Вводный курс для разработчиков и администраторов БД

Изучаем SQL

Символ®

O'REILLY®

Алан Бьюли

Learning SQL

Alan Beaulieu

O'REILLY®

Изучаем SQL

Алан Бьюли

Санкт-Петербург — Москва
2007

Алан Бьюли

Изучаем SQL

Перевод Н. Шатохиной

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Научный редактор	*В. Коренев*
Редактор	*А. Кузнецов*
Корректор	*Н. Ткачева*
Верстка	*Д. Орлова*

Бьюли А.
Изучаем SQL. – Пер. с англ. – СПб: Символ-Плюс, 2007. – 312 с., ил.
ISBN-13: 978-5-93286-051-9
ISBN-10: 5-93286-051-0

Книга Алана Бьюли, эксперта по языку SQL, – прекрасный учебник для тех, кто еще не знает, но хочет освоить этот язык. Книга не только позволит приобрести начальные знания, но и расскажет о наиболее часто употребляемых мощных средствах языка SQL, используемых опытными программистами.

Многие книги, посвященные SQL, грешат скучным изложением основ. Здесь же автор в стиле живого рассказа обсуждает SQL-выражения и блоки, различные типы условий, показывает, как посредством соединения таблиц создавать запросы к нескольким таблицам, рассматривает наборы данных и как они могут взаимодействовать в запросах, демонстрирует встроенные и агрегатные функции, показывает, как и где используются подзапросы. Подробно описаны различные типы соединений таблиц, применение условной логики, работа с транзакциями, индексы и ограничения.

Поскольку лучший способ изучения SQL – это практика, автор создает учебную базу данных MySQL и приводит множество вариантов реальных запросов, охватывающих весь теоретический материал. При таком подходе не научиться просто невозможно. Примеры кода можно использовать в своих программах и документации. Книга предназначена разработчикам приложений БД, администраторам БД и тем, кто создает отчеты.

ISBN-13: 978-5-93286-051-9
ISBN-10: 5-93286-051-0
ISBN 0-596-00727-2 (англ)

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, edit@symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.
Подписано в печать 13.02.2007. Формат 70x100 1/16 . Печать офсетная.
Объем 19,5 печ. л. Тираж 2000 экз. Заказ N
Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

Оглавление

Предисловие

Языки программирования постоянно появляются и исчезают, и очень немногие из современных языков имеют более чем 10-летнюю историю. Среди долгожителей можно назвать КОБОЛ, который до сих пор довольно широко используется в мэйнфреймовых средах, и C, по-прежнему весьма популярный при разработке операционных систем, серверов и встроенных систем. В области баз данных это SQL, корни которого уходят в далекие 1970-е.

SQL – язык для формирования, манипулирования и извлечения данных из реляционной БД. Одна из причин популярности реляционных БД в том, что, будучи правильно спроектированными, они могут оперировать гигантскими объемами данных. В работе с большими наборами данных SQL напоминает современную цифровую фотокамеру с мощным объективом: он позволяет просматривать большие объемы данных или перейти к «крупному плану», т. е. сфокусироваться на отдельных строках (подвластно и все, что между этими крайностями). Другие СУБД дают сбой при мощных нагрузках, потому что их фокус слишком узок (увеличительные линзы достигают своего максимума). Именно по этой причине все попытки низвергнуть реляционные БД и SQL оканчиваются неудачей. Поэтому, даже несмотря на то, что SQL – старый язык, похоже, его ждет еще очень долгая жизнь и блестящее будущее.

Зачем изучать SQL?

Если вы собираетесь работать с реляционными БД – писать приложения, или выполнять задачи по администрированию, или формировать отчеты, – вам понадобится знать, как взаимодействовать с данными БД. Даже при использовании инструмента, генерирующего SQL (например, инструмента создания отчетов), могут возникнуть ситуации, в которых понадобится обойти автоматические возможности и создавать собственные SQL-выражения.

Дополнительное преимущество изучения SQL в том, что вы быстрее рассмотрите и поймете структуры данных, применяемые для хранения информации о вашей организации. Почувствовав себя уверенно со своей БД, вы сможете вносить предложения по изменению или дополнению ее схемы.

Почему именно эта книга?

