------+
```

Обратите внимание, что подсказка меняется с `mysql>` на `->` после ввода первой строки запроса. Это означает, что `mysql` ожидает продолжения ввода запроса. Это очень важный элемент обратной связи, показывающий пользователю, что даже если вы забыли ввести точку с запятой, `mysql` ожидает ее ввода и показывает, что ввод запроса еще не завершен. Другими словами, пока пользователь сидит и удивляется, почему так долго нет ответа от `mysql`, клиент `mysql`, в свою очередь, ожидает завершения ввода запроса! (В `mysql` также существуют другие запросы. Они описаны в приложении Д, “Программы MySQL”.)

В большинстве случаев регистр, в котором вводится команда, не имеет особого значения. Запросы, приведенные ниже, эквивалентны.

```
SELECT USER();
select user();
SeLeCt UsEr();
```

Примеры, приведенные в этой книге, выполнены в соответствии со следующим правилом: прописными буквами вводятся все ключевые слова и функции языка SQL; для имен баз данных, таблиц и столбцов используются строчные буквы.

При вызове функции в запросе недопустим пробел между именем функции и последующими скобками.

```
mysql> SELECT NOW ();
ERROR 1064: You have an error in your SQL syntax near '()' at line 1
mysql> SELECT NOW();
+---------------------+
| NOW()               |
+---------------------+
| 2002-09-01 13:56:36 |
+---------------------+
```

Эти два запроса выглядят аналогично, но во втором сообщается об ошибке, поскольку за именем функции нет скобок.

Отменить запрос можно также, напечатав "\g":

```
mysql> SELECT NOW()\g
+---------------------+
| NOW()               |
+---------------------+
| 2002-09-01 13:56:47 |
+---------------------+
```

Вы также можете использовать параметр `'\G'`, который отображает результат в вертикальном формате:

```
mysql> SELECT NOW(), USER(), VERSION()\G
*************************** 1. row ***************************
    NOW(): 2002-09-01 13:56:58
   USER(): sampadm@localhost
VERSION(): 4.0.4-beta-log
```

Для запроса, который генерирует короткие строки, параметр `'\G'` практически бесполезен, но если строки настолько длинны, что засоряют экран, `'\G'` делает вывод более читабельным.

Если вы начали вводить запрос, а затем передумали, введите `'\c'` для его отмены.

```
mysql> SELECT NOW(),
    -> VERSION(),
    -> \c
mysql>
```

Обратите внимание, что подсказка опять изменилась на `mysql>`. Это свидетельствует о том, что `mysql` ожидает ввода нового запроса.

Запросы можно сохранять в файле и затем запускать их на выполнение прямо из файла, не создавая вновь. Для этого можно воспользоваться функциями переназначения ввода-вывода. Например, если запрос сохранен в файле `my_file.sql`, его можно выполнить с помощью команды.

```
% mysql < my_file.sql
```

Таким образом можно вызвать любой файл. Для того чтобы показать, что файл содержит операторы SQL, был указан суффикс `.sql`.

Работа `mysql` более детально будет описана в разделе "Дополнение таблиц". Таким образом, будут добавлены данные в базу данных `sampdb`. Действительно, намного удобнее заполнить таблицу с помощью операторов `INSERT`, считываемых из файла, чем вводить каждый оператор по отдельности.

В оставшейся части этого раздела будут продемонстрированы запросы, работу которых вы сможете повторить затем сами. Их можно найти по подсказке `'mysql>'` перед запросом и завершающей точке с запятой. Как правило, эти примеры сопровождаются полученным результатом запроса. Вы сможете ввести запросы сами и получить аналогичный ответ. Запросы, показанные без подсказки и завершающей точки с запятой, несут смысловую нагрузку просто как разъяснение материала, и их выполнять нет смысла. (При желании их можно запустить на выполнение, не забывая при этом завершать оператор точкой с запятой.)

> **Когда требуется точка с запятой**
>
> В этой книге большинство запросов заканчивается точкой с запятой, что удобно для обозначения завершения каждого запроса (особенно в примерах, сразу содержащих несколько операторов). Она также указывает на способ ввода запросов программе mysql. Но точка с запятой — это не часть SQL-синтаксиса для операторов. Так, когда вы делаете запрос в другом контексте, например из сценария Perl или PHP, вы не должны использовать точку с запятой. Если вы этого не сделаете, вполне вероятно появление ошибки.

Создание базы данных

Начнем с процедуры создания базы данных `sampdb`, ее таблиц, заполнения таблиц и выполнения простейших запросов по этим данным.

Работа с базой данных предполагает несколько этапов.

1. Создание (инициализация) базы данных.
2. Создание таблиц в базе данных.
3. Взаимодействие с таблицами посредством операций вставки, выборки, модификации или удаления данных.

Выборка существующих данных фактически является самой частой операцией при работе с базами данных. В числе других наиболее часто используемых операций над таблицами баз данных являются вставка новых данных, их модификация и удаление. Далее частота использования операций располагается следующим образом: операция создания таблиц и операция создания базы данных.

Начнем с самого начала. Поэтому нашей первой операцией будет редко выполняемая операция — создание базы данных. Затем мы перейдем к созданию таблиц, загрузке начальных данных, а уже потом — к наиболее популярной в базах данных операции — выборке.

Для создания базы данных необходимо подключиться к серверу. Для этого воспользуемся клиентской программой `mysql`. После этого с помощью команды `CREATE DATABASE sampdb;` зададим имя базы данных.

```
mysql> CREATE DATABASE sampdb;
```

Очевидно, что, перед тем как приступить к созданию различных таблиц, необходимо создать саму базу данных, которая будет включать их в себя.

Каков механизм создания базы данных? Делает ли создание базы данных ее текущей базой данных? Нет. Это можно увидеть по результату запроса.

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

Для перевода базы данных `sampdb` в статус текущей необходимо ввести оператор `USE`.

```
mysql> USE sampdb
mysql> SELECT DATABASE();
+-------------+
| DATABASE() |
+-------------+
| sampdb ....|
+-------------+
```

В качестве другого способа активизации базы данных можно назвать следующий: достаточно указать имя базы данных в командной строке вызова клиента.

```
% mysql sampdb
```

Это обычный способ активизации базы данных. Перед именем базы данных можно задать другие инициализирующие параметры. Например, следующие две команды позволят подключиться к базе данных `sampdb` на локальном узле и на узле `cobra.snake.net`:

```
% mysql -p -u sampadm sampdb
% mysql -h cobra.snake.net -p -u sampadm sampdb
```

Во всех последующих примерах предполагается, что при запуске `mysql` в командной строке активизируется база данных `sampdb`.

Создание таблиц

В этом разделе вы узнаете, как создавать таблицы, составляющие базу данных `sampdb`. Сначала рассмотрим таблицы из проекта "Историческая Лига". Затем наступит очередь таблиц из проекта "Учет успеваемости". Сейчас мы находимся в той точке повествования, с которой во многих книгах начинаются рассуждения об анализе и о проектировании, диаграммах "сущность–связь", процедурах нормализации. В такой последовательности изложения есть рациональное зерно, но я предлагаю прежде всего задуматься над тем, как будет выглядеть наша база данных: что будут содержать таблицы и как отображать наши данные.

Есть масса вариантов представления данных. В зависимости от требований, предъявляемых к приложениям, и от предназначения данных, одни и те же данные можно представить различными способами.

Таблицы проекта "Историческая Лига"

Набор таблиц для проекта "Историческая Лига" достаточно прост.

- **Таблица** `president`. Содержит данные о президентах США. Это нужно для викторины в режиме on-line на Web-узле (интерактивный аналог напечатанной викторины, которая появляется в детском разделе газеты, издаваемой "Исторической Лигой").

- **Таблица** `member`. Содержит текущую информацию о каждом члене "Исторической Лиги". Предназначена для печатной и интерактивной версии списка членов, а также для автоматической рассылки напоминаний о продлении членства.

Таблица `president`

Таблица `president` проще, поэтому с нее и начнем. Она содержит описание основных биографических данных из жизни всех президентов США.

- **Имя.** Имена можно хранить различными способами. Например, для хранения имени и фамилии можно воспользоваться одним столбцом, а можно создать отдельные столбцы. Конечно же, проще воспользоваться первым методом, но это создаст следующие проблемы в будущем.
  - Если в такой столбец сначала вводится имя, а затем фамилия, то сортировка по фамилии будет затруднена.
  - Если ввод производится в обратном порядке, то отображение имени, а затем фамилии будет несколько затруднено.
  - Выборка по именам будет затруднена. Например, при поиске конкретной фамилии можно использовать шаблон. Это менее эффективно и значительно медленнее, чем производить поиск непосредственно по фамилии.

  В таблице `president` для обхода этих ограничений имя и фамилию нужно будет хранить в отдельных столбцах.

  Столбец имени также будет содержать отчество или инициалы. Это не затруднит работу с этим столбцом, так как сортировать по отчеству нет необходимости. Отображение будет иметь вполне приемлемый вид, независимо от формата вывода, "Bush, George W." или "George W. Bush,".

  Существует еще одно небольшое затруднение. Всего один президент (Jimmy Carter) имеет окончание "Jr." в конце имени. Что это значит? В зависимости от формата отображения имя президента будет иметь вид "James E. Carter, Jr." или "Carter, James E., Jr.". Окончание "Jr." не ассоциируется ни с именем, ни с фамилией. Поэтому для хранения его создадим еще одно поле. Вот вам яркий пример того, как всего одно значение может создать проблемы при проектировании структуры базы данных. Отсюда видно, насколько важно иметь как можно более полную информацию о данных, с которыми придется работать на этапе создания базы данных. Но СУБД MySQL позволяет менять структуру баз данных во время ее использования. Это не беда, но этого по возможности нужно избегать.
- **Место рождения (город и штат).** Аналогично имени и фамилии, эти данные можно хранить как вместе, так и отдельно. Проще использовать один столбец. Но и в этом случае раздельное хранение позволит вам делать многие вещи, которые в противном случае будут существенно затруднены. Например, будет проще сделать выборку по президентам, родившимся в каком-то одном штате, если город и штат хранятся раздельно.
- **Даты рождения и смерти.** Единственной проблемой в этом случае является то, что нет даты смерти, если президент еще жив. СУБД MySQL име-

ет механизм присвоения пустого значения NULL, означающее, что значения нет. Это значение будем использовать для обозначения "в настоящее время жив".

Таблица member

Таблица member предназначена для фиксации членства в "Исторической Лиге". Она аналогична таблице president в том смысле, что в ней хранится вся основная описательная информация одного человека. При этом каждая запись таблицы member хранит больше столбцов.

- **Имя.** В этой таблице мы будем хранить имена подобно тому, как это сделано в таблице president в трех отдельных столбцах: фамилия, имя (и при наличии — отчество), суффикс.
- **Идентификационный номер.** Это уникальное значение, присваиваемое каждому новому члену организации. "Историческая Лига" никогда раньше не использовала идентификационных номеров для регистрации своих членов. Теперь это поможет систематизировать записи. (Я предвижу, как вы в дальнейшем будете расширять использование СУБД MySQL. Это произойдет тогда, когда вы начнете пользоваться информацией, хранящейся в базе данных. Когда это произойдет, вы увидите, что для ссылки на имена членов из других вновь созданных таблиц будет удобнее использовать идентификационные номера, хранящиеся в таблице member.)
- **Срок завершения полномочий**. Члены организации должны периодически возобновлять свое членство. В некоторых приложениях может представлять интерес дата самого последнего возобновления членства. Но это не имеет отношения к "Исторической Лиге", где членство может быть возобновлено на определенное количество лет (обычно один, два, три года или пять лет), и дата последнего возобновления не подскажет вам, когда было последнее возобновление. Кроме того, "Историческая Лига" имеет институт так называемого пожизненного членства. Пожизненное членство можно отобразить какой-то достаточно дальней датой, но очевидно, что значение NULL больше подойдет для этой цели. Значение "нет значения" больше соответствует "никогда не завершается".
- **Адрес электронной почты.** Это облегчит общение между членами "Исторической Лиги", имеющими электронную почту. Для секретаря это упростит рассылку напоминаний о возобновлении членства. Отпадет необходимость ходить на почту, да и обойдется это дешевле. Электронной почтой можно воспользоваться и для того, чтобы разослать членам общества их учетные записи с просьбой обновить их в случае, если появились какие-либо изменения в адресе, фамилии и т.д.
- **Почтовый адрес.** Эти данные необходимы для тех членов организации, у которых нет электронной почты (или которые не отвечают на письма). Название улицы, города, почтового индекса будем хранить в отдельных столбцах. Я предполагаю, что все члены "Исторической Лиги" живут в США. Для международных организаций это будет упрощением, но для нашей задачи это вполне подходит. В случае хранения адресов из разных

стран читатель обязательно увязнет в проблемах хранения адреса в различных форматах. Например, применение почтового индекса не является международным стандартом, многие страны делятся на провинции и области, а не на штаты.

- **Телефонный номер.** Очень полезен для поддержания контактов с членами "Исторической Лиги".
- **Ключевые слова для определения области интересов.** Без сомнения, областью научных интересов всех членов "Исторической Лиги" является история США, но вероятно, что каждый из них имеет свои специфические интересы. Этот столбец предназначен для хранения соответствующей информации. Он будет полезен для поиска по интересам.

Создание таблиц

Теперь можно приступить к созданию таблиц базы данных "Историческая Лига". Воспользуемся для этого оператором `CREATE TABLE`, который имеет следующий синтаксис:

```
CREATE TABLE tbl_name (column_specs);
```

где атрибут *`tbl_name`* содержит имя таблицы, а *`column_specs`* – спецификацию столбцов и индексов в таблице (если они имеются). Индексы нужны для ускорения просмотра таблицы. О них мы поговорим позже, в главе 4, "Оптимизация запросов".

Оператор `CREATE TABLE`, предназначенный для создания таблицы `president`, имеет следующий вид:

```
CREATE TABLE president
(
   last_name VARCHAR(15) NOT NULL,
   first_name VARCHAR(15) NOT NULL,
   suffix VARCHAR(5) NULL,
   city VARCHAR(20) NOT NULL,
   state VARCHAR(2) NOT NULL,
   birth DATE NOT NULL,
   death DATE NULL
);
```

Но перед этим нужно запустить `mysql`, используя команду

```
% mysql sampdb
```

Затем введите оператор `CREATE TABLE`, приведенный выше (не забудьте ввести точку с запятой в конце оператора, в противном случае `mysql` не будет знать, где заканчивается оператор).

Для создания таблицы `president` с уже созданным описанием используйте файл `create_president.sql` из дистрибутива `sampdb`. Файл находится в папке `sampdb`, которая создается после распаковки дистрибутива.

Перейдите в эту папку и запустите команду

```
% mysql sampdb < create_president.sql
```

Во время каждого запуска `mysql` в командной строке перед именем базы данных указывайте необходимые параметры подключения (имя узла, имя пользователя и пароль).

Спецификация столбца в операторе CREATE TABLE состоит из имени столбца, типа (вид значений, которые может содержать столбец) и, возможно, некоторых атрибутов столбца.

В таблице president используются два типа столбцов: VARCHAR и DATE.

Тип VARCHAR(*n*) подразумевает, что столбец содержит символьные (строковые) данные переменной длины с максимальной длиной *n* каждый. Значение *n* выбирается в зависимости от того, какой максимальной длины ожидается значение столбца. Так, state объявлен как VARCHAR(2). Этого вполне достаточно, если для хранения названия штата будет использоваться его традиционная двухбуквенная аббревиатура. Другие строковые столбцы должны быть длинней, чтобы вместить более длинные названия.

Еще одним типом данных, которым мы здесь воспользовались, является тип DATE. Вас может удивить то, что формат хранения данных начинается с года. Стандарт формата даты 'CCYY–MM–DD', где CC, YY, MM, и DD представляют соответственно столетие, год, месяц и дату (это стандарт представления данных ANSI SQL; также известен как формат ISO 8601). Например, дата 18 июля 2002 в MySQL выглядит как '2002-07-18', но не как '07-18-2002' или '18-07-2002'.

Единственным атрибутом столбцов, которым мы воспользуемся в таблице president, будет атрибут NULL (значения данного столбца могут быть пустыми) и NOT NULL (значения данного столбца не могут быть пустыми). Большинство столбцов имеют атрибут NOT NULL. Это означает, что они всегда должны быть заполнены. Только два столбца могут иметь пустые значения — это suffix (суффиксы большей частью в именах отсутствуют) и death (некоторые президенты живы, и дата смерти отсутствует).

Оператор CREATE TABLE, предназначенный для создания таблицы member, имеет такой вид:

```
CREATE TABLE member
(
  member_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (member_id),
  last_name VARCHAR(20) NOT NULL,
  first_name VARCHAR(20) NOT NULL,
  suffix VARCHAR(5) NULL,
  expiration DATE NULL DEFAULT '0000-00-00',
  email VARCHAR(100) NULL,
  street VARCHAR(50) NULL,
  city VARCHAR(50) NULL,
  state VARCHAR(2) NULL,
  zip VARCHAR(10) NULL,
  phone VARCHAR(20) NULL,
  interests VARCHAR(255) NULL
);
```

Введите этот оператор из mysql или запустите из оболочки следующую команду:

```
% mysql sampdb < create_member.sql
```

С точки зрения типов столбцов таблица member не представляет особого интереса: все столбцы, за исключением одного, заданы как строки переменной длины. Одно исключение — столбцы member_id и expiration, которые соответственно содержат порядковые номера и даты.

В столбце member_id указываются уникальные числа, чтобы избежать путаницы с учетными записями.

Столбец AUTO_INCREMENT нужен для автоматического генерирования кода учетной записи при добавлении новой учетной записи. Несмотря на то что он просто содержит числа, его объявление состоит из нескольких частей.

- INT указывает на то, что число содержит целые числа (без дробной части).
- UNSIGNED запрещает отрицательные числа.
- NOT NULL указывает на то, что столбец не может быть пустым. (Это значит, что никакой член не может быть без идентификационного номера.)
- AUTO_INCREMENT является специальным атрибутом. Он указывает, что столбец содержит порядковые номера. Механизм AUTO_INCREMENT работает следующим образом: если отсутствует значение столбца member_id (или стоит параметр NULL), когда вы создаете новую учетную запись, MySQL автоматически генерирует порядковый номер и присваивает его столбцу. Таким образом, будет легче присваивать новые номера, так как MySQL сделает это за вас.

PRIMARY KEY указывает на то, что столбец member_id проиндексирован для быстрого поиска, и на то, что каждое значение в столбце должно быть уникально. Последнее свойство требуется для значений идентификационных номеров пользователей, так как предотвращает повторное использование номера. Кроме того, MySQL требует, чтобы у каждого столбца AUTO_INCREMENT был своего рода уникальный индекс, поскольку описание таблицы без него считается недействительным.

Если вы не понимаете назначения AUTO_INCREMENT и PRIMARY KEY, просто рассматривайте их как возможность генерировать идентификационный номер для каждой записи. Неважно, какие значения, лишь бы они были уникальными. (Когда вы будете готовы узнать больше о том, как использовать столбцы AUTO_INCREMENT, обратитесь к главе 2, “Работа с данными в MySQL”.)

Столбец expiration имеет тип данных DATE. По умолчанию он имеет значение “0000-00-00”, не являющееся значением NULL, которое означает, что не была введена дата. Это объясняется тем, что столбец expiration может не иметь значения, что указывает на тот факт, что член организации имеет пожизненное членство. Однако значением по умолчанию столбца, который может иметь пустое значение, есть пустое значение при условии, что другое значение по умолчанию не определено. В этом случае это нежелательно. Возникает ситуация, когда при создании записи для нового члена не вводится дата истечения срока полномочий и член получает пожизненное членство! Воспользовавшись значением по умолчанию “0000–00–00”, мы решаем эту проблему. Это также позволит проводить периодическую проверку на правильность ввода даты.

Давайте убедимся, что нам удалось создать именно такую таблицу, какую мы планировали создать.