Язык SQL включает несколько категорий. Выражения, с помощью которых создаются объекты БД (таблицы, индексы, ограничения и т. д.), называют *SQL-выражениями управления схемой данных* (*schema statements*). Выражения, предназначенные для создания, манипулирования и извлечения данных, хранящихся в БД, называют *SQL-выражениями для работы с данными* (*data statements*). Если вы администратор, то будете использовать и те и другие SQL-выражения. Если вы программист или составитель отчетов, то сможете (или вам будет *позволено*) использовать только SQL-выражения для работы с данными. Хотя в этой книге встречается много SQL-выражений управления схемой, основное внимание в ней уделено возможностям программирования.

Поскольку команд немного, SQL-выражения для работы с данными кажутся простыми. По-моему, многие из имеющихся книг по SQL только усиливают это впечатление, давая лишь поверхностный обзор того, что можно делать с помощью этого языка. Однако если вы собираетесь работать с SQL, вам следует полностью понимать все его возможности и то, как сочетать их для получения мощных результатов. На мой взгляд, эта книга – единственная, где язык SQL описан подробно, и при этом она не является «кирпичом» (вам знакомы эти «полные руководства» по 1250 страниц, пылящиеся у народа на полках).

Хотя примеры из книги подходят для MySQL, Oracle Database и SQL Server, мне пришлось отобрать один из этих продуктов, чтобы разместить БД для выполнения примеров и форматировать результирующие наборы, возвращаемые примерами запросов. Из этих трех я выбрал MySQL, потому что он свободно доступен, его легко установить и просто администрировать. Читателей, использующих другой сервер, прошу скачать и установить MySQL и загрузить предлагаемую БД, чтобы иметь возможность выполнять примеры и экспериментировать с данными.

Структура книги

Книга содержит 13 глав и 4 приложения:

- В главе 1 «Немного истории» рассматривается история компьютерных БД, включая возникновение реляционной модели и языка SQL.
- В главе 2 «Создание и заполнение базы данных» показывается, как создавать БД MySQL и таблицы, используемые в примерах к книге, и как заполнять таблицы данными.
- Глава 3 «Азбука запросов» знакомит с выражением `select` и представляет наиболее распространенные блоки (clauses): `select`, `from`, `where`.

- Глава 4 «Фильтрация» представляет разные типы условий, которые могут использоваться в блоке `where` выражений `select`, `update` и `delete`.
- В главе 5 «Запрос к нескольким таблицам» показывается, как запросы могут работать с несколькими таблицами посредством соединений таблиц.
- Глава 6 «Работа с множествами» – все о множествах данных и о том, как они могут взаимодействовать внутри запросов.
- Глава 7 «Создание, преобразование и работа с данными» представляет несколько встроенных функций, используемых для манипулирования или преобразования данных.
- В главе 8 «Группировка и агрегаты» показывается, как можно агрегировать данные.
- Глава 9 «Подзапросы» представляет подзапрос (мой любимый прием) и показывает, как применяются подзапросы.
- Глава 10 «И снова соединения» продолжает рассматривать различные типы соединений таблиц.
- В главе 11 «Условная логика» рассматривается использование условной логики (т. е. `if-then-else`) в выражениях `select`, `insert`, `update` и `delete`.
- Глава 12 «Транзакции» знакомит с транзакциями и их использованием.
- В главе 13 «Индексы и ограничения» исследуются индексы и ограничения.
- Приложение A «ER-диаграмма примера базы данных» содержит схему базы данных, используемой для всех примеров книги.
- Приложение B «MySQL-расширения языка SQL» представляет некоторые интересные возможности реализации SQL – MySQL, не входящие в стандарт ANSI.
- Приложение C «Решения к упражнениям» содержит решения упражнений, приводимых в главах.
- Приложение D «Дополнительные источники» подсказывает, куда можно обратиться, чтобы получить более глубокие навыки.

Условные обозначения, используемые в книге

В книге используются следующие типографские обозначения:

Курсив

Используется для имен файлов, имен каталогов и URL-адресов. Также используется для выделения и при первом упоминании технического термина.

Моноширинный шрифт

Используется для примеров кода и обозначения ключевых слов SQL в тексте.

Моноширинный курсив

Используется для обозначения пользовательских терминов.

ВЕРХНИЙ РЕГИСТР

Используется для обозначения ключевых слов SQL в примерах кода.

Моноширинный полужирный шрифт

Выделяет ввод пользователя в примерах с интерактивным взаимодействием. Также выделяет элементы кода, на которые следует обратить особое внимание.

Так выделяются советы, рекомендации или общие примечания. Например, с помощью примечаний я обращаю ваше внимание на полезные новые возможности Oracle9*i*.

Обозначает предупреждение или предостережение. Так я предупреждаю, например, о том, что неаккуратное применение некоего блока SQL может иметь неожиданные последствия.

Контакты

Пожалуйста, присылайте комментарии и вопросы по данной книге издателю:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (в Соединенных Штатах или Канаде)
(707) 829-0515 (международный или местный)
(707) 829-0104 (факс)

Для этой книги издательство O'Reilly поддерживает веб-страницу, на которой приведены список опечаток, примеры и вся дополнительная информация. Эту страницу можно найти по адресу:

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

Чтобы прокомментировать или задать технические вопросы по этой книге, присылайте электронные сообщения по адресу:

bookquestions@oreilly.com

Более подробная информация о книгах издательства O'Reilly, конференциях, центрах ресурсов и портале O'Reilly Network представлена на веб-сайте:

http://www.oreilly.com

Использование примеров кода

Цель этой книги – помочь вам выполнить работу. Код, представленный в книге, в общем случае можно использовать в программах и документации. На воспроизведение небольших фрагментов кода в вашей программе разрешение не требуется. Для продажи или распространения CD-ROM с примерами из книг O'Reilly разрешение *необходимо*. Если, отвечая на вопросы, вы ссылаетесь на книгу и цитируете пример кода, разрешение не требуется. Для включения существенного объема кода примеров из этой книги в документацию собственного продукта разрешение *необходимо*.

Мы признательны за указание авторства, но не требуем этого. Обычно указание источника включает название, автора, издателя и ISBN. Например: «Learning SQL by Alan Beaulieu. Copyright 2005 O'Reilly Media, Inc., 0-596-00727-2.»

Если вы сомневаетесь в корректности использования вами примеров кода, обратитесь за разъяснениями по адресу *permissions@oreilly.com*.

Safari Enabled

Если на обложке книги есть пиктограмма «Safari® Enabled», это означает, что книга доступна в Сети через O'Reilly Network Safari Bookshelf.

Safari предлагает намного лучшее решение, чем электронные книги. Это виртуальная библиотека, позволяющая без труда находить тысячи лучших технических книг, вырезать и вставлять примеры кода, загружать главы и находить быстрые ответы, когда требуется наиболее верная и свежая информация. Она свободно доступна по адресу *http://safari.oreilly.com*.

Благодарности

Книга – живой организм, и то, что сейчас перед вами, далеко от моих первоначальных набросков. Эта метаморфоза произошла во многом благодаря моему редактору Джонатану Геннику (Jonathan Gennick). Спасибо тебе за помощь на каждом этапе проекта – как за твою редакторскую доблесть, так и за экспертную поддержку в вопросах, связанных с языком SQL. Еще я хотел бы поблагодарить трех моих технических рецензентов: Питера Гулутзана (Peter Gulutzan), Джозефа Молинаро (Joseph Molinaro) и Джеффа Кокса (Jeff Cox), побудивших меня сделать эту книгу и технически насыщенной, и подходящей для читателей, не знакомых с SQL. Также огромная благодарность многим сотрудникам O'Reilly Media, помогавшим воплотить эту книгу в реальность, в том числе корректору Мэтт Хатчинсон (Matt Hutchinson), дизайнеру обложки Элли Волкхаузен (Ellie Volckhausen) и художнику-оформителю Робу Романо (Rob Romano).

3

Азбука запросов

Первые две главы содержали несколько примеров запросов к базам данных (т. е. выражений `select`). Теперь пришло время поближе рассмотреть разные части выражения `select` и их взаимодействие.