```
mysql> DESCRIBE president;
+------------+-------------+------+-----+------------+-------+
| Field      | Type        | Null | Key | Default    | Extra |
+------------+-------------+------+-----+------------+-------+
| last_name  | varchar(15) |      |     |            |       |
| first_name | varchar(15) |      |     |            |       |
| suffix     | varchar(5)  | YES  |     | NULL       |       |
```

```
| city       | varchar(20) |      |     |            |       |
| state      | char(2)     |      |     |            |       |
| birth      | date        |      |     | 0000-00-00 |       |
| death      | date        | YES  |     | NULL       |       |
+------------+-------------+------+-----+------------+-------+
```

В некоторых версиях MySQL результаты работы оператора DESCRIBE содержат информацию о правах доступа. Здесь они не приводятся, так как они слишком длинны.

Все вроде бы нормально, за исключением одного: столбец state имеет тип CHAR(2). Это странно. Разве он не был объявлен как VARCHAR(2)? Да, был, но СУБД MySQL по умолчанию изменила тип с VARCHAR на CHAR. Причина — проблема оптимизации области хранения данных, которой мы здесь еще не касались. Более подробно это обсуждается в главе 3, "Синтаксис и использование языка SQL". А на этом этапе разница между двумя типами не играет существенной роли.

Важным является то, что столбец содержит два символьных значения.

Введя оператор DESCRIBE member, вы увидите аналогичную информацию для таблицы member.

Оператор DESCRIBE полезен в тех случаях, когда пользователь забыл имя столбца или необходимо уточнить тип данных, размер столбца и т. п. Это также полезно для того, чтобы узнать порядок, в котором СУБД MySQL хранит столбцы в таблице. Знать это очень важно при добавлении строк в таблицу и загрузке данных. Для операторов INSERT и LOAD DATA необходимо полное соответствие вводимых данных со структурой таблицы.

Информацию, создаваемую оператором DESCRIBE, можно просмотреть несколькими способами.

Оператор DESCRIBE можно сократить до DESC или ввести оператор EXPLAIN или SHOW.

Эти операторы эквивалентны:

```
DESCRIBE president;
DESC president;
EXPLAIN president;
SHOW COLUMNS FROM president;
SHOW FIELDS FROM president;
```

Кроме того, эти операторы также позволяют ограничить вывод определенных столбцов. Например, вы можете в конце оператора SHOW добавить параметр LIKE, чтобы отобразить информацию только для столбцов, которые соответствуют шаблону:

```
mysql> SHOW COLUMNS FROM president LIKE '%name';
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| last_name  | varchar(15) |      |     |         |       |
| first_name | varchar(15) |      |     |         |       |
+------------+-------------+------+-----+---------+-------+
```

Символ '%' используется как групповой символ, который описан в разделе "Соответствие шаблону". Похожие ограничения могут использоваться вместе с операторами DESCRIBE и EXPLAIN. Точный синтаксис приведен в приложении Г, "Синтаксис SQL".

Оператор SHOW имеет также другие формы, которые могут пригодиться при получении разных типов информации от MySQL. SHOW TABLES выводит таблицы в текущей базе данных, так что вместе с двумя таблицами, которые мы создали в базе данных sampdb, вывод выглядит так:

```
mysql> SHOW TABLES;
+-----------------+
| Tables_in_sampdb |
+-----------------+
| member           |
| president        |
+-----------------+
```

Оператор SHOW DATABASES отображает таблицы, которые управляются сервером, к которому вы подключены:

```
mysql> SHOW DATABASES;
+-----------+
| Database  |
+-----------+
| menagerie |
| mysql     |
| sampdb    |
| test      |
+-----------+
```

На каждом сервере список баз данных может варьироваться, но две из них должны присутствовать всегда. Это sampdb и mysql. Последняя база данных хранит таблицы, содержащие привилегии и права доступа.

Программа mysqlshow предоставляет интерфейс командной строки для тех же видов информации, которые отображает оператор SHOW. Без параметров mysqlshow отображает список баз данных:

```
% mysqlshow
+-------------+
| Databases   |
+-------------+
| menagerie   |
| mysql       |
| sampdb      |
| test        |
+-------------+
```

При указании имени базы данных она отображает таблицы в указанной базе данных:

```
% mysqlshow sampdb
Database: sampdb
+-----------+
| Tables    |
+-----------+
| member    |
| president |
+-----------+
```

Вместе с указанными именами базы данных и таблицы mysqlshow позволяет отобразить информацию о столбцах в таблице подобно тому, как это делает оператор SHOW COLUMNS.

Таблицы проекта “Учет успеваемости”

Попробуем посмотреть на примере учета успеваемости в простом журнале, какого рода таблицы нам нужны для реализации проекта контроля успеваемости. На рис. 1.2 изображена страница из обыкновенного журнала. Основным телом журнала является матрица с оценками. Присутствуют и другие данные, необходимые для того, чтобы придать смысл этой информации. Перечислены имена и идентификационные номера учащихся. (Для простоты ограничимся только четырьмя учащимися.) Вверху записаны даты викторин и тестов. На рисунке видно, что викторины проводились 3, 6, 16 и 23 сентября, а тесты — 9 сентября и 1 октября.

Для контроля всей этой информации нам понадобится таблица итоговых оценок `score`. Что должны хранить записи этой таблицы? В каждой строке указываются имя учащегося, дата викторины или теста и результат. На рис. 1.3 нашло отражение такое представление информации. (Даты представлены в формате `'CCYY-MM-DD'`.)

Учащиеся		Оценки						
ID	name	Q 9/3	Q 9/6	T 9/9	Q 9/1	Q 9/2	T 10/	. .
1	Billy	14	10	73	14	15	67	.
2	Missy	17	10	68	17	14	73	.
3	Johnny	15	10	78	12	17	82	.
4	Jenny	14	13	85	13	19	79	.
. .	. . .	.	.	.	.	.	.	.

Рис. 1.2. Пример журнала успеваемости

Таблица score

name	date	score
Billy	2002-09-23	15
Missy	2002-09-23	14
Johnny	2002-09-23	17
Jenny	2002-09-23	19
Billy	2002-10-01	67
Missy	2002-10-01	73
Johnny	2002-10-01	82
Jenny	2002-10-01	79

Рис. 1.3. Первоначальная структура таблицы **`score`**

Однако похоже, что такая структура таблицы может создать определенные проблемы. Кажется, в ней недостает какой-то информации. Например, глядя на запись рис. 1.3, трудно сказать, результатом чего она является. Результатом викторины или теста? Возможно, при определении окончательной оценки потребуется определить типы результатов. Это нужно в том случае, когда результаты викторин и результаты тестов имеют различный приоритет. Конечно, можно определить тип результата по диапазону результатов, выпадающих на определенную дату (викторины обычно имеют более низкие оценки, чем тесты), но это будет неверно, так как полученный результат будет основываться не на явных данных, а на предположениях.

Можно просто добавить столбец, содержащий флаг T или Q, для определения типа оценки. На рис. 1.4 T – означает оценку за тест, Q – оценку за викторину. Теперь тип оценки объявлен явным образом. Недостатком является то, что эта информация несколько избыточна. Обратите внимание на то, что даты и типы оценок всегда совпадают. Все результаты за 23 сентября имеют тип Q, а за 1 октября – T. Это бесспорно. При записи результатов тестов и викторин мы несколько раз записываем одну ту же дату и один и тот же тип. Кому нужно вводить эту избыточную информацию?

Попробуем воспользоваться другим представлением данных. Вместо записи типов результатов в таблицу `score` мы будем вычислять его из даты. Будем хранить перечень дат и использовать его для определения "типа события": просто найдите в таблице `event` дату, соответствующую дате в таблице `score`. В результате получим тип события. На рис. 1.5 приведена структура этой таблицы и видно, как реагирует эта связь на дату 23 сентября. Подобрав соответствующую запись в таблице `event`, мы увидим, что этот результат является результатом викторины.

Это значительно лучше, чем пытаться определить тип результата, основываясь на догадках. Вместо этого тип результата определяется прямо из базы данных. Это предпочтительнее хранения результатов в таблице `score`, так как при этом тип записывается только один раз.

Однако сейчас мы сопоставляем информацию из нескольких таблиц. Но здесь будет правомочен вопрос о том, не слишком ли это увеличивает объемы работ, не затрудняет ли это работу?

Таблица score

name	date	score	type
Billy	2002-09-23	15	Q
Missy	2002-09-23	14	Q
Johnny	2002-09-23	17	Q
Jenny	2002-09-23	19	Q
Billy	2002-10-01	67	T
Missy	2002-10-01	73	T
Johnny	2002-10-01	82	T
Jenny	2002-10-01	79	T

Рис. 1.4. Измененная структура таблицы `score`, содержащая тип оценки

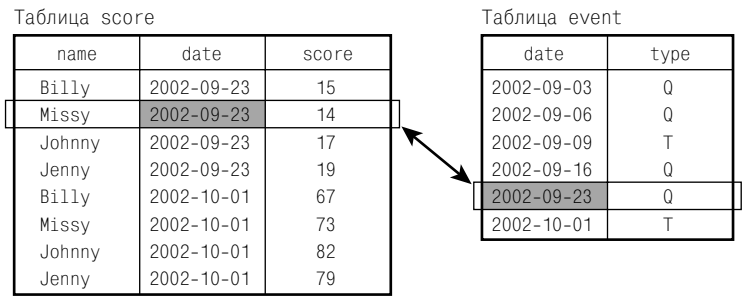

Рис. 1.5. Таблицы `score` и `event`, связанные по дате

Отчасти этот вопрос резонен. Хранить два списка записей труднее, чем один. Но давайте посмотрим еще раз на журнал (см. рис 1.2). Разве вы *уже* не храните два набора записей? Рассмотрим такие факты.

- Результаты отслеживаются в ячейках матрицы результатов. Каждая ячейка отмечена именем учащегося и датой (строки — имена, а столбцы — даты). Это один набор данных. Его аналог — содержимое таблицы `score`.
- Как можно узнать, какой тип события представляет каждая дата? Оказывается, над каждой датой надписано маленькое "T" или "Q"! Таким образом, вы отслеживаете взаимосвязь между датой и типом результата. Это второй набор данных. Его аналог — содержимое таблицы `event`.

Другими словами, даже невзирая на то, что вы думаете обо всем этом, на самом деле вы не делаете ничего, что отличалось бы от того, что предлагает автор. Единственным отличием является то, что в журнале эти два набора данных разделены не столь явно.

Страничка журнала иллюстрирует наше понимание информации и сложность переноса этих данных в базу данных. Обычно мы стараемся объединить различные типы информации и представить все это как единое целое. Базы данных работают не так. Вот почему они кажутся искусственными и неестественными. Природная человеческая потребность унифицировать информацию делает страшной мысль о множестве типов данных вместо одного. Поэтому для читателя может оказаться довольно проблематично "мыслить так, как мыслит база данных".

Единственное требование к таблице `event`: даты должны быть уникальными. Это требование появилось потому, что по дате устанавливается связь между записями из таблиц `score` и `event`. Другими словами, нельзя провести две викторины или викторину и тест в один день. В противном случае у вас получатся два набора записей в таблице `score` и две записи в таблице `event`, все с одной и той же датой. И у вас не будет возможности связать записи таблицы `score` с записями таблицы `event`.

Эта проблема не возникнет, если делать в один день не больше одного экзамена, но насколько вероятно обратное? На первый взгляд, это достаточно правдоподобно; в конце концов, не садист же вы, чтобы назначать и викторину, и тест на один и тот же день. Но я верю, что читатель простит мой скепсис. Очень часто я сталкивался с такой ситуацией, когда мне говорят, что этого никогда не случится, но это вдруг происходит, и мне постфактум приходится переделывать структуру таблиц.

Проблемы, которые могут возникнуть в будущем, надо предвидеть и заранее продумать, как избежать их возникновения. Итак, предположим, что вам может понадобиться записать два результата за день. Что делать в этом случае? Оказывается, эту проблему решать не так уж трудно. При минимальных изменениях в структуре данных мы добиваемся хранения нескольких событий по одной дате.

1. Добавим столбец в таблицу `event` и будем в ней хранить номер, уникальный для каждой записи в таблице. Это присвоит каждой записи таблицы свой собственный идентификационный номер. Назовем ее `event_id`. (Просмотрите журнал на рис. 1.2, если вам покажется это странным: идентификационный номер события очень напоминает номер столбца на матрице результатов вашего журнала. Номер может быть записан явно, даже назван "идентификатором события", но это именно он.)

2. При записи результатов в таблицу `score` вводите идентификатор, а не дату события.

Результат этих изменений показан на рис. 1.6. Связь между таблицами `score` и `event` осуществляется с помощью идентификатора. Теперь в таблице `event` будет храниться не только тип события, но и дата, когда оно произошло. Кроме того, уникальность должна соблюдаться теперь не для даты события, а для его идентификатора. Это означает, что у вас может быть дюжина тестов и викторин в один и тот же день. (Без сомнения, ваши учащиеся начнут нервничать, узнав это.)

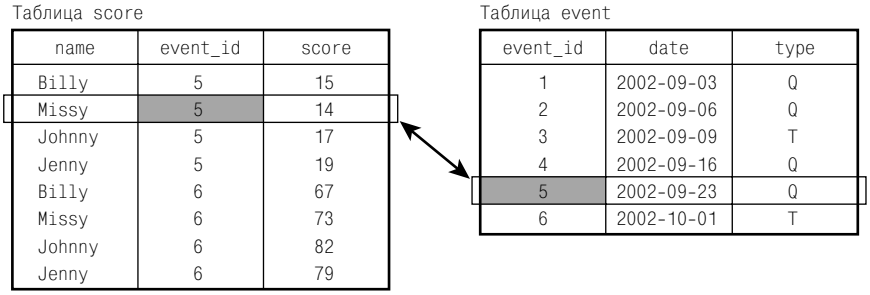

Таблица score

name	event_id	score
Billy	5	15
Missy	5	14
Johnny	5	17
Jenny	5	19
Billy	6	67
Missy	6	73
Johnny	6	82
Jenny	6	79

Таблица event

event_id	date	type
1	2002-09-03	Q
2	2002-09-06	Q
3	2002-09-09	T
4	2002-09-16	Q
5	2002-09-23	Q
6	2002-10-01	T

Рис. 1.6. Таблицы `score` и `event`, связанные по идентификатору события

К сожалению, с моей точки зрения, структура таблицы на рис. 1.6 кажется менее удовлетворительной, чем предыдущая. Таблица `score` является более абстрактной, чем ее предыдущие воплощения. Это явилось следствием того, что стало меньше столбцов с явным содержанием. Структура таблицы `score`, представленная на рис. 1.4, более понятна, так как содержит столбцы с датами и типами результатов. Последняя структура таблицы (см. рис. 1.6) имеет столбцы, назначение которых объяснить уже трудно. Она достаточно далека от понятных категорий. Кому захочется смотреть на таблицу `score` с "идентификатором события"? Для непосвященного это ничего не значит.

Теперь мы стоим на распутье. Вероятно, вы заинтересовались возможностью хранить журнал успеваемости в электронном виде и не заниматься вычислениями вручную при подсчете баллов. Но по мере знакомства с тем, как можно хранить информацию в базе данных, вы заметили, как эта информация абстрактна и несвязанна.

Возникает естественный вопрос: "Насколько целесообразно использовать базу данных? Может быть, СУБД MySQL не для нас?" Как можно догадаться, мой ответ может быть только отрицательным, иначе эта книга теряет свой смысл. Ведь когда вы обдумываете, как сделать работу, не мешает рассмотреть различные альтернативы, например, выбор какой-либо базы данных (предположим, СУБД MySQL) или что-то наподобие электронных таблиц.

- Электронная таблица тоже может хранить данные в виде строк и столбцов. Это делает электронную таблицу и журнал успеваемости концептуально и визуально подобными.
- Используя электронную таблицу, можно производить вычисления. Это позволит суммировать оценки учащихся. Задача немного усложняется "присутствием" весового коэффициента. Но и это преодолимо.

Однако если вам захочется просмотреть только какие-то определенные данные (только тесты или викторины), сравнить оценки девочек с оценками мальчиков, отобразить суммарные данные различными способами — это уже другое дело. Электронной таблице с этими задачами будет уже справиться труднее. А СУБД с этим справляются запросто.

Абстрактность и несвязанность информации, находящейся в таблицах базы данных, — не проблема. Во время планирования и проектирования структуры базы данных необходимо думать о представлении базы данных. Так что во время работы с базой данных ее механизм поможет отображать информацию в понятном виде. Поэтому не надо смотреть на таблицы базы данных просто как на груду несвязной информации.

Например, при выборке результатов из таблицы `scores` идентификаторы событий вам не нужны, а нужны даты. По идентификатору события база данных просмотрит даты из таблицы `event` и выдаст их. Может, просмотреть, с какими результатами вы имеете дело? С результатами тестов или результатами викторин? Это — также не проблема. База данных найдет типы результатов аналогичным образом, — пользуясь идентификатором события. Помните, для чего удобна СУБД, подобная СУБД MySQL: для связывания информации из одного источника с информацией из другого. В случае с данными по успеваемости СУБД MySQL связывает информацию с помощью идентификатора события.

Теперь посмотрим, как работает связь одной таблицы с другой в СУБД MySQL. Для этого сделаем запрос. Предположим, нам требуются все результаты за 23 сентября 2002 года. Получится вот такой запрос:

```
SELECT score.name, event.date, score.score, event.type
FROM score, event
WHERE event.date = '2002-09-23'
AND score.event_id = event.event_id;
```

Невероятно! Посредством объединения (связывания) записей из таблиц `score` и `event` этот запрос производит выборку имени учащегося, даты, оценки и типа оценки. Полученный результат выглядит следующим образом:

```
+--------+------------+-------+------+
| name   | date       | score | type |
+--------+----------- +-------+------+
| Billy  | 2002-09-23 |    15 | Q    |
| Missy  | 2002-09-23 |    14 | Q    |
| Johnny | 2002-09-23 |    17 | Q    |
| Jenny  | 2002-09-23 |    19 | Q    |
+--------+------------+-------+------+
```

Заметили что-то знакомое в формате этой информации? А должны были! Это похоже на структуру таблицы, показанную на рис. 1.4! Нет необходимости знать идентификатор события для того, чтобы получить этот результат. Просто вы описываете дату, которая вас интересует, и предоставляете возможность СУБД MySQL определить, какие оценки принадлежат этой дате. Удивительно, куда делась вся абстракция и связанность, присущая базе данных?

Конечно, посмотрев на все это, у вас останутся вопросы. Например, нельзя ли побыстрее узнать оценки для заданной даты? К сожалению, нет. Однако есть способы избежать ввода нескольких строк на языке SQL каждый раз, когда надо запустить запрос.

Впрочем, я, кажется, немного поторопился, показав этот запрос. Это, поверьте мне, немного проще, чем то, что мы будем использовать для получения оценок. Это значит, что настало время внести еще одну поправку в структуру таблицы. Вместо хранения имени учащегося в таблице `score` будем хранить там его идентификационный номер, т.е. столбец `name` будет заменен столбцом `ID`. Затем будет создана новая таблица `student`, содержащая столбцы `name` и `student_id` (рис. 1.7).

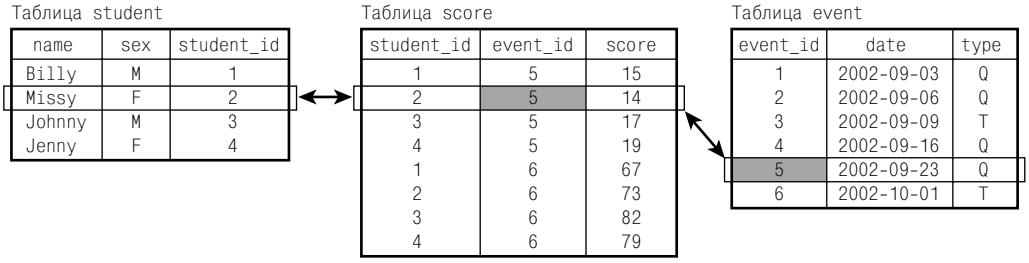

Рис. 1.7. Таблицы `score`, `event` и `student`, связанные по идентификаторам события и учащегося

Зачем вносить эти изменения? Дело в том, что могут существовать двое учащихся с одинаковым именем. Использование уникального идентификатора учащегося позволит вам получить возможность различать их оценки. (Это аналогично тому, как мы с помощью уникального идентификатора события сможем разделять оценки, полученные за викторину, и оценки, полученные за тесты.)

Повторим запрос после изменения структуры таблиц.

```
SELECT student.name, event.date, score.score, event.type
FROM event, score, student
WHERE event.date = '2002-09-23'
AND event.event_id = score.event_id
AND score.student_id = student.student_id;
```

Не расстраивайтесь, если вы не поняли смысл этого запроса сразу. Мы вернемся к нему в процессе более углубленного изучения. Но разница между "сейчас и потом" заключается в том, что потом вы его поймете. Конечно, я не шучу.

Вероятно, вы заметили, что я что-то изменил в таблице `student`, чего не было в журнале. Теперь она содержит столбец для хранения информации о поле учащегося. Это позволит производить более простые операции, например подсчет девочек и мальчиков в классе, или более сложные, такие как сравнение суммарных оценок девочек и мальчиков.

С проектом учета успеваемости учащихся мы почти закончили. Осталась еще одна таблица, предназначенная для отметок посещаемости. Ее содержимое относительно несложно: идентификационный номер учащегося и дата (рис. 1.8). Каждая строка в таблице свидетельствует о том, что определенный учащийся отсутствовал в определенный день. В конце этой главы мы обратимся к вычислительным возможностям СУБД MySQL для того, чтобы узнать, сколько раз учащийся пропустил занятия.

Теперь мы готовы к созданию таблиц проекта контроля успеваемости, поскольку знаем, что они собой представляют.

Таблица absence

student_id	date
2	2002-09-02
4	2002-09-15
2	2002-09-20

Рис. 1.8. Таблица absence

Оператор CREATE TABLE, предназначенный для создания таблицы student, имеет следующий вид:

```
CREATE TABLE student
(
   name VARCHAR(20) NOT NULL,
   sex ENUM('F','M') NOT NULL,
   student_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
   PRIMARY KEY (student_id)
);
```

Введите этот оператор, работая в mysql, или запустите следующую команду из оболочки:

```
% mysql sampdb < create_student.sql
```

После этого оператор CREATE TABLE создаст таблицу student с тремя столбцами: name, sex и student_id.

Столбец name является столбцом переменной длины, который может хранить до 20 символов. Такое представление имени проще, чем в случае с таблицами "Исторической Лиги": один столбец используется для хранения имени и фамилии. Это потому, что я знаю наперед, что ни в одном примере запроса для базы данных контроля успеваемости не потребуется раздельная выборка имени и фамилии.

Столбец sex отражает пол учащегося. Это столбец перечисляемого типа ENUM, а следовательно, что он может принимать одно из двух значений: 'F' или 'M'. Этот тип столбца полезен тогда, когда столбец принимает значения только из ограниченного диапазона. В этом случае можно использовать тип CHAR(1), но ENUM будет явно определять, какие значения может принимать столбец. Оператор DESCRIBE tbl_name для такой таблицы покажет, какие точно значения может принимать этот столбец.

Столбец student_id является столбцом целого типа, в котором будет храниться уникальный идентификационный номер учащегося. Обычно идентификационные номера можно получить из какого-то центрального источника, каким может быть учительская. Мы же создадим их с помощью столбца AUTO_INCREMENT, практически идентичного столбцу member_id, который является частью таблицы member, созданной ранее. Если вы действительно собирались узнать идентификационные номера из учительской, можно было бы создать столбец student_id без атрибута AUTO_INCREMENT, оставив только атрибут PRIMARY KEY, чтобы избежать повторяющихся номеров. Таблица table выглядит следующим образом:

```
CREATE TABLE event
(
   date DATE NOT NULL,
   type ENUM('T','Q') NOT NULL,
   event_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
```

```
PRIMARY KEY (event_id)
);
```

Введите этот оператор в `mysql` или запустите из оболочки следующую команду:

```
% mysql sampdb < create_event.sql
```

Все столбцы объявлены здесь как `NOT NULL` потому, что ни одна из них не может быть пропущена.

Столбец `date` хранит стандартное для СУБД MySQL значение типа `DATE` в формате `'CCYY-MM-DD'`.

Столбец `type` хранит тип оценки и так же, как столбец `sex` в таблице `student` `type`, является перечисляемым типом. Допустимые значения здесь T и Q, что соответствует "тесту" и "викторине".

Столбец `event_id` является столбцом типа `AUTO_INCREMENT`, полностью идентичным столбцу `student_id` в таблице `student`. Аналогично столбцу `student_id`, в таблице `student` конкретные значения менее важны, чем то, что они должны быть уникальными.

Таблица `score` имеет следующий вид:

```
CREATE TABLE score
(
  student_id INT UNSIGNED NOT NULL,
  event_id INT UNSIGNED NOT NULL,
  PRIMARY KEY (event_id, student_id),
  score INT NOT NULL
);
```

Введите этот оператор, работая в `mysql`, или запустите из оболочки следующую команду:

```
% mysql sampdb < create_score.sql
```

Столбцы `student_id` и `event_id` объявлены с типом `INT`, т.е. предполагается, что все значения оценок являются целочисленными и задают ученика и экзамен, к которому относится оценка. Связав по этим полям таблицы `student` и `event`, мы сможем узнать имя учащегося и дату экзамена. Комбинация этих столбцов является первичным ключом. Это предотвратит от дублирования оценок одного учащегося за один и тот же тест или викторину. Упростится также последующее изменение оценки. Например, если оценка была введена неверно, в дальнейшем будет нетрудно изменить ее с помощью оператора `REPLACE`. При этом нет необходимости выполнять операции `DELETE` и `INSERT`.

Обратите внимание на то, что комбинация `student_id` и `event_id` является уникальной. В таблице `score` нет столбца, уникального самого по себе. Каждому идентификатору `event_id` может соответствовать несколько записей (по одной на каждого учащегося). Аналогично, каждому идентификатору `student_id` может соответствовать сразу несколько записей (по одной на каждый тест или викторину).

Столбец `score` является целочисленным столбцом. Я предполагаю, что оценка всегда будет представляться целыми числами. Следовательно, если в оценках необходимо использовать числа, подобные 58,5, нужно задать типы данных с плавающей точкой, такие как `FLOAT` или `DECIMAL`. Таблица `absence` имеет следующую структуру:

```
CREATE TABLE absence
(
```

```
  student_id INT UNSIGNED NOT NULL,
  date DATE NOT NULL,
  PRIMARY KEY (student_id, date)
);
```

Введите этот оператор, работая в `mysql`, или запустите из оболочки следующую команду:

```
% mysql sampdb < create_absence.sql
```

Во избежание потери данных столбцы `student_id` и `date` объявлены как `NOT NULL`. Комбинация этих столбцов объявлена первичным ключом. Это воспрепятствует случайному созданию повторяющихся записей. Ведь действительно будет несправедливо посчитать одного учащегося отсутствующим дважды в день.

Дополнение таблиц

Наконец, наша база данных и ее таблицы созданы. Теперь нужно заполнить таблицы. И неплохо бы знать, как просмотреть содержимое таблицы после добавления новых записей. Несмотря на то что восстановление данных не рассматривается до следующего раздела, вы должны знать хотя бы оператор, который покажет вам содержание таблицы *`tbl_name`*:

```
SELECT * FROM tbl_name;
```

Например:

```
mysql> SELECT * FROM student;
Empty set (0.00 sec)
```

Сейчас видно, что таблица пуста, но после выполнения примеров в этом разделе мы добьемся другого результата.

Можно назвать несколько методов добавления информации в базы данных. Можно добавить записи в таблицы вручную с помощью оператора `INSERT`. Можно добавлять записи прямо из файла или как исходные данные с помощью оператора `LOAD DATA`; можно с помощью программы `mysqlimport`.

В этом разделе демонстрируются все эти методы. Вам остается только проверить их в работе. С помощью команд, описанных здесь, вы сможете удалять и заново создавать таблицы, а также заполнять их данными. Сделав это, можно быть уверенным в том, что эти таблицы будут содержать записи, аналогичные тем, с которыми я буду работать в следующем разделе. (Вы можете пропустить этот раздел, если уже знаете, как добавляются записи.)

Начнем добавление записей с оператора `INSERT`. Это оператор языка SQL, в котором определяется таблица, куда будет производиться добавление, строка добавляемых данных и значений.

Оператор `INSERT` имеет несколько форм.

- Определение значений всех столбцов.

  ```
  INSERT INTO tbl_name VALUES(value1,value2,...);
  ```

  Например:

  ```
  mysql> INSERT INTO student VALUES('Kyle','M',NULL);
  mysql> INSERT INTO event VALUES('2002-9-3','Q',NULL);
  ```

Список VALUE должен содержать все столбцы, имеющиеся в таблице. (Обычно это порядок следования названий столбцов, заданный в операторе CREATE TABLE.)

Если вы не знаете порядка следования столбцов, воспользуйтесь оператором DESCRIBE *tbl_name*.

Выделять строковые значения или значения типа "дата" можно как одинарными, так и двойными кавычками. Пустые значения здесь будут присвоены столбцам с атрибутом AUTO_INCREMENT в таблицах student и event. (Ввод недостающего значения приводит к генерированию следующего номера для student_id или event_id.)

Начиная с версии СУБД MySQL 3.22.5 можно добавить сразу несколько строк с помощью одного оператора INSERT:

```
INSERT INTO tbl_name VALUES(...),(...),...
```

Например:

```
mysql> INSERT INTO student VALUES('Abby','F',NULL),('Kyle','M',NULL);
```

Такой оператор потребует ввода меньшего количества информации, да и сервер будет эффективнее обработать эту команду.

- Вы можете давать имена столбцам, которым хотите присваивать значения, и потом выводить их. Это полезно, если вы хотите создать запись, для которой придется изначально создать несколько столбцов.

```
INSERT INTO tbl_name (col_name1,col_name2,...)
VALUES(value1,value2,...);
```

Например:

```
mysql> INSERT INTO member (last_name,first_name)
VALUES('Stein','Waldo');
```

Начиная с версии СУБД MySQL 3.22.5 можно добавить сразу несколько строк с помощью одного оператора INSERT:

```
mysql> INSERT INTO student (name, sex)
VALUES('Abby','F'),('Kyle','M');
```

Столбцы, не перечисленные в списке столбцов, получают значение по умолчанию. Например, поскольку предыдущие записи не содержат значений в столбце member_id или student_id, MySQL по умолчанию присвоит им значение NULL. (Столбцы member_id и student_id объявлены с атрибутом AUTO_INCREMENT, поэтому результатом будет присвоение порядкового номера точно так же, как это происходит при присвоении значения NULL.)

- Версия СУБД MySQL 3.22.10 позволяет задавать имена столбцов и их значение в форме *col_name = value*:

```
INSERT INTO tbl_name SET col_name1=value1, col_name2=value2, ... ;
```

Например:

```
mysql> INSERT INTO student SET last_name ='Stein',
first_name='Waldo';
```

Столбцы, не перечисленные в выражении SET, получают значение по умолчанию. Такая форма оператора INSERT неприменима для вставки нескольких строк.

Существуют и другие методы загрузки данных в таблицы базы данных прямо из файлов. Например, если у вас есть файл с именем `insert_president.sql`, который содержит оператор `INSERT` для добавления новых записей, вы можете запустить их следующим образом:

```
% mysql sampdb < insert_president.sql
```

(Файл `insert_president.sql` содержится в дистрибутиве `sampdb`.)

Если `mysql` уже запущен, вы можете использовать команду `SOURCE`, чтобы прочитать файл:

```
mysql> SOURCE insert_president.sql;
```

Команда `SOURCE` требует MySQL версии 3.23.9 или более поздней.

Если ваши записи хранятся в файле в необработанном виде, вы можете использовать оператор `LOAD DATA` или программу `mysqlimport` для загрузки этих данных.

Оператор `LOAD DATA` действует как загрузчик массовых данных, считывающий данные из файла. Так он работает из `mysql`:

```
mysql> LOAD DATA LOCAL INFILE "member.txt" INTO TABLE member;
```

Допуская, что файл `member.txt` находится в текущей папке на клиентском компьютере, этот оператор читает его и отправляет на сервер для внесения в таблицу `member`. (Файл `member.txt` вы найдете в дистрибутиве `sampdb`.)

Формат данных в файлах по умолчанию предполагает, что столбцы разделены табуляциями, строки заканчиваются с началом новой строки, значения располагаются в порядке следования столбцов таблицы. Но есть возможность загружать файлы в других форматах или определять другой порядок столбцов. Подробнее с этой проблемой можно ознакомиться в приложении Г, “Синтаксис SQL”.

Вариант `LOAD DATA LOCAL` не работает в версиях до 3.22.15, так как начиная только с этой версии была добавлена возможность чтения данных прямо с компьютера клиента. (Без ключа `LOCAL` загружаемый файл должен быть расположен прямо на сервере, и поэтому для загрузки такого файла пользователь должен обладать широкими правами доступа к серверу, которых у него обычно нет.) Кроме того, в версии 3.23.49 функция `LOCAL` может присутствовать, но быть отключенной. Если при использовании оператора `LOAD DATA` возникает ошибка, попробуйте запустить `mysql` с ключом `- -local-infile`:

```
% mysql --local-infile sampdb
mysql> LOAD DATA LOCAL INFILE 'member.txt' INTO TABLE member;
```

Если так не получится, возможно, требуется использовать функцию `LOCAL` сервером. Чтобы узнать, как это делается, обратитесь к главе 11, “Операторы и функции”.

Программу `mysqlimport` можно рассматривать как интерфейс между вводом на уровне оболочки операционной системы и оператором `LOAD DATA`.

```
% mysql — — local sampdb member.txt
```

Такая команда не сработает для версии СУБД MySQL ниже 3.22.15, поскольку для нее потребуется оператор `LOAD DATA LOCAL`. Здесь все делается так, как это делается в `mysql`: если нужны параметры для связи, указывайте их перед именем базы данных.

Программа `mysqlimport` получает имя таблицы, для которой предназначены данные из имени файла. (Для этого используется все, что указывается в имени

файла до первой точки.) Например, данные из файла `member.txt` будут загружены в таблицу `member`, а из файла `president.txt` — в таблицу `president`. Будьте осторожны, если возникла необходимость загрузить таблицы базы данных из нескольких файлов. Так, при использовании имен `member1.txt` и `member2.txt` для корректной работы программы `mysqlimport` должны использоваться таблицы `member1` и `member2`. В нашем случае подойдут имена `member.1.txt` и `member.2.txt` или `member.txt1` и `member.txt2`.

Если вы уже поэкспериментировали с добавлением данных в тестовую базу данных, удалите содержимое таблиц и загрузите данные, необходимые для ознакомления с материалом следующего раздела.

Для этого прямо из оболочки выполните следующие команды:

```
% mysql sampdb < create_president.sql
% mysql sampdb < insert_president.sql
% mysql sampdb < create_member.sql
% mysql sampdb < insert_member.sql
% mysql sampdb < create_student.sql
% mysql sampdb < insert_student.sql
% mysql sampdb < create_score.sql
% mysql sampdb < insert_score.sql
% mysql sampdb < create_event.sql
% mysql sampdb < insert_event.sql
% mysql sampdb < create_absence.sql
% mysql sampdb < insert_absence.sql
```

Этот ввод можно упростить и ввести команду (в системе UNIX):

```
% sh init_all_tables.sh sampdb
```

Для Windows применима следующая команда:

```
C:\> init_all_tables.bat sampdb
```

Если требуется указать параметры соединения, укажите их в командной строке перед именем базы данных.

Выборка информации

Теперь, когда таблицы созданы и заполнены данными, посмотрим, что можно сделать с этими данными. Оператор `SELECT` позволяет производить выборку и отображать информацию из таблиц любым способом. Можно сделать выборку сразу всех столбцов и всех строк таблицы:

```
mysql> SELECT * FROM president
```

Или сделать выборку одного столбца из одной строки таблицы:

```
mysql> SELECT birth_date FROM president WHERE last_name = "Eisenhover"
```

Оператор `SELECT` состоит из нескольких предложений (частей), которые можно сочетать в любом порядке, в зависимости от того, какая информация требуется для выборки. Любое предложение может быть сложным или простым, в зависимости от чего весь оператор `SELECT` может быть сложным или простым. Но я гарантирую, что в этой книге вы не найдете запросов размером со страницу, для объяснения которых потребуется еще одна такая же страница.

Общий синтаксис оператора SELECT имеет вид

```
SELECT что выбирается
FROM таблица или таблицы
WHERE условия, которые должны удовлетворяться
```

Для создания оператора SELECT необходимо определить, что требуется выбрать из таблиц. Для этой цели служат предложения FROM и WHERE, которые используются наиболее часто, и GROUP BY, ORDER BY и LIMIT, которые встречаются реже. Нужно помнить, что SQL является языком со свободным синтаксисом, поэтому вы не обязательно должны следовать авторскому стилю написания запросов.

Предложение FROM обычно присутствует в операторе SELECT, но в нем нет необходимости, если отображаются данные не из таблиц. Например, следующий запрос просто отображает значения выражений, которые могут вычисляться без ссылки на таблицу, так что в предложении FROM нет необходимости:

```
mysql> SELECT 2+2, 'Hello, world', VERSION();
+-----+--------------+----------------+
| 2+2 | Hello, world | VERSION()      |
+-----+--------------+----------------+
|   4 | Hello, world | 4.0.4-beta-log |
+-----+--------------+----------------+
```

При использовании предложения FROM для определения таблицы, из которой требуется произвести выборку, мы можем получить наиболее "общую" форму запроса. Для этого вместо указания конкретного столбца введем "*", что означает "все". Такой запрос выбирает и отображает все столбцы из таблицы student.

```
mysql> SELECT * FROM student;
+-----------+-----+------------+
| name      | sex | student_id |
+-----------+-----+------------+
| Megan     | F   |          1 |
| Joseph    | M   |          2 |
| Kyle      | M   |          3 |
| Katie     | F   |          4 |
...
```

Значения всех столбцов возвращаются в том же порядке, в котором они хранятся в таблице. Этот порядок совпадает с порядком, в котором столбцы перечислены оператором DESCRIBE student ('...' в конце распечатки означает, что запрос возвращает больше строк, чем показано).

В операторе SELECT имена столбцов можно указывать явно. Для выборки только имен учащихся нужно сделать следующий запрос:

```
mysql> SELECT name FROM student;
+-----------+
| name      |
+-----------+
| Megan     |
| Joseph    |
| Kyle      |
| Katie     |
...
```

Для отображения нескольких столбцов в операторе SELECT можно через запятую указать имена этих столбцов. Этот оператор аналогичен оператору SELECT * FROM student, но каждый столбец указывается здесь явным образом:

```
mysql> SELECT name, sex, student_id FROM student;
+-----------+-----+------------+
| name      | sex | student_id |
+-----------+-----+------------+
| Megan     | F   |          1 |
| Joseph    | M   |          2 |
| Kyle      | M   |          3 |
| Katie     | F   |          4 |
...
```

Столбцы можно перечислять в произвольном порядке.

```
SELECT name, student_id FROM student;
SELECT student_id, name FROM student;
```

Столбцы можно указывать сколько угодно раз.

MySQL позволяет вам выбирать столбцы больше чем из одной таблицы за раз. Вы узнаете об этом ниже, в разделе "Выборка данных из нескольких таблиц".

Имена столбцов можно указывать в любом регистре.

```
SELECT name, student_id FROM student;
SELECT NAME, STUDENT_ID FROM student;
SELECT nAmE, sTuDeNt_Id FROM student;
```

С другой стороны, имена таблиц и баз данных могут быть чувствительны к регистру. Все зависит от файловой системы ОС, работающей на узле сервера. На сервере, работающем под управлением ОС UNIX, имена таблиц и баз данных чувствительны к регистру, так как имена файлов в ОС UNIX чувствительны к регистру. В ОС Windows имена файлов не чувствительны к регистру, поэтому имена таблиц и баз данных тоже будут не чувствительны к регистру.

(Исключением является ОС Mac OS X, где системы с файловой системой HFS+ не чувствительны к регистру, а системы с файловой системой UFS — чувствительны.) Если вы хотите, чтобы таблицы в MySQL не зависели от регистра, обратитесь к разделу "Ограничения операционной системы на имена баз данных и таблиц" в главе 10, "Каталог данных MySQL".

Определение критериев выборки

Ограничение набора выбираемых оператором SELECT записей производится с помощью предложения WHERE, которым определяется набор выбираемых строк.

В качестве критериев можно задавать цифровые значения.

```
mysql> SELECT * FROM score WHERE score > 95;
+------------+----------+-------+
| student_id | event_id | score |
+------------+----------+-------+
|          5 |        3 |    97 |
|         18 |        3 |    96 |
|          1 |        6 |   100 |
|          5 |        6 |    97 |
|         11 |        6 |    98 |
|         16 |        6 |    98 |
+------------+----------+-------+
```

Можно задавать в качестве критериев строковые значения. (Обратите внимание на то, что операции сравнения строк обычно не чувствительны к регистру.)

```
mysql> SELECT last_name, first_name FROM president
    -> WHERE last_name='ROOSEVELT';
+-----------+-------------+
| last_name | first_name  |
+-----------+-------------+
| Roosevelt | Theodore    |
| Roosevelt | Franklin D. |
+-----------+-------------+
mysql> SELECT last_name, first_name FROM president
    -> WHERE last_name='roosevelt';
+-----------+-------------+
| last_name | first_name  |
+-----------+-------------+
| Roosevelt | Theodore    |
| Roosevelt | Franklin D. |
+-----------+-------------+
```

Можно также производить выборку по дате.

```
mysql> SELECT last_name, first_name, birth FROM president
    -> WHERE birth < '1750-1-1';
+------------+------------+------------+
| last_name  | first_name | birth      |
+------------+------------+------------+
| Washington | George     | 1732-02-22 |
| Adams      | John       | 1735-10-30 |
| Jefferson  | Thomas     | 1743-04-13 |
+------------+------------+------------+
```

Выборку можно производить и по комбинации значений.

```
mysql> SELECT last_name, first_name, birth, state FROM president
    -> WHERE birth < '1750-1-1' AND (state='VA' OR state='MA');
+------------+------------+------------+-------+
| last_name  | first_name | birth      | state |
+------------+------------+------------+-------+
| Washington | George     | 1732-02-22 | VA    |
| Adams      | John       | 1735-10-30 | MA    |
| Jefferson  | Thomas     | 1743-04-13 | VA    |
+------------+------------+------------+-------+
```

Выражения в предложениях `WHERE` могут содержать арифметические операторы (табл. 1.1), операторы сравнения (табл. 1.2) и логические операторы (табл. 1.3). Выражения группируются с помощью скобок. Операторы могут содержать константы, столбцы таблиц и вызовы функций. Позже вы познакомитесь с применением функций СУБД MySQL в запросах, а их полный список представлен в приложении B, “Операторы и функции”.

Таблица 1.1. Арифметические операторы

Оператор	Значение
+	Сложение
-	Вычитание
*	Умножение
.	Деление

Таблица 1.2. Операторы сравнения

Оператор	Значение
<	Меньше
<=	Меньше или равно
=	Равно
!= или <>	Не равно
>=	Больше или равно
>	Больше

Таблица 1.3. Логические операторы

Оператор	Значение
AND	Логическое “и”
OR	Логическое “или”
NOT	Логическое отрицание

При создании запроса, требующего использования логических операторов, необходимо понимать разницу между оператором логического “и” и обычным использованием слова “и” в повседневной жизни. Предположим, что требуется найти “президентов, родившихся в штате Виргиния, и президентов, родившихся в штате Массачусетс”. Обратите внимание на то, как при этом произносится “и”, что предполагает следующее написание запроса:

```
mysql> SELECT last_name, first_name, state FROM president
   -> WHERE state='VA' AND state='MA';
Empty set (0.36 sec)
```

При отсутствии выборки ясно, что запрос трактуется “выбрать президентов, родившихся как в штате Виргиния, так и в штате Массачусетс”, что совершенно бессмысленно. В английском языке этот запрос можно делать с использованием “и”, но в языке SQL эти два условия объединяются оператором OR.

```
mysql> SELECT last_name, first_name, state FROM president
    -> WHERE state='VA' OR state='MA';
+------------+-------------+-------+
| last_name  | first_name  | state |
+------------+-------------+-------+
| Washington | George      | VA    |
| Adams      | John        | MA    |
| Jefferson  | Thomas      | VA    |
| Madison    | James       | VA    |
| Monroe     | James       | VA    |
| Adams      | John Quincy | MA    |
| Harrison   | William H.  | VA    |
| Tyler      | John        | VA    |
| Taylor     | Zachary     | VA    |
| Wilson     | Woodrow     | VA    |
| Kennedy    | John F      | MA    |
| Bush       | George H.W. | MA    |
+------------+-------------+-------+
```

Нужно принимать во внимание, что различие между разговорным языком и SQL проявляется не только при формулировании собственных запросов, но и

при написании запросов для других людей. Я советую сначала точно определить, чего они хотят. Например, вышеуказанный запрос проще сформулировать следующим образом: “Выбрать президентов, родившихся в штате Виргиния или в штате Массачусетс”.

Значение NULL

NULL — это особенное значение, потому что оно обозначает ситуацию, когда “нет значения”; его нельзя выбрать как обычное значение и сравнить с другим значением, так как это можно сделать с обычными значениями. При сравнении пустого значения с помощью обычного арифметического оператора сравнения будет получен неопределенный результат.

```
mysql> SELECT NULL < 0, NULL = 0, NULL != 0, NULL > 0;
+----------+----------+-----------+----------+
| NULL < 0 | NULL = 0 | NULL != 0 | NULL > 0 |
+----------+----------+-----------+----------+
| NULL     | NULL     | NULL      | NULL     |
+----------+----------+-----------+----------+
```

Более того, нельзя сравнить два значения NULL. Это естественно — результат сравнения двух неизвестных значений сам не может быть известен.

```
mysql> SELECT NULL = NULL, NULL != NULL;
+-------------+--------------+
| NULL = NULL | NULL != NULL |
+-------------+--------------+
| NULL        | NULL         |
+-------------+--------------+
```

Для выборки пустых значений требуется особый синтаксис. Вместо использования символов “=” или “!=” для проверки равенства или неравенства нужно задавать IS NULL или IS NOT NULL.

Например, мы присвоили дате смерти живых президентов значение NULL, поэтому выбрать их будет проще всего, сделав следующий запрос:

```
mysql> SELECT last_name, first_name FROM president WHERE death IS NULL;
+-----------+-------------+
| last_name | first_name  |
+-----------+-------------+
| Ford      | Gerald R    |
| Carter    | James E.    |
| Reagan    | Ronald W.   |
| Bush      | George H.W. |
| Clinton   | William J.  |
| Bush      | George W.   |
+-----------+-------------+
```

Поиск имен, имеющих суффикс, можно осуществить, пользуясь выражением IS NOT NULL.

```
mysql> SELECT last_name, first_name, suffix
    -> FROM president WHERE suffix IS NOT NULL;
+-----------+------------+--------+
| last_name | first_name | suffix |
+-----------+------------+--------+
| Carter    | James E.   | Jr.    |
+-----------+------------+--------+
```

В СУБД MySQL версии 2.23 есть уникальный оператор сравнения '<=>', работающий даже при сравнении пустых значений. Два предыдущих запроса можно изменить, воспользовавшись этим оператором.

```
mysql> SELECT last_name, first_name FROM president WHERE death <=> NULL;
+-----------+-------------+
| last_name | first_name  |
+-----------+-------------+
| Ford      | Gerald R    |
| Carter    | James E.    |
| Reagan    | Ronald W.   |
| Bush      | George H.W. |
| Clinton   | William J.  |
| Bush      | George W.   |
+-----------+-------------+
mysql> SELECT last_name, first_name, suffix
    -> FROM president WHERE NOT (suffix <=> NULL);
+-----------+------------+--------+
| last_name | first_name | suffix |
+-----------+------------+--------+
| Carter    | James E.   | Jr.    |
+-----------+------------+--------+
```

Сортировка результатов запроса

Очень скоро пользователь заметит, что в результате выполнения команды SELECT * FROM *tbl_name* он получает выборку данных из нужной таблицы в том же порядке, в котором она была загружена. Нельзя полностью полагаться, что данные в таблице всегда будут храниться в нужном порядке. В процессе работы с таблицей записи модифицируются, удаляются и добавляются. (Удаление записей оставляет "дыры" неиспользуемого пространства, которые СУБД MySQL попытается заполнить позже, по мере добавления записей в таблицу.)

Вы должны знать следующее: при выборке данных сервер не гарантирует их порядок. Для их сортировки предназначено предложение ORDER BY.

Следующий запрос выводит имена президентов, отсортированные в алфавитном порядке по фамилии:

```
mysql> SELECT last_name, first_name FROM president
    -> ORDER BY last_name;
+------------+---------------+
| last_name  | first_name    |
+------------+---------------+
| Adams      | John          |
| Adams      | John Quincy   |
| Arthur     | Chester A.    |
| Buchanan   | James         |
...
```

Ключевым словом ASC или DESC можно задать порядок сортировки столбца по возрастанию или убыванию. Например, для сортировки имен президентов в обратном (убывающем) порядке можно сделать такой запрос:

```
mysql> SELECT last_name, first_name FROM president
    -> ORDER BY last_name DESC;
+------------+---------------+
```

```
| last_name  | first_name    |
+------------+---------------+
| Wilson     | Woodrow       |
| Washington | George        |
| Van Buren  | Martin        |
| Tyler      | John          |
...
```

По умолчанию принимается порядок сортировки по возрастанию.

Сортировка может задаваться по нескольким столбцам, при этом любой столбец может быть отсортирован независимо от других столбцов как в возрастающем порядке, так и в убывающем. Следующий запрос производит выборку строк из таблицы `president`, отсортированных в убывающем порядке по названию штата, где родились президенты, и по фамилии в убывающем порядке внутри каждого штата.

```
mysql> SELECT last_name, first_name, state FROM president
    -> ORDER BY state DESC, last_name ASC;
+------------+---------------+-------+
| last_name  | first_name    | state |
+------------+---------------+-------+
| Arthur     | Chester A.    | VT    |
| Coolidge   | Calvin        | VT    |
| Harrison   | William H.    | VA    |
| Jefferson  | Thomas        | VA    |
| Madison    | James         | VA    |
| Monroe     | James         | VA    |
| Taylor     | Zachary       | VA    |
| Tyler      | John          | VA    |
| Washington | George        | VA    |
| Wilson     | Woodrow       | VA    |
| Eisenhower | Dwight D.     | TX    |
| Johnson    | Lyndon B.     | TX    |
...
```

Если вы сортируете столбец, который может содержать значения `NULL`, будет ли он отсортирован? Все зависит от версии. В MySQL версии 4.0.2 они всегда будут в начале, даже если указать параметр `DESC`. Прежде они выводились в начале списка при сортировке по возрастанию и в конце — при сортировке по убыванию.

Если вы хотите быть уверены, что значения `NULL` появятся там, где надо, добавьте дополнительный столбец сортировки, который отличает пустые значения от непустых.

Например, чтобы отсортировать президентов по дате смерти, но при этом поместить живущих президентов (со значениями даты смерти `NULL`) в начало списка, используйте следующий запрос:

```
mysql> SELECT last_name, first_name, death FROM president
    -> ORDER BY IF(death IS NULL,0,1), death;
+------------+---------------+------------+
| last_name  | first_name    | death      |
+------------+---------------+------------+
| Ford       | Gerald R.     | NULL       |
| Carter     | James E.      | NULL       |
| Reagan     | Ronald W.     | NULL       |
| Bush       | George H.W.   | NULL       |
| Clinton    | William J.    | NULL       |
```

```
| Bush       | George W.     | NULL       |
| Washington | George        | 1799-12-14 |
| Adams      | John          | 1826-07-04 |
...
| Truman     | Harry S.      | 1972-12-26 |
| Johnson    | Lyndon B.     | 1973-01-22 |
| Nixon      | Richard M.    | 1994-04-22 |
+------------+---------------+------------+
```

А чтобы поместить живущих президентов в конец списка, используйте следующий запрос:

```
mysql> SELECT last_name, first_name, death FROM president
    -> ORDER BY IF(death IS NULL,1,0), death;
+------------+---------------+------------+
| last_name  | first_name    | death      |
+------------+---------------+------------+
| Washington | George        | 1799-12-14 |
| Adams      | John          | 1826-07-04 |
...
| Truman     | Harry S.      | 1972-12-26 |
| Johnson    | Lyndon B.     | 1973-01-22 |
| Nixon      | Richard M.    | 1994-04-22 |
| Ford       | Gerald R.     | NULL       |
| Carter     | James E.      | NULL       |
| Reagan     | Ronald W.     | NULL       |
| Bush       | George H.W.   | NULL       |
| Clinton    | William J.    | NULL       |
| Bush       | George W.     | NULL       |
+------------+---------------+------------+
```

Функция `IF()` определяет выражение и возвращает значение второго и третьего аргумента, в зависимости от того, является ли результат выражения "правдой" или "ложью". В первом запросе `IF()` присваивает строкам с пустыми датами смерти значения 0, а строкам с датами смерти — значения 1, тем самым помещая все строки без даты смерти перед строками с датой смерти. Во втором запросе все происходит наоборот. Это может использоваться начиная с MySQL версии 3.23.2, которая является первой версией, позволяющей помещать выражения в предложения `ORDER BY`.

Ограничение количества строк результатов запроса

Предложение `LIMIT` предназначено для ограничения количества строк, выводимых запросом. Этот элемент синтаксиса очень полезен в комбинации с предложением `ORDER BY`. СУБД MySQL позволяет ограничить вывод первыми `n` строками. Следующий запрос возвращает пять имен президентов, которым посчастливилось родиться первыми:

```
mysql> SELECT last_name, first_name, birth FROM president
    -> ORDER BY birth LIMIT 5;
+------------+------------+------------+
| last_name  | first_name | birth      |
+------------+------------+------------+
| Washington | George     | 1732-02-22 |
| Adams      | John       | 1735-10-30 |
| Jefferson  | Thomas     | 1743-04-13 |
| Madison    | James      | 1751-03-16 |
| Monroe     | James      | 1758-04-28 |
+------------+------------+------------+
```

Изменив порядок сортировки на сортировку по убыванию `ORDER BY birth DESC`, пользователь получит список пяти самых молодых президентов.

```
mysql> SELECT last_name, first_name, birth FROM president
    -> ORDER BY birth DESC LIMIT 5;
+-----------+-------------+------------+
| last_name | first_name  | birth      |
+-----------+-------------+------------+
| Clinton   | William J.  | 1946-08-19 |
| Bush      | George W.   | 1946-07-06 |
| Carter    | James E.    | 1924-10-01 |
| Bush      | George H.W. | 1924-06-12 |
| Kennedy   | John F.     | 1917-05-29 |
+-----------+-------------+------------+
```

Предложение `LIMIT` может помочь выбрать записи из середины выборки. Для этого необходимо указать два значения. Первое значение — это начальная запись результирующего набора, а второе значение — количество строк, которые будут возвращены. Этот запрос аналогичен предыдущему, но с той лишь разницей, что он возвращает пять строк, начиная с одиннадцатой.

```
mysql> SELECT last_name, first_name, birth FROM president
    -> ORDER BY birth LIMIT 10, 5;
+-----------+------------+------------+
| last_name | first_name | birth      |
+-----------+------------+------------+
| Tyler     | John       | 1790-03-29 |
| Buchanan  | James      | 1791-04-23 |
| Polk      | James K.   | 1795-11-02 |
| Fillmore  | Millard    | 1800-01-07 |
| Pierce    | Franklin   | 1804-11-23 |
+-----------+------------+------------+
```

Для того чтобы выбрать произвольную строку из таблицы `president`, воспользуйтесь `ORDER BY RAND()` вместе с `LIMIT`.

```
mysql> SELECT last_name, first_name FROM president
    -> ORDER BY RAND() LIMIT 1;
+-----------+------------+
| last_name | first_name |
+-----------+------------+
| McKinley  | William    |
+-----------+------------+
```

Сортировка по результату такой формулы работает начиная с MySQL версии 3.23.2. Исходя из этого, вы должны создать столбец, содержащий случайные числа (подробности — в разделе “Подавление оптимизации” главы 4, “Оптимизация запросов”).

Подсчет и присвоение имен выводимым значениям столбцов таблиц

Все предыдущие запросы генерировали результат посредством выборки значений из таблицы. СУБД MySQL также позволяет производить подсчет значений из выводимых столбцов. Последующий запрос оценивает простое выражение (константа) и более сложное выражение, включающее несколько арифметических операций и пару вызовов функций.

```
mysql> SELECT 17, FORMAT(SQRT(3*3+4*4),0);
+----+-------------------------+
| 17 | FORMAT(SQRT(3*3+4*4),0) |
+----+-------------------------+
| 17 | 5                       |
+----+-------------------------+
```

Выражения могут содержать ссылки на столбцы таблиц.

```
mysql> SELECT CONCAT(first_name, ' ',last_name),CONCAT(city,',',state)
    -> FROM president;
+---------------------------------+--------------------------+
| CONCAT(first_name,' ',last_name) | CONCAT(city, ', ',state) |
+---------------------------------+--------------------------+
| George Washington               | Wakefield, VA            |
| John Adams                      | Braintree, MA            |
| Thomas Jefferson                | Albemarle County, VA     |
| James Madison                   | Port Conway, VA          |
...
```

Этот запрос сводит имена и фамилии президентов в одну строку, разделив их пробелом, места рождения объединены в одну строку, разделенную запятой.

Выражение, по которому вычисляется результат, отображается в шапке. Отсюда и большая длина столбца (это хорошо видно в предыдущем примере.) Преодолеть это неудобство помогает конструкция AS *name*, присваивающая столбцу псевдоним. В итоге можно получить более вразумительный результат.

```
mysql> SELECT CONCAT(first_name, ' ',last_name) AS Name,
    -> CONCAT(city, ', ',state) AS Birthplace
    -> FROM president;
+----------------------+-------------------------+
| Name                 | Birthplace              |
+----------------------+-------------------------+
| George Washington    | Wakefield, VA           |
| John Adams           | Braintree, MA           |
| Thomas Jefferson     | Albemarle County, VA    |
| James Madison        | Port Conway, VA         |
...
```

Если столбец содержит пробелы, их необходимо заключать в кавычки.

```
mysql> SELECT CONCAT(first_name, ' ',last_name) AS 'President Name',
    -> CONCAT(city, ', ',state) AS 'Place of Birth'
    -> FROM president;
+----------------------+-------------------------+
| President Name       | Place of Birth          |
+----------------------+-------------------------+
| George Washington    | Wakefield, VA           |
| John Adams           | Braintree, MA           |
| Thomas Jefferson     | Albemarle County, VA    |
| James Madison        | Port Conway, VA         |
...
```

Работа с датами

Принципиальный момент, о котором надо всегда помнить, – формат представления дат в СУБД MySQL таков, что год представляется первым. Так, дата 27 июля 2002 года представляется записью “2002–07–27”. Она не может быть

представлена в виде записи “07–27–2002” или “27–07–2002”, хотя это может быть более удобно.

СУБД MySQL позволяет производить с датами следующие действия:

- сортировка по дате (это мы уже видели);
- выбор по определенной дате или диапазону дат;
- выделение фрагментов даты (год, месяц или день);
- вычисление разницы между датами;
- вычисление относительной даты.

Вот несколько примеров таких операций.

Выборка по определенной дате, по точной дате, по сравнению с определенной датой, сравнение значения столбца типа DATE со значением, которым вы интересуетесь.

```
mysql> SELECT * FROM event WHERE date = '2002-10-01';
+------------+------+----------+
| date       | type | event_id |
+------------+------+----------+
| 2002-10-01 | T    |        6 |
+------------+------+----------+
mysql> SELECT last_name, first_name, death
    -> FROM president
    -> WHERE death >= '1970-01-01' AND death < '1980-01-01';
+-----------+------------+------------+
| last_name | first_name | death      |
+-----------+------------+------------+
| Truman    | Harry S.   | 1972-12-26 |
| Johnson   | Lyndon B.  | 1973-01-22 |
+-----------+------------+------------+
```

Для выборки по фрагменту даты также можно воспользоваться функциями YEAR(), MONTH(), DAYOFMONTH(). Например, так можно сделать выборку всех президентов, которые родились в один и тот же месяц, что и автор этой книги (в марте). Для этого произведем выборку всех записей с датами рождения, в которых значится третий месяц.

```
mysql> SELECT last_name, first_name, birth
    -> FROM president WHERE MONTH(birth) = 3;
+-----------+------------+------------+
| last_name | first_name | birth      |
+-----------+------------+------------+
| Madison   | James      | 1751-03-16 |
| Jackson   | Andrew     | 1767-03-15 |
| Tyler     | John       | 1790-03-29 |
| Cleveland | Grover     | 1837-03-18 |
+-----------+------------+------------+
```

Запрос может также содержать название месяца.

```
mysql> SELECT last_name, first_name, birth
    -> FROM president WHERE MONTHNAME(birth) = 'March';
+-----------+------------+------------+
| last_name | first_name | birth      |
+-----------+------------+------------+
| Madison   | James      | 1751-03-16 |
```

```
| Jackson   | Andrew     | 1767-03-15 |
| Tyler     | John       | 1790-03-29 |
| Cleveland | Grover     | 1837-03-18 |
+-----------+------------+------------+
```

Усложним запрос и сделаем выборку президентов, день рождения которых полностью совпадает с днем рождения автора.

```
mysql> SELECT last_name, first_name, birth
    -> FROM president WHERE MONTH(birth) = 3 AND DAYOFMONTH(birth) =
29;
+-----------+------------+------------+
| last_name | first_name | birth      |
+-----------+------------+------------+
| Tyler     | John       | 1790-03-29 |
+-----------+------------+------------+
```

Результат этого запроса напоминает список “Люди, родившиеся сегодня” из раздела “Развлечения” вашей любимой газеты. Для выборки президентов, родившихся сегодня, необходимо сравнить их дни рождения со значением `CURDATE()` (текущей даты).

```
SELECT last_name, first_name, birth
FROM president WHERE MONTH(birth) = MONTH(CURDATE())
AND DAYOFMONTH(birth) = DAYOFMONTH(CURDATE());
```

Для получения интервала между датами их можно вычитать одну из другой. Например, для выяснения того, кто из президентов жил дольше всех, необходимо из даты смерти вычесть дату рождения. Для этого с помощью функции `TO_DAYS()`преобразуем значения `death` и `birth`, произведем вычитание и разделим полученный результат на 365. Получим приблизительный возраст в годах.

```
mysql> SELECT last_name, first_name, birth, death,
    -> TO_DAYS(death) - TO_DAYS(birth) AS age
    -> FROM president WHERE death IS NOT NULL
    -> ORDER BY age DESC LIMIT 5;
+-----------+------------+------------+------------+-------+
| last_name | first_name | birth      | death      | age   |
+-----------+------------+------------+------------+-------+
| Adams     | John       | 1735-10-30 | 1826-07-04 | 33119 |
| Hoover    | Herbert C. | 1874-08-10 | 1964-10-20 | 32943 |
| Truman    | Harry S.   | 1884-05-08 | 1972-12-26 | 32373 |
| Madison   | James      | 1751-03-16 | 1836-06-28 | 31150 |
| Jefferson | Thomas     | 1743-04-13 | 1826-07-04 | 30397 |
+-----------+------------+------------+------------+-------+
```

Для преобразования возраста в днях в приблизительный возраст в годах разделим возраст на 365. (Функция `FLOOR()`, которой мы воспользовались в этом запросе, отсекает дробную часть от полученного числа лет.)

```
mysql> SELECT last_name, first_name, birth, death,
    -> FLOOR((TO_DAYS(death) - TO_DAYS(birth))/365) AS age
    -> FROM president WHERE death IS NOT NULL
    -> ORDER BY age DESC LIMIT 5;
+-----------+------------+------------+------------+-----+
| last_name | first_name | birth      | death      | age |
+-----------+------------+------------+------------+-----+
| Adams     | John       | 1735-10-30 | 1826-07-04 | 90  |
```

```
| Hoover    | Herbert C. | 1874-08-10 | 1964-10-20 | 90  |
| Truman    | Harry S.   | 1884-05-08 | 1972-12-26 | 88  |
| Madison   | James      | 1751-03-16 | 1836-06-28 | 85  |
| Jefferson | Thomas     | 1743-04-13 | 1826-07-04 | 83  |
+-----------+------------+------------+------------+-----+
```

В этом частном случае значения возраста соответствуют действительному возрасту на момент смерти. Так получается не всегда, поскольку год не всегда равен 365 дням. Для подсчета лет мы обычно берем разницу между годами, а потом вычитаем единицу, если день смерти раньше дня рождения.

```
mysql> SELECT last_name, first_name, birth, death,
    -> (YEAR(death) - YEAR(birth)) - IF(RIGHT(death,5) <
RIGHT(birth,5),1,0)
    -> AS age
    -> FROM president WHERE death IS NOT NULL
    -> ORDER BY age DESC LIMIT 5;
+-----------+------------+------------+------------+-------+
| last_name | first_name | birth      | death      | age   |
+-----------+------------+------------+------------+-------+
| Adams     | John       | 1735-10-30 | 1826-07-04 |    90 |
| Hoover    | Herbert C. | 1874-08-10 | 1964-10-20 |    90 |
| Truman    | Harry S.   | 1884-05-08 | 1972-12-26 |    88 |
| Madison   | James      | 1751-03-16 | 1836-06-28 |    85 |
| Jefferson | Thomas     | 1743-04-13 | 1826-07-04 |    83 |
+-----------+------------+------------+------------+-------+
```

Выражение `IF()` выполняет тест календарных дней, который основывается на простом построчном сравнении пяти последних символов дат. Так получается по двум причинам. Во-первых, MySQL трактует даты как строки, если обрабатывать их строковыми функциями, в данном случае функцией `RIGHT()`, которая возвращает `n` крайних правых символов строки. Во-вторых, MySQL генерирует даты с ограниченным числом цифр в каждой из частей даты. Сравнение не будет работать, если в значениях дней и месяцев меньше 10 не будет предшествующего нуля.

Получение разности между датами полезно также для определения, какой период времени разделяет эти две даты. С помощью подобных вычислений можно напоминать членам "Исторической Лиги" о необходимости обновить свое членство. Вычислите разность между датой истечения срока членства и текущей датой. Если она не превышает допустимого значения, скоро понадобится обновление. Ниже приведен запрос, делающий выборку членов, которые должны обновить членство в течение 60 дней.

```
SELECT last_name, first_name, expiration FROM member
WHERE (TO_DAYS(expiration) - TO_DAYS(CURDATE())) < 60;
```

Для вычитания одной даты из другой можно использовать функции `DATE_ADD()` и `DATE_SUB()`, которые по дате и интервалу возвращают новую дату, например:

```
mysql> SELECT DATE_ADD('1970-1-1', INTERVAL 10 YEAR);
+---------------------------------------+
| DATE_ADD('1970-1-1', INTERVAL 10 YEAR) |
+---------------------------------------+
| 1980-01-01                            |
+---------------------------------------+
mysql> SELECT DATE_SUB('1970-1-1', INTERVAL 10 YEAR);
+---------------------------------------+
| DATE_SUB('1970-1-1', INTERVAL 10 YEAR) |
```

```
+----------------------------------------+
| 1960-01-01                             |
+----------------------------------------+
```

Запрос, приведенный выше в этом разделе, делал выборку президентов, умерших в 1970-х годах. Для задания даты окончания интервала использовались литералы. Перепишем этот запрос по-другому: укажем с помощью литерала начальную дату.

```
mysql> SELECT last_name, first_name, death
    -> FROM president
    -> WHERE death >= '1970-1-1'
    -> AND death < DATE_ADD('1970-1-1', INTERVAL 10 YEAR);
+-----------+------------+------------+
| last_name | first_name | death      |
+-----------+------------+------------+
| Truman    | Harry S.   | 1972-12-26 |
| Johnson   | Lyndon B.  | 1973-01-22 |
+-----------+------------+------------+
```

Запрос на обновление членства можно переписать с применением функции `DATE_ADD()`:

```
SELECT last_name, first_name, expiration FROM member
WHERE expiration < DATE_ADD(CURDATE(), INTERVAL 60 DAY);
```

Ранее в этой главе был представлен запрос для определения тех пациентов стоматолога, кто забыл пройти регулярный профилактический осмотр.

```
SELECT last_name, first_name, last_visit FROM patient
WHERE last_visit < DATE_SUB(CURDATE(),INTERVAL 6 MONTH);
```

Понятен ли этот запрос теперь?

Соответствие шаблону

СУБД MySQL позволяет производить выборку значений, которые соответствуют определенному шаблону. При этом не надо задавать определенное значение. Для выборки можно использовать специальные операторы (`LIKE` и `NOT LIKE`) и указать строку, содержащую символы-заменители. Символ "_" соответствует значению "любой одиночный символ". Символ "%" соответствует значению "любая последовательность символов" (включая и пустую последовательность). Шаблоны, удовлетворяющие `LIKE` и `NOT LIKE`, не чувствительны к регистру.

Ниже приведен шаблон, соответствующий всем фамилиям, начинающимся с буквы "W" или "w".

```
mysql> SELECT last_name, first_name FROM president
    -> WHERE last_name LIKE 'W%';
+------------+------------+
| last_name  | first_name |
+------------+------------+
| Washington | George     |
| Wilson     | Woodrow    |
+------------+------------+
```

С другой стороны, это ошибочный шаблон:

```
mysql> SELECT last_name, first_name FROM president
    -> WHERE last_name = 'W%';
Empty set (0.00 sec)
```

Это типичная ошибка при работе с шаблонами. Такой запрос даст результат только в том случае, если существует столбец, в действительности содержащий "W%" или "w%".

Вот запрос на выборку фамилий, в которых встречается "W" или "w".

```
mysql> SELECT last_name, first_name FROM president
    -> WHERE last_name LIKE '%W%';
+------------+------------+
| last_name  | first_name |
+------------+------------+
| Washington | George     |
| Wilson     | Woodrow    |
| Eisenhower | Dwight D.  |
+------------+------------+
```

Запрос на выборку фамилий, которые содержат четыре буквы, выглядит так:

```
mysql> SELECT last_name, first_name FROM president
    -> WHERE last_name LIKE '____';
+-----------+-------------+
| last_name | first_name  |
+-----------+-------------+
| Polk      | James K.    |
| Taft      | William H.  |
| Ford      | Gerald R    |
| Bush      | George H.W. |
| Bush      | George W.   |
+-----------+-------------+
```

СУБД MySQL имеет также и другую форму выборки по шаблону с помощью расширенных регулярных выражений. Регулярные выражения описаны в приложении B, "Операторы и функции".

Настройка и использование переменных SQL

MySQL версий 3.23.6 и выше позволяет настроить переменные при помощи результатов запросов, которые предоставляют возможность сохранить переменные для использования в последующих запросах. Предположим, вы хотите узнать, какие президенты родились до Эндрю Джексона. Для этого следует ввести его дату рождения в переменную, а затем выбрать президентов с датой рождения меньше даты, указанной в переменной.[1]

```
mysql> SELECT @birth := birth FROM president
    -> WHERE last_name = 'Jackson' AND first_name = 'Andrew';
+-----------------+
| @birth := birth |
+-----------------+
| 1767-03-15      |
+-----------------+
mysql> SELECT last_name, first_name, birth FROM president
    -> WHERE birth < @birth ORDER BY birth;
+------------+------------+------------+
| last_name  | first_name | birth      |
+------------+------------+------------+
```

[1]Эту проблему можно решить, используя запрос с объединением, но нам еще рановато создавать объединения. Кроме того, иногда проще использовать переменную.

```
| Washington | George     | 1732-02-22 |
| Adams      | John       | 1735-10-30 |
| Jefferson  | Thomas     | 1743-04-13 |
| Madison    | James      | 1751-03-16 |
| Monroe     | James      | 1758-04-28 |
+------------+------------+------------+
```

Переменным присваиваются имена при помощи синтаксиса `@name`, а значения — посредством оператора `SELECT`, используя выражение типа `@name := value`. Первый запрос ищет дату рождения Эндрю Джексона и присваивает значение переменной `@birth`. (Результат оператора SELECT отображается, так как присвоение результата переменной не приводит к игнорированию вывода запроса.) Второй запрос обращается к переменной и использует ее значение, чтобы найти президента с более ранней датой рождения. Переменные также могут назначаться оператором `SET`, несмотря на то, что в этом случае не важен вид оператора присваивания (он может иметь вид либо `=`, либо `:=`.)

```
mysql> SET @one_week_ago = DATE_SUB(CURDATE(),INTERVAL 7 DAY);
mysql> SELECT CURDATE(), @one_week_ago;
+------------+---------------+
| CURDATE()  | @one_week_ago |
+------------+---------------+
| 2002-09-03 | 2002-08-27    |
+------------+---------------+
```

Получение итоговых результатов

Одна из самых полезных возможностей СУБД MySQL (как, впрочем, и всех прочих СУБД) — обработка огромных объемов данных и их суммирование. По мере ознакомления с СУБД MySQL она становится мощным союзником в деле получения итогов и сумм, особенно принимая во внимание тот факт, что это очень нудная, требующая больших временных затрат и являющаяся источником ошибок работа, особенно если выполнять ее вручную.

Простым примером этого может служить необходимость определения значений, которые представлены в таблице. Для удаления повторений воспользуемся ключевым словом `DISTINCT`. Например, ниже представлен перечень штатов, в которых когда-либо рождались президенты США.

```
mysql> SELECT DISTINCT state FROM president ORDER BY state;
+-------+
| state |
+-------+
| AR    |
| CA    |
| CT    |
| GA    |
| IA    |
| IL    |
| KY    |
| MA    |
| MO    |
| NC    |
| NE    |
| NH    |
| NJ    |
| NY    |
```

```
| OH    |
| PA    |
| SC    |
| TX    |
| VA    |
| VT    |
+-------+
```

О другом способе получения итогов очень легко догадаться — это использование функции `COUNT()`. Выражение `COUNT(*)` выдает количество строк, выбранных запросом. Если запрос не содержит предложения `WHERE`, выражение `COUNT(*)` будет подсчитывать количество строк, содержащихся в таблице.

Следующий запрос показывает, сколько всего членов в таблице членства "Исторической Лиги":

```
mysql> SELECT COUNT(*) FROM member;
+----------+
| COUNT(*) |
+----------+
|      102 |
+----------+
```

При наличии предложения `WHERE` функция `COUNT(*)` покажет, сколько строк удовлетворяет заданному критерию выборки.

```
mysql> SELECT COUNT(*) FROM event WHERE type = 'Q';
+----------+
| COUNT(*) |
+----------+
|        4 |
+----------+
```

Функция `COUNT(*)` подсчитывает каждую выбранную строку. Функция `COUNT(col_name)` подсчитывает только строки, содержащие ненулевые значения в данном столбце. Следующий запрос отчетливо демонстрирует это различие:

```
mysql> SELECT COUNT(*), COUNT(email), COUNT(expiration) FROM member;
+----------+--------------+-------------------+
| COUNT(*) | COUNT(email) | COUNT(expiration) |
+----------+--------------+-------------------+
|      102 |           80 |                96 |
+----------+--------------+-------------------+
```

Отсюда видно, что всего в таблице `member` 102 записи, только 80 из них имеют адрес электронной почты. Также видно, что 6 членов имеют пожизненное членство. (Отсутствие значения в столбце `expiration` указывает на пожизненное членство, а так как у 96 из 102 записей есть значение, получается 6 пожизненных членов.)

В версиях выше 3.23.2 уже можно совмещать функцию `COUNT(*)` с ключевым словом `DISTINCT` для подсчета количества точных ответов, например, для подсчета количества штатов, в которых когда-либо рождались президенты.

```
mysql> SELECT COUNT(DISTINCT state) FROM president;
+-----------------------+
| COUNT(DISTINCT state) |
+-----------------------+
|                    20 |
+-----------------------+
```

Можно производить подсчеты и по отдельным категориям. Например, общее число учащихся по вашему предмету можно определить с помощью такого запроса:

```
mysql> SELECT COUNT(*) FROM student;
+----------+
| COUNT(*) |
+----------+
|       31 |
+----------+
```

Но сколько из них мальчиков и девочек?

```
mysql> SELECT COUNT(*) FROM student WHERE sex='f';
+----------+
| COUNT(*) |
+----------+
|       15 |
+----------+
mysql> SELECT COUNT(*) FROM student WHERE sex='m';
+----------+
| COUNT(*) |
+----------+
|       16 |
+----------+
```

Несмотря на то что такой метод применим, он не иллюстративен и не очень подходит для случая, когда столбцы имеют несколько различных значений. Предположим, что необходимо определить, сколько президентов родилось в каждом штате. В соответствии с вышеприведенным методом сначала с помощью запроса `SELECT DISTINCT state FROM president` нужно узнать, сколько таких штатов, а затем сделать запрос `SELECT COUNT(*)` для каждого полученного штата. Очевидно, что это излишне трудоемкая процедура.

К счастью, СУБД MySQL позволяет производить подсчет с помощью одного лишь запроса. Количество девочек и мальчиков можно одновременно подсчитать следующим образом:

```
mysql> SELECT sex, COUNT(*) FROM student GROUP BY sex;
+-----+----------+
| sex | COUNT(*) |
+-----+----------+
| F   |       15 |
| M   |       16 |
+-----+----------+
```

Аналогичный запрос покажет, сколько президентов родилось в каждом штате.

```
mysql> SELECT state, COUNT(*) FROM president GROUP BY state;
+-------+----------+
| state | COUNT(*) |
+-------+----------+
| AR    |        1 |
| CA    |        1 |
| CT    |        1 |
| GA    |        1 |
| IA    |        1 |
| IL    |        1 |
| KY    |        1 |
| MA    |        4 |
```

```
| MO    |         1 |
| NC    |         2 |
| NE    |         1 |
| NH    |         1 |
| NJ    |         1 |
| NY    |         4 |
| OH    |         7 |
| PA    |         1 |
| SC    |         1 |
| TX    |         2 |
| VA    |         8 |
| VT    |         2 |
+-------+-----------+
```

При таком подсчете нам поможет предложение `GROUP BY`, которое сообщает базе данных, каким образом группировать значения. Его отсутствие будет расценено СУБД MySQL как ошибка.

Использование `COUNT(*)` и `GROUP BY` для подсчета значений имеет ряд преимуществ при подсчете количества вхождений определенных значений столбцов:

- нет необходимости знать все значения, имеющиеся в столбце, значения которого подсчитываются;
- нет необходимости в серии последовательных запросов, вполне достаточно одного;
- так как результат получен одним запросом, он вполне поддается сортировке.

Первые два преимущества позволяют создавать запросы более простым способом. Третье преимущество также важно, поскольку позволяет гибко манипулировать полученными результатами. При задании предложения `GROUP BY` результаты группируются по группируемому столбцу. Но это совсем не исключает применения группировки с помощью `ORDER BY` в другом порядке. Например, нужно подсчитать количество президентов по штату, в котором они родились, но отсортировать их по убыванию, начиная со штата, в котором родилось больше всего президентов США.

```
mysql> SELECT state, COUNT(*) AS count FROM president
    -> GROUP BY state ORDER BY count DESC;
+-------+-------+
| state | count |
+-------+-------+
| VA    |     8 |
| OH    |     7 |
| MA    |     4 |
| NY    |     4 |
| NC    |     2 |
| VT    |     2 |
| TX    |     2 |
| SC    |     1 |
| NH    |     1 |
| PA    |     1 |
| KY    |     1 |
| NJ    |     1 |
| IA    |     1 |
| MO    |     1 |
| CA    |     1 |
```

```
| NE    |     1 |
| GA    |     1 |
| IL    |     1 |
| AR    |     1 |
| CT    |     1 |
+-------+-------+
```

Столбцу, который нужно отсортировать, но который является результатом вычислений, можно присвоить псевдонимы и в предложении `ORDER BY` ссылаться прямо на него. Это продемонстрировано в последнем запросе. Столбцу `COUNT(*)` присвоен псевдоним `count`. Можно ссылаться на такой столбец и по-другому: указав порядок его следования в выводимых данных. Перепишем этот запрос следующим образом:

```
SELECT state, COUNT(*) FROM president
GROUP BY state ORDER BY 2 DESC;
```

Я не нахожу читабельной ссылку на столбец по его позиции. К примеру, при добавлении, удалении или изменении порядка следования столбцов в этом случае нужно помнить о необходимости проверки предложения `ORDER BY` и изменить номер столбца, если он менялся. Псевдонимы не приводят к таким проблемам.

Это справедливо для предложения `GROUP BY`. При использовании предложения `GROUP BY` с вычисляемым столбцом ссылка на него может производиться как по его позиции в списке столбцов, так и по его псевдониму. Вот запрос, который определяет, сколько президентов США родилось в каждом месяце.

```
mysql> SELECT MONTH(birth) AS Month, MONTHNAME(birth) AS Name,
    -> COUNT(*) AS count
    -> FROM president GROUP BY Name ORDER BY Month;
+-------+-----------+-------+
| Month | Name      | count |
+-------+-----------+-------+
|     1 | January   |     4 |
|     2 | February  |     4 |
|     3 | March     |     4 |
|     4 | April     |     4 |
|     5 | May       |     2 |
|     6 | June      |     1 |
|     7 | July      |     4 |
|     8 | August    |     4 |
|     9 | September |     1 |
|    10 | October   |     6 |
|    11 | November  |     5 |
|    12 | December  |     3 |
+-------+-----------+-------+
```

С указанием позиций столбцов запрос будет иметь следующий вид:

```
SELECT MONTH (birth), MONTHNAME(birth), COUNT(*)
FROM president GROUP BY 2 ORDER BY 1;
```

Функцию `COUNT()` можно указать в комбинации с `ORDER BY` и `LIMIT` для того, чтобы, например, найти четыре наиболее широко представленных штата из таблицы `president`.

```
mysql> SELECT state, COUNT(*) AS count FROM president
    -> GROUP BY state ORDER BY count DESC LIMIT 4;
```

```
+-------+-------+
| state | count |
+-------+-------+
| VA    |     8 |
| OH    |     7 |
| MA    |     4 |
| NY    |     4 |
+-------+-------+
```

Если вы не хотите ограничивать вывод предложением `LIMIT`, анализируя только значение, выдаваемое счетчиком `COUNT(*)`, используйте предложение `HAVING`. Предложение `HAVING` похоже на предложение `WHERE` тем, что оно указывает условия, которым должны соответствовать выводимые строки. Отличается же это предложение тем, что может ссылаться на результаты агрегирующих функций, таких как функция `COUNT()`. Следующий запрос покажет, какие штаты представляются двумя или более президентами:

```
mysql> SELECT state, COUNT(*) AS count FROM president
    -> GROUP BY state HAVING count > 1 ORDER BY count DESC;
+-------+-------+
| state | count |
+-------+-------+
| VA    |     8 |
| OH    |     7 |
| MA    |     4 |
| NY    |     4 |
| NC    |     2 |
| VT    |     2 |
| TX    |     2 |
+-------+-------+
```

Более общий случай применения этого типа запросов — поиск повторений (или поиск неповторяющихся значений с использованием `HAVING count = 1`).

Кроме `COUNT()`, существуют и другие агрегирующие функции. Функции `MIN()`, `MAX()`, `SUM()` и `AVG()` предназначены для вычисления минимума, максимума, суммы и среднего значений столбца соответственно. Их можно использовать одновременно. Вот запрос, который позволяет производить подсчеты различных характеристик для любого теста или викторины. Здесь видно, как много подсчетов можно произвести по любому из значений. (Некоторые учащиеся отсутствуют, и их оценки не попадают в подсчеты.)

```
mysql> SELECT
    -> event_id,
    -> MIN(score) AS minimum,
    -> MAX(score) AS maximum,
    -> MAX(score)-MIN(score)+1 AS range,
    -> SUM(score) AS total,
    -> AVG(score) AS average,
    -> COUNT(score) AS count
    -> FROM score
    -> GROUP BY event_id;
+----------+---------+---------+-------+-------+---------+-----+
| event_id | minimum | maximum | range | total | average |count|
+----------+---------+---------+-------+-------+---------+-----+
|        1 |       9 |      20 |     12 |  439 | 15.1379 | 29  |
|        2 |       8 |      19 |     12 |  425 | 14.1667 | 30  |
|        3 |      60 |      97 |     38 | 2425 | 78.2258 | 31  |
```

```
|        4 |       7 |      20 |     14 |  379 | 14.0370 | 27  |
|        5 |       8 |      20 |     13 |  383 | 14.1852 | 27  |
|        6 |      62 |     100 |     39 | 2325 | 80.1724 | 29  |
+----------+---------+---------+--------+------+---------+-----+
```

Конечно, информация может быть более осмысленной, когда известно, откуда получены значения — из викторины или из теста. Для получения этой информации необходимо также обратиться к таблице `event_table`. Мы вернемся к этому запросу в разделе "Выборка данных из нескольких таблиц". Работать с агрегирующими функциями очень просто, так как они достаточно мощны. Но можно легко получить ошибочный результат. Рассмотрим следующий запрос:

```
mysql> SELECT
    -> state AS State,
    -> AVG((TO_DAYS(death)-TO_DAYS(birth))/365) AS Age
    -> FROM president WHERE death IS NOT NULL
    -> GROUP BY state ORDER BY Age;
+-------+-----------+
| State | Age       |
+-------+-----------+
| KY    | 56.208219 |
| VT    | 58.852055 |
| NC    | 60.141096 |
| OH    | 62.866145 |
| NH    | 64.917808 |
| NY    | 69.342466 |
| NJ    | 71.315068 |
| TX    | 71.476712 |
| MA    | 72.642009 |
| VA    | 72.822945 |
| PA    | 77.158904 |
| SC    | 78.284932 |
| CA    | 81.336986 |
| MO    | 88.693151 |
| IA    | 90.254795 |
+-------+-----------+
```

Этот запрос производит выборку умерших президентов, группирует их по дате рождения, определяет их возраст на момент смерти, вычисляет средний возраст по штату, а затем сортирует полученные результаты по среднему возрасту. Другими словами, запрос определяет средний возраст по штатам на момент смерти для всех президентов США, которых уже нет в живых.

Что же он показывает? Он показывает только то, что вы умеете писать запросы. И ничего больше. Не все операции, которые можно осуществлять с базами данных, имеют смысл. Однако люди иногда впадают в состояние эйфории, когда вдруг понимают, что они могут "вытворять" с базами данных. В результате этого в последние годы особенно бросается в глаза бешеный рост экзотерической (и совершенно бесцельной) спортивной статистики. Спортивные статистики с помощью своих баз данных вытягивают на свет все что нужно и не нужно знать о спортивных клубах. Неужели кого-то действительно может заинтересовать рекорд по перехватам мяча правого полузащитника в момент, когда его клуб вел с разницей в два мяча?

Выборка данных из нескольких таблиц

Все предыдущие запросы, рассмотренные нами, были получены в результате манипуляций с одной таблицей. А теперь мы приблизились к очень интересному

моменту нашего повествования. Я уже упоминал, что основное преимущество реляционных СУБД заключается в возможности ссылаться на различные объекты базы данных, что позволяет совмещать информацию и отвечать на вопросы, ответы на которые на основании данных из одной таблицы просто невозможны. В этом разделе рассказывается, как писать запросы, которые позволят это сделать.

При выборке информации из нескольких таблиц пользователь выполняет операцию, которая называется *объединение* (*join*). Такой термин здесь применяется из-за того, что результат такого запроса получен в результате объединения информации, взятой из одной таблицы, с информацией, взятой из другой таблицы.

Рассмотрим пример. Ранее в разделе "Таблицы проекта "Учет успеваемости" запрос такого рода был представлен без объяснений. Теперь пришло время и для них. В действительности это объединение является трехуровневым объединением. Таким образом, разработаем его в два этапа.

На первом этапе создадим запрос выборки результатов тестов для определенной даты:

```
mysql> SELECT student_id, date, score, type
    -> FROM event, score
    -> WHERE date = '2002-09-23'
    -> AND event.event_id = score.event_id;
+------------+------------+-------+------+
| student_id | date       | score | type |
+------------+------------+-------+------+
|          1 | 2002-09-23 |    15 | Q    |
|          2 | 2002-09-23 |    12 | Q    |
|          3 | 2002-09-23 |    11 | Q    |
|          5 | 2002-09-23 |    13 | Q    |
|          6 | 2002-09-23 |    18 | Q    |
...
```

Запрос выполняется так: сначала делается выборка записи события по заданной дате, а идентификатор события используется для поиска результатов по тому же идентификатору события. В результате выполнения запроса отображаются идентификатор учащегося, результат, дата и тип события всех записей с совпадающими записями событий.

Этот запрос отличается от всех уже записанных нами по двум важным аспектам:

- поиск производится из нескольких таблиц, и в предложении `FROM` перечислены две таблицы:
  `FROM event, score`
- в предложении `WHERE` определено, что таблицы `event` и `score` объединяются по значениям столбца `event_id`:
  `WHERE ... event.event_id = score.event_id`

Обратите внимание на синтаксис обращения к столбцу вида *`tbl_name.col_name`*. В нем обязательно указано имя таблицы. (Столбец `event_id` присутствует в обеих таблицах, поэтому здесь упоминание имени столбца без имени таблицы приводит к неоднозначности.)

Другие столбцы, участвующие в запросе (`date`, `score`, `type`), можно указывать без имени таблицы. Для этого запроса они уникальны, и их упоминание не вызывает неоднозначности. Однако я предпочитаю указывать имена таблиц в за-

просах. Это делает его более прозрачным. В полностью квалифицированном запросе мы получим:

```
SELECT score.student_id, event.date, score.score, event.type
FROM event, score
WHERE event.date = '2002-09-23'
AND event.event_id = score.event_id;
```

Первый запрос использует таблицу `event` для отображения даты в столбце `event ID`, а затем, чтобы найти соответствующие баллы в таблице `scores`, использует идентификационный номер. Вывод запроса содержит значения таблицы `student_id`. Однако очевидно, что имена были бы выразительнее. При помощи таблицы `student` мы сможем выводить идентификационные номера учеников, что представляет собой второй шаг. Это можно осуществить с помощью столбца `student_id`, присутствующего в таблицах `score` и `student`. Получим запрос следующего вида:

```
mysql> SELECT student.name, event.date, score.score, event.type
    -> FROM event, score, student
    -> WHERE event.date = '2002-09-23'
    -> AND event.event_id = score.event_id
    -> AND score.student_id = student.student_id;
+-----------+------------+-------+------+
| name      | date       | score | type |
+-----------+------------+-------+------+
| Megan     | 2002-09-23 |    15 | Q    |
| Joseph    | 2002-09-23 |    12 | Q    |
| Kyle      | 2002-09-23 |    11 | Q    |
| Abby      | 2002-09-23 |    13 | Q    |
| Nathan    | 2002-09-23 |    18 | Q    |
...
```

Этот запрос отличается от предыдущего следующим.

- В предложение `FROM` добавляется таблица `student`. Она необходима в дополнение к таблицам `event` и `score`.
- Теперь столбец `student_id` без упоминания имени таблицы приводит к неоднозначности. Необходима нотация `score.student_id` или `student.student_id`, чтобы конкретизировать столбец. (Это справедливо, даже если вы имеете привычку не именовать таблицы в запросах-объединениях.)
- В предложение `WHERE` добавляется условие соответствия записей таблиц `score` и `student` по значениям столбцов `student_id`:
  `WHERE ... score.student_id = student.student_id;`
- Запрос выводит имя учащегося, а не его идентификатор. (Конечно, при желании можно выводить оба поля сразу.)

С помощью этого запроса можно запросить любую дату и получить результаты по ней. Совсем не обязательно знать идентификационные номера учащихся или экзаменов. СУБД MySQL автоматически берет на себя всю заботу об определении соответствующих значений и их использовании для поиска соответствия между строками таблиц.

Еще одной задачей проекта учета успеваемости является учет посещаемости. Отсутствие учащихся на занятиях регистрируется по идентификатору учащихся и дате,

хранящимся в таблице absence. Для того чтобы получить имена учащихся (а не только их идентификаторы), нам необходимо объединить таблицы absence и student. Это можно сделать по значению student_id. Вот запрос, выводящий идентификатор учащихся и их имена вместе с суммой занятий, пропущенных учащимися:

```
mysql> SELECT student.student_id, student.name,
    -> COUNT(absence.date) as absences
    -> FROM student, absence
    -> WHERE student.student_id = absence.student_id
    -> GROUP BY student.student_id;
+------------+-------+----------+
| student_id | name  | absences |
+------------+-------+----------+
|          3 | Kyle  |        1 |
|          5 | Abby  |        1 |
|         10 | Peter |        2 |
|         17 | Will  |        1 |
|         20 | Avery |        1 |
+------------+-------+----------+
```

Совет

Несмотря на то что здесь в предложении GROUP BY указано имя таблицы, это совсем не обязательно. Предложение GROUP BY имеет отношение к столбцам, указанным в списке выборки (в первых двух строках запроса). А там указан только один столбец student_id, поэтому СУБД MySQL "знает", какая таблица имеется в виду.

Этот запрос выдает отличный результат тогда, когда мы стремимся узнать имена учащихся, которые имеют пропуски занятий. Но если такой список предоставить педагогическому совету, то может последовать совершенно естественный вопрос: "А как же остальные учащиеся? Нам необходимо оценить результаты и оставшихся учеников". Но это совсем другой вопрос. Это значит, что необходимо получить количество пропущенных занятий даже для учащихся, которые пропусков не имеют. Этот запрос отличается от предыдущего.

Для того чтобы на него ответить, воспользуемся конструкцией LEFT JOIN в предложении WHERE. Она указывает СУБД MySQL сделать выборку всех строк из таблицы, указанной в предложении слева (слева от слов LEFT JOIN). Назвав сначала таблицу student, мы получим перечень всех учащихся, даже тех, кто не представлен в таблице absence. Теперь запрос будет выглядеть следующим образом:

```
mysql> SELECT student.student_id, student.name,
    -> COUNT(absence.date) as absences
    -> FROM student LEFT JOIN absence
    -> ON student.student_id = absence.student_id
    -> GROUP BY student.student_id;
+------------+-----------+----------+
| student_id | name      | absences |
+------------+-----------+----------+
|          1 | Megan     |        0 |
|          2 | Joseph    |        0 |
|          3 | Kyle      |        1 |
|          4 | Katie     |        0 |
|          5 | Abby      |        1 |
|          6 | Nathan    |        0 |
|          7 | Liesl     |        0 |
...
```

Выше, в разделе “Получение итоговых результатов”, был показан пример запроса, который выдавал цифровые данные на основе информации, содержащейся в таблице `score`. Результат этого запроса содержал идентификатор события, но не мог содержать дату получения результата теста или его тип. Теперь мы уже знаем, каким образом можно объединить таблицы `score` и `event`, чтобы получить даты и типы результатов.

```
mysql> SELECT
    -> event.date,event.type,
    -> MIN(score.score) AS minimum,
    -> MAX(score.score) AS maximum,
    -> MAX(score.score)-MIN(score.score)+1 AS range,
    -> SUM(score.score) AS total,
    -> AVG(score.score) AS average,
    -> COUNT(score.score) AS count
    -> FROM score, event
    -> WHERE score.event_id = event.event_id
    -> GROUP BY event.date;
+------------+------+---------+---------+-------+-------+---------+-------+
| date       | type | minimum | maximum | range | total | average | count |
+------------+------+---------+---------+-------+-------+---------+-------+
| 2002-09-03 | Q    |       9 |      20 |    12 |   439 | 15.1379 |    29 |
| 2002-09-06 | Q    |       8 |      19 |    12 |   425 | 14.1667 |    30 |
| 2002-09-09 | T    |      60 |      97 |    38 |  2425 | 78.2258 |    31 |
| 2002-09-16 | Q    |       7 |      20 |    14 |   379 | 14.0370 |    27 |
| 2002-09-23 | Q    |       8 |      20 |    13 |   383 | 14.1852 |    27 |
| 2002-10-01 | T    |      62 |     100 |    39 |  2325 | 80.1724 |    29 |
+------------+------+---------+---------+-------+-------+---------+-------+
```

Для получения суммарных значений на основании данных из столбцов различных таблиц можно также воспользоваться агрегатными функциями `COUNT()` и `AVG()`. Следующий запрос определяет число результатов и среднеарифметическое для всех комбинаций дат событий и пола учащихся:

```
mysql> SELECT event.date, student.sex,
    -> COUNT(score.score) AS count, AVG(score.score) AS average
    -> FROM event, score, student
    -> WHERE event.event_id = score.event_id
    -> AND score.student_id = student.student_id
    -> GROUP BY event.date, student.sex;
+------------+-----+-------+---------+
| date       | sex | count | average |
+------------+-----+-------+---------+
| 2002-09-03 | F   |    14 | 14.6429 |
| 2002-09-03 | M   |    15 | 15.6000 |
| 2002-09-06 | F   |    14 | 14.7143 |
| 2002-09-06 | M   |    16 | 13.6875 |
| 2002-09-09 | F   |    15 | 77.4000 |
| 2002-09-09 | M   |    16 | 79.0000 |
| 2002-09-16 | F   |    13 | 15.3077 |
| 2002-09-16 | M   |    14 | 12.8571 |
| 2002-09-23 | F   |    12 | 14.0833 |
| 2002-09-23 | M   |    15 | 14.2667 |
| 2002-10-01 | F   |    14 | 77.7857 |
| 2002-10-01 | M   |    15 | 82.4000 |
+------------+-----+-------+---------+
```

Аналогичный запрос можно сделать для проекта учета успеваемости. Например, для вычисления суммарного результата каждого учащегося в конце периода обучения:

```
SELECT student.student_id, student.name,
SUM(score.score) AS total, COUNT(score.score) AS n
FROM event, score, student
WHERE event.event_id = score.event_id
AND score.student_id = student.student_id
GROUP BY score.student_id
ORDER BY total;
```

Объединять можно не только различные таблицы, но и таблицу саму с собой. Например, можно определить, есть ли президенты, которые родились в одном и том же месте:

```
mysql> SELECT p1.last_name, p1.first_name, p1.city, p1.state
    -> FROM president AS p1, president AS p2
    -> WHERE p1.city = p2.city AND p1.state = p2.state
    -> AND (p1.last_name != p2.last_name OR p1.first_name !=
p2.first_name)
    -> ORDER BY state, city, last_name;
+-----------+-------------+-----------+-------+
| last_name | first_name  | city      | state |
+-----------+-------------+-----------+-------+
| Adams     | John Quincy | Braintree | MA    |
| Adams     | John        | Braintree | MA    |
+-----------+-------------+-----------+-------+
```

Этот запрос имеет две особенности.

- Требуется ссылка на два экземпляра одной и той же таблицы. Для этого создаются два экземпляра одной и той же таблицы с различными псевдонимами (`p1`, `p2`).
- Запись каждого президента соответствует самой себе. Но они не нужны в отклике запроса. Вторая строка предложения `WHERE` позволяет избежать этого.

Аналогичный запрос делает выборку президентов, родившихся в один и тот же день. При этом нельзя сравнивать даты рождения непосредственно из таблицы. Для сравнения дня и месяца рождения воспользуемся функциями `MONTH()` и `DAYOFMONTH()`.

```
mysql> SELECT p1.last_name, p1.first_name, p1.birth
    -> FROM president AS p1, president AS p2
    -> WHERE MONTH(p1.birth) = MONTH(p2.birth)
    -> AND DAYOFMONTH(p1.birth) = DAYOFMONTH(p2.birth)
    -> AND (p1.last_name != p2.last_name OR p1.first_name !=
p2.first_name)
    -> ORDER BY p1.last_name;
+-----------+------------+------------+
| last_name | first_name | birth      |
+-----------+------------+------------+
| Harding   | Warren G.  | 1865-11-02 |
| Polk      | James K.   | 1795-11-02 |
+-----------+------------+------------+
```

Запрос можно упростить, если вместо комбинации функций MONTH() и DAYOFMONTH() воспользоваться функцией DAYOFYEAR(), но это даст некорректные результаты при сравнении дат високосных и невисокосных годов.

Вот таким образом объединение позволяет манипулировать информацией из таблиц, имеющих некую смысловую взаимосвязь, но логику этих связей знает только сам создатель запроса. СУБД MySQL не "знает", есть ли вообще что-то общее между объединенными таблицами. Например, можно объединить таблицы event и president для того, чтобы определить, проводились ли какие-либо викторины или тесты на тему дней рождения президентов.

```
mysql> SELECT president.last_name, president.first_name,
    -> president.birth, event.type
    -> FROM president, event
    -> WHERE MONTH(president.birth) = MONTH(event.date)
    -> AND DAYOFMONTH(president.birth) = DAYOFMONTH(event.date);
+-----------+------------+------------+------+
| last_name | first_name | birth      | type |
+-----------+------------+------------+------+
| Carter    | James E.   | 1924-10-01 | T    |
+-----------+------------+------------+------+
```

Оказывается, такое возможно. Ну и что? А то, что СУБД MySQL может выдавать результаты независимо от того, имеют они смысл или абсолютно бессмысленны. Результаты запроса необязательно имеют пользу и какой-то смысл только потому, что они получены с помощью компьютера. К счастью, человек все еще должен думать и понимать, что он делает.

Удаление и модификация существующих записей

Иногда требуется избавиться от определенных записей или изменить их содержимое. Для этой цели предназначены операторы DELETE и UPDATE.

Оператор DELETE имеет следующий синтаксис:

```
DELETE FROM tbl_name
WHERE which records to delete;
```

Предложение WHERE определяет, какая именно запись удаляется. Она может опускаться, но в этом случае будут удалены все записи в таблице! Это значит, что чем проще оператор DELETE, тем он "опаснее":

```
DELETE FROM tbl_name;
```

Этот запрос стирает все содержимое таблицы. *Будьте бдительны!*

Чтобы при удалении определенных записей указать их, воспользуйтесь предложением WHERE. Совсем так, как в операторе SELECT. Например, для того чтобы удалить из таблицы president всех президентов, родившихся в штате Огайо, выведите следующую команду:

```
mysql> DELETE FROM president WHERE state='OH';
Query OK, 7 rows affected (0.00 sec)
```

Перед использованием предложения WHERE в операторе DELETE советую проверить его работу на операторе SELECT. Не мешает убедиться, какие записи будут удалены оператором DELETE. Предположим, что нужно удалить запись о президенте Теодоре Рузвельте. Выполнит ли следующий запрос эту задачу?

```
DELETE FROM president WHERE last_name='Roosevelt';
```

Да, он выполнит в том плане, что удалит запись, которую вы задумали удалить. Нет, поскольку он также удалит запись о Франклине Рузвельте. Безопаснее предварительно использовать предложение WHERE с оператором SELECT:

```
mysql> SELECT last_name, first_name FROM president
    -> WHERE last_name='Roosevelt';
+-----------+-------------+
| last_name | first_name  |
+-----------+-------------+
| Roosevelt | Theodore    |
| Roosevelt | Franklin D. |
+-----------+-------------+
```

Этот результат показывает, что требуется более серьезная детализация запроса:

```
mysql> SELECT last_name, first_name FROM president
    -> WHERE last_name='Roosevelt' AND first_name='Theodore';
+-----------+------------+
| last_name | first_name |
+-----------+------------+
| Roosevelt | Theodore   |
+-----------+------------+
```

Теперь нам известно правильное предложение WHERE. Таким образом, правильный оператор будет иметь вид

```
mysql> DELETE FROM president
    -> WHERE last_name='Roosevelt' AND first_name='Theodore';
```

Правда, операция удаления записи требует много предварительной работы? Семь раз отмерь, один раз отрежь! (На этом не стоит экономить. Сэкономить можно на копировании, вставке или методах редактирования строк. Более подробно об этом читайте в разделе “Как работать с mysql”.)

Для модификации уже существующих записей можно воспользоваться оператором UPDATE, который имеет следующий синтаксис:

```
UPDATE tbl_name
SET which columns to change
WHERE which records to update;
```

Здесь предложение WHERE ведет себя так же, как и предложение WHERE для оператора DELETE. Оно может опускаться, но в этом случае *изменения будут сделаны во всех записях таблицы*! Вот запрос, который изменит имена всех учащихся из таблицы student на имя “George”:

```
mysql> UPDATE student SET name='George';
```

Очевидно, что необходимо быть предельно осторожными с такими запросами.

К записям, которые подвергаются модификации, необходимо подходить более детально. Предположим, что недавно в “Историческую Лигу” вступил новый член. При этом были заполнены только отдельные столбцы:

```
mysql> INSERT INTO member (last_name,first_name)
    -> VALUES('York','Jerome');
```

После этого стало понятно, что была пропущена дата истечения срока членства. Это легко исправить с помощью оператора UPDATE, который содержит соответствующее предложение WHERE, чтобы определить, какую запись надо изменить:

```
mysql> UPDATE member
    -> SET expiration='2001-7-20'
    -> WHERE last_name='York' AND first_name='Jerome';
```

Можно модифицировать несколько столбцов одновременно. Этот запрос изменит адреса электронной и обычной почты одновременно:

```
mysql> UPDATE member
    -> SET email='jeromey@aol.com',street='123 Elm St',city='Anytown',
    -> state='NY',zip='01003'
    -> WHERE last_name='York' AND first_name='Jerome';
```

Можно присвоить значениям столбцов пустое значение (если столбец имеет такую возможность). Допустим, Джером решил заплатить большой членский взнос, что позволяет ему стать пожизненным членом. Это можно зафиксировать, установив дату истечения срока полномочий в NULL (никогда не наступает):

```
mysql> UPDATE member
    -> SET expiration=NULL
    -> WHERE last_name='York' AND first_name='Jerome';
```

Для оператора UPDATE, так же как и для оператора DELETE, верно утверждение, что нужно проверять правильность предложения WHERE с помощью оператора SELECT. Если ваш критерий выборки слишком широк, то модифицируется слишком много записей, если слишком узок — слишком мало записей.

Тем читателям, которые в процессе ознакомления с этими разделами выполняли все описанные шаги на тестовой базе данных sampdb, перед тем, как перейти к следующему разделу, рекомендуем отменить все изменения, сделанные в таблицах базы данных. Проще всего это сделать, перезагрузив все таблицы в соответствии с рекомендациями, данными в разделе “Дополнение таблиц”.

Как работать с mysql

В этом разделе описываются приемы эффективной работы с клиентской программой mysql. Показано, как можно просто подключиться к серверу и вводить запросы, копируя их из одного черновика, а также как изменить приглашение, если вам не нравится стандартное.

Упрощение процесса подключения

Очевидно, что при вызове mysql необходимо указать такие параметры подключения, как имя узла, имя пользователя или пароль. Эта процедура требует длительного ввода и очень быстро утомляет. Ниже перечислено несколько способов, упрощающих эту процедуру.

- Храните параметры подключения в конфигурационном файле.
- Воспользуйтесь возможностью повторения команд из журнала регистрации.
- С помощью псевдонима или сценария оболочки задайте ярлык командной строки mysql.

Использование конфигурационных файлов

Начиная с версии 3.22 появилась возможность сохранять параметры соединения в конфигурационном файле. В этом случае нет необходимости вводить

параметры каждый раз при запуске `mysql`. Эти параметры используются также и в других клиентских приложениях, таких как, например, `mysqlimport`

Конфигурационный файл упростит процесс ввода команд не только для клиента `mysql`, но и для других программ.

В ОС UNIX создадим файл `~/.my.cnf` (т.е. файл под именем `my.cnf` в вашем корневом каталоге). В ОС Windows создадим файл с названием `my.cnf` в корневой папке диска C или файл `my.ini` в папке `System` (`C:\my.cnf` или `%SYSTEM%\my.ini`). Конфигурационный файл является текстовым файлом, поэтому вы можете создавать его и редактировать в любом текстовом редакторе. Его содержание должно быть похоже на следующее:

```
[client]
host=server_host
user=your_name
password=your_pass
```

Строка `[client]` идентифицирует начало группы параметров данного клиента. Каждая последующая строка считывается СУБД MySQL как значение параметров клиентской программы. Завершается информация концом файла или началом группы параметров другого клиента. Здесь нужно заменить *`server_host`*, *`your_name`* и *`your_pass`* именем вашего сервера, вашим именем и паролем. Например, мой `.my.cnf` выглядит следующим образом:

```
[client]
host=cobra.snake.net
user=sampadm
password=secret
```

Здесь является обязательной только строка `[client]`. Строки, определяющие значения параметров, могут быть опущены; можно указывать только те, которые необходимы. Например, если ваше имя пользователя в базе данных совпадает с именем пользователя ОС UNIX, то его можно не указывать и не включать в строку `user`.

После создания файла `my.cnf` рекомендуется ограничить к нему доступ. Делается это следующим образом:

```
% chmod 600 .my.cnf
% chmod u=rw,go-rwx .my.cnf
```

Более подробную информацию о конфигурационных файлах можно найти в приложении Д, “Программы MySQL”.

Использование журнала регистрации команд

Такие оболочки, как `csh`, `tcsh` и `bash`, запоминают отработанные команды в журналах регистрации. Это позволяет вызывать команды непосредственно из журнала регистрации без их повторного ввода. Например, повторить уже один раз введенную команду `mysql` можно следующим образом:

```
% !my
```

Символ “!” сообщает оболочке, что необходимо найти последнюю команду, которая начинается с “`my`”, и повторить ее так, как если бы ее ввели с клавиатуры. Некоторые оболочки позволяют просматривать историю вверх и вниз с помощью клавиш <↑> и <↓> (или комбинаций клавиш <Ctrl+P> и <Ctrl+N>). Таким образом выбирается нужная команда, после чего для ее активизации следует нажать клавишу

<Enter>. Оболочки tcsh и bash имеют такую возможность. Для того чтобы узнать больше о возможностях оболочки, просмотрите документацию и по ней.

Псевдонимы и сценарии

Присвоение псевдонима позволит вызвать большую командную строку с помощью небольшой команды. Например, создадим в оболочках csh и tcsh псевдоним sampdb, используя следующую командную строку:

```
alias sampdb 'mysql -h cobra.snake.net -p -u sampadm sampdb'
```

Оболочка bash имеет другой синтаксис для команды создания псевдонима:

```
alias sampdb='mysql -h cobra.snake.net -p -u sampadm sampdb'
```

Создание псевдонима уравняет в функциональности такие две команды:

```
% sampdb
% mysql -h cobra.snake.net -p -u sampadm sampdb
```

Совершенно очевидно, что первую командную строку значительно проще напечатать, чем вторую. Для того чтобы псевдоним активизировался при каждой регистрации, добавьте команду alias в ваш стартовый файл (например, .cshrc для оболочки csh или .bash_profile для оболочки bash).

В системе Windows можно создать ярлык, который указывает на программу mysql, а затем, чтобы указать параметры соединения, отредактировать свойства ярлыка.

Еще одной формой ярлыка является сценарий, который будет выполняться программой mysql. В ОС UNIX файл сценария, являющийся эквивалентом псевдонима sampdb, выглядит следующим образом:

```
#! /bin/sh
exec mysql -h cobra.snake.net -p -u sampadm sampdb
```

Для того чтобы этот сценарий, назовем его sampdb, заработал, необходимо сделать его исполнимым (используя команду chmod +x sampdb). Теперь можно, напечатав sampdb, запустить клиентское приложение mysql.

В ОС Windows для этой цели используются пакетные файлы. Назовем такой файл sampdb.bat и введем туда такую строку:

```
mysql -h cobra.snake.net -p -u sampadm sampdb
```

Этот файл можно запустить из консоли DOS или двойным щелчком на соответствующей пиктограмме.

Чтобы подключиться к нескольким узлам и иметь доступ к нескольким базам данных, можно создать сразу несколько псевдонимов или сценариев, каждый из которых будет активизировать mysql с различными параметрами.

Упрощение процесса создания запросов

Программа mysql — это очень удобный инструмент взаимодействия с базой данных, но ее интерфейс больше всего подходит для создания коротких запросов в одну строку. Программа mysql сама по себе ничего не "знает" о длинных запросах, занимающих несколько строк. Такие запросы очень трудно создавать. Очень огорчительно после длительного ввода запроса обнаружить, что он содержит синтаксическую ошибку и его надо переделать.

Вот несколько приемов, которые позволят избежать ненужного ввода и перепечатывания.

- Для ввода строк в `mysql` пользуйтесь текстовым редактором.
- Не забывайте о возможностях копирования и вставки.
- Работайте с `mysql` в пакетном режиме.

Для ввода строк в `mysql` пользуйтесь текстовым редактором

Программа `mysql` имеет встроенную библиотеку для редактирования. В процессе ввода командных строк ими можно манипулировать, вызывать уже введенные строки, повторять их и модифицировать. Это очень удобно. Если при вводе команды вы допустили ошибку, можно вернуться назад и в переделах строки внести исправление. Аналогичным образом можно повторить запрос, который оказался ошибочным. (Эта задача существенно упрощается, если запрос был введен одной строкой.)

В табл. 1.4 приведены наиболее часто используемые комбинации клавиш для работы с текстовым редактором `mysql`. Но кроме этих существует еще множество команд. С ними вы можете ознакомиться в разделе команд редактирования в справочнике `bash`, который доступен на Web-узле GNU проекта (`http://www.gnu.org/manual/`).

Таблица 1.4. Команды редактора программы `mysql`

Комбинации клавиш	Назначение
<↑> или <Ctrl+P>	Отобразить предыдущую строку
<↓> или <Ctrl+N>	Отобразить следующую строку
<←> или <Ctrl+B>	Передвинуть курсор влево (назад)
<→> или <Ctrl+F>	Передвинуть курсор вправо (вперед)
<Escape>, <Ctrl+ B>	Передвинуть курсор назад на одно слово
<Escape>, <Ctrl+ F>	Передвинуть курсор вперед на одно слово
<Ctrl+ A>	Передвинуть курсор в начало строки
<Ctrl+ E>	Передвинуть курсор в конец строки
<Ctrl+ D>	Удалить символ, находящийся под курсором
<Delete>	Удалить символ, находящийся слева от курсора
<Escape D>	Удалить слово
<Escape>, <Backspace>	Удалить слово, находящееся слева от курсора
<Ctrl+K>	Удалить все символы от курсора до конца строки
<Ctrl+_>	Отменить последнее действие (может быть повторено)

Вот пример, который отлично иллюстрирует простоту редактирования вводимых строк. Предположим, что была введена следующая ошибочная командная строка:

```
mysql> SHOW COLUMNS FROM persident;
```

Допустим, вы заметили ошибку в слове president, до того как нажали клавишу <Enter>. С помощью комбинации клавиш <Ctrl+B> или клавиши <←> переместите

курсор до символа "s". Затем дважды нажмите клавишу <Delete>, чтобы удалить "er", и введите "e". Нажмите клавишу <Enter>. Теперь допустим, что ошибка была обнаружена после нажатия клавиши <Enter>. В этом случае после диагностического сообщения нажмите клавишу <↑> или комбинацию клавиш <Ctrl+P>. После того как ошибочная строка будет выведена на экран, внесите изменения в строку вышеуказанным способом.

Возможность редактирования вводимой командной строки отсутствует в обычной версии `mysql` для ОС Windows. (В системе Windows NT `mysql` поддерживается переход между командами с помощью стрелок, но не более того.) Чтобы иметь возможность редактировать команды, используйте программу `mysqlc`, которая идентична программе `mysql`, но работает с библиотеками `Cygnus`, поддерживающими редактирование строк.

Чтобы проверить, правильно ли установлена `mysqlc`, обратитесь к приложению Д, "Программы MySQL".

Пользуйтесь возможностями копирования и вставки

Работая в оконной среде, можно сохранять и воспроизводить нужные запросы или их отдельные части с помощью обычных операций копирования и вставки. Вот примерная последовательность действий.

1. Вызовите `mysql` в окне терминала или в консоли DOS.
2. Откройте в окне текстового редактора файл, содержащий запросы. (Например, `BBEdit` — в операционной системе Mac OS и `vi` — в окне `xterm` под X Window в ОС UNIX).
3. Для того чтобы вызвать запрос, хранящийся в файле, выделите и скопируйте его. Затем перейдите в окно `Telnet` или консоль DOS и вставьте запрос в `mysql`.

Как видите, процедура достаточно громоздкая в печатном виде, но это самый легкий путь быстрого ввода запросов без изнурительного перепечатывания.

Этот метод также позволяет не только редактировать запросы в окне текстового редактора, но там же создавать новые запросы, просто копируя и вставляя фрагменты старых запросов. Предположим, вы часто выбираете строки из определенной таблицы, но при этом их необходимо сортировать различными способами. Для этого можно хранить разные предложения `ORDER BY` и по необходимости просто их копировать.

Операцией копирования и вставки можно пользоваться и в обратном направлении (из окна терминала — в файл запросов). В процессе работы с `mysql` все команды сохраняются в файле `.mysql_history`, находящемся в корневом каталоге текущего клиента. Любой ранее введенный запрос можно повторить из файла `.mysql_history`, скопировав его из этого файла в файл запроса.

Работайте с `mysql` в пакетном режиме

Совсем не обязательно работать с программой `mysql` в интерактивном режиме. Программа `mysql` имеет возможность считывать командные строки из пакетного файла в неинтерактивном (пакетном) режиме. Это удобно для работы с за-

просами, которые нужно запускать с определенной периодичностью, потому что затруднительно вводить командную строку всякий раз, когда требуется запустить этот запрос. Значительно проще записать их в файл один раз, а потом периодически обращаться к нему.

Предположим, что существует запрос выборки членов "Исторической Лиги" по научным интересам. Например, для выборки членов, интересующихся временами "Великой Депрессии", необходимо написать следующий запрос:

```
SELECT last_name, first_name, email, interests FROM member
WHERE interests LIKE '%depression%'
ORDER BY last_name, first_name;
```

Сохраним этот запрос в файле `interests.sql` и попробуем запустить `mysql` из командной строки:

```
% mysql sampdb < interests.sql
```

По умолчанию `mysql` при работе в пакетном режиме осуществляет вывод без разделения столбцов табуляцией. Для получения вывода с разделением столбцов табуляцией необходимо указать ключ `–t`:

```
% mysql -t sampdb < interests.sql
```

Для сохранения результата работы запроса перенаправьте вывод в файл:

```
% mysql -t sampdb < interests.sql > output_file
```

Этот же запрос можно использовать для выборки членов "Исторической Лиги", интересующихся периодом правления президента Джефферсона. Замените в предложении `WHERE depression` на `Jefferson`. Затем запустите `mysql` повторно. Такой метод хорош и при редком обращении к запросу. При более частом обращении подойдет другой метод. Для того чтобы запрос стал более гибким, создайте его в виде параметризированного сценария. Параметр будет передавать переменное значение для предложения `WHERE`. Посмотрим, как данный метод работает.

```
#! /bin/sh
# interests.sh - поиск членов "Лиги" с определенными интересами
if [ $# -ne 1 ]; then echo 'Введите одно ключевое слово'; exit; fi
mysql -t sampdb <<QUERY_INPUT
SELECT last_name, first_name, email, interests FROM member
WHERE interests LIKE '%$1%'
ORDER BY last_name, first_name;
QUERY_INPUT
```

Третья строка проверяет, что в командной строке было введено одно ключевое слово. Выводится короткая строка, в противном случае работа завершается. Все, что находится между первым `<<QUERY_INPUT` и последним `QUERY_INPUT`, является вводом в `mysql`. Оболочка заменяет ссылку `$1` на значение, введенное в командной строке. (В сценариях оболочки `$1`, `$2` и т.д. обозначают аргументы команды.)

Перед тем как запустить запрос, его необходимо сделать исполнимым:

```
% chmod +x interests.sh
```

Теперь необходимость в редактировании запроса отпала. То, что вам требуется от запроса, можно определить командной строкой:

```
% interests.sh depression
% interests.sh Jefferson
```

Сценарий `interests.sh` вы найдете в папке `misc` дистрибутива `sampdb`. Его эквивалент, пакетный файл Windows `interests.bat`, находится там же.

Изменение приглашения `mysql`

В MySQL версии 4.0.2 вы можете изменить приглашение, если вам не нравится стандартное. Например, чтобы вписать имя текущей базы данных в приглашение, напишите команду `PROMPT`, а потом выберите разные базы данных, чтобы увидеть, насколько приглашение соответствует выбору:

```
% mysql
mysql> PROMPT \d>\_
PROMPT set to '\d>\_'
(none)> USE sampdb;
Database changed
sampdb> USE test;
Database changed
test>
```

За ключевым словом PROMPT следует желаемая строка запроса. Последовательности, которые начинаются с обратной косой черты, — это специальные параметры приглашения. Последовательности \d и _ объявляют текущее имя базы данных и пробел. Полный перечень параметров приведен в приложении Д, "Программы MySQL".

Что дальше?

Теперь нам кое-что известно о работе СУБД MySQL. Мы знаем, как создать базу данных и ее таблицы, умеем заносить данные в эти таблицы, производить различные выборки, модификации и удаления. Но это только начало. Это можно заметить по состоянию базы данных с примером. Мы только что создали базу данных и ее таблицы, занесли туда начальные данные, а также рассмотрели методы написания запросов, без которых невозможно обойтись, работая с базой данных.

К примеру, сейчас у нас есть на вооружении достаточно удобные способы ввода запросов, которые необходимы для ввода оценок в проекте учета успеваемости или добавления записей о новых членах "Исторической Лиги". Но, к сожалению, нет простых методов редактирования существующих записей. Неизвестно также, как печатать или отображать в интерактивном режиме список членов "Исторической Лиги". Эти и другие проблемы будут рассмотрены в последующих главах, в частности в главах 7 и 8.

То, к изучению какой главы вы перейдете сейчас, зависит от того, чем вы интересуетесь. Если вы хотите продолжить работу, начатую в проекте "Историческая Лига", или интересуетесь учетом успеваемости, то рекомендую перейти к части II, в которой описывается программирование в СУБД MySQL. Часть III сможет удовлетворить тех, кто интересуется проблемами администрирования. Но я рекомендую сначала набраться опыта в работе с

СУБД MySQL, прочитав до конца главы части I. В них описывается, как обрабатываются данные из баз данных, глубже проработан синтаксис и использование операторов SQL, приведены методы оптимизации запросов. Хорошие знания по этой теме станут фундаментом для дальнейшего использования СУБД MySQL независимо от контекста ее применения — работаете ли вы с `mysql`, пишете ли свои собственные программы или выполняете функции администратора.