Механика запроса

Прежде чем анализировать выражение `select`, любопытно узнать, как сервер MySQL (или, коли на то пошло, любой сервер БД) выполняет запросы. Если вы используете клиентскую программу командной строки *mysql* (что я предполагаю), то уже зарегистрировались на сервере MySQL, предоставив свои имя пользователя и пароль (и, возможно, имя хоста, если сервер MySQL выполняется на другом компьютере). Как только сервер проверил правильность имени пользователя и пароля, для вас создается соединение с БД. Это соединение удерживается запросившим его приложением (которым в данном случае является инструмент *mysql*) до тех пор, пока приложение не высвободит соединение (например, в результате введения команды *quit*) или пока соединение не будет закрыто сервером (например, при выключении сервера). Каждому соединению с сервером MySQL присваивается идентификатор (ID), предоставляемый пользователю сразу после регистрации:

```
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 2 to server version: 4.1.11-nt

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

В данном случае ID соединения – 2. Эта информация может быть полезной администратору БД в случае каких-либо неполадок. Например, если требуется прервать плохо сформированный запрос, выполняющийся часами.

После того как сервер открыл соединение, проверив достоверность имени пользователя и пароля, можно выполнять запросы (и другие

SQL-выражения). При каждом запросе перед выполнением выражения сервер проверяет следующее:

- Есть ли у вас разрешение на выполнение выражения?
- Есть ли у вас разрешение на доступ к необходимым данным?
- Правилен ли синтаксис выражения?

Если выражение проходит все три теста, оно передается *оптимизатору запросов*, работа которого заключается в определении наиболее эффективного способа выполнения запроса. Оптимизатор рассмотрит порядок соединения таблиц, перечисленных в запросе, и доступные индексы, а затем определит *план выполнения*, используемый сервером при выполнении этого запроса.

Многие из вас заинтересуются тем, как понять и воздействовать на выбор сервером БД плана выполнения. Читатели, использующие MySQL, могут посмотреть книгу «High Performance MySQL» (O'Reilly). Кроме прочего, вы научитесь генерировать индексы, анализировать планы выполнения, оказывать влияние на оптимизатор посредством подсказок запроса и настраивать параметры запуска сервера. Для пользователей Oracle Database или SQL Server есть десятки книг по этой тематике.

По завершении выполнения запроса сервер возвращает в вызывающее приложение (опять же в инструмент *mysql*) *результирующий набор* (*result set*). Как было упомянуто в главе 1, результирующий набор – это просто еще одна таблица со строками и столбцами. Если по запросу не удается найти никаких данных, инструмент *mysql* отобразит сообщение, приведенное в конце следующего примера:

```
mysql> SELECT emp_id, fname, lname
    -> FROM employee
    -> WHERE lname = 'Bkadfl';
Empty set(0.00 sec)
```

Если запрос возвращает одну или более строк, программа форматирует результаты, добавляя заголовки столбцов и обводя столбцы рамкой из символов -, | и +, как показано в следующем примере:

```
mysql> SELECT fname, lname
-> FROM employee;
+----------+-----------+
| fname    | lname     |
+----------+-----------+
| Michael  | Smith     |
| Susan    | Barker    |
| Robert   | Tyler     |
| Susan    | Hawthorne |
| John     | Gooding   |
| Helen    | Fleming   |
| Chris    | Tucker    |
```

```
| Sarah    | Parker    |
| Jane     | Grossman  |
| Paula    | Roberts   |
| Thomas   | Ziegler   |
| Samantha | Jameson   |
| John     | Blake     |
| Cindy    | Mason     |
| Frank    | Portman   |
| Theresa  | Markham   |
| Beth     | Fowler    |
| Rick     | Tulman    |
+----------+-----------+
18 rows in set (0.00 sec)
```

Этот запрос возвращает имена и фамилии всех сотрудников из таблицы `employee`. После отображения последней строки данных инструмент *mysql* выводит на экран сообщение, указывающее, сколько строк было возвращено, в данном случае – 18 строк.

Блоки запроса

Выражение `select` могут образовывать несколько компонентов, или *блоков* (*clauses*). Хотя при работе с MySQL обязательным является только один из них (блок `select`), обычно в запрос включаются, по крайней мере, два-три из шести доступных блоков. В табл. 3.1 показаны разные блоки и их назначение.

Таблица 3.1. Блоки запроса

Блок	Назначение
`Select`	Определяет столбцы, которые должны быть включены в результирующий набор запроса
`From`	Указывает таблицы, из которых должны быть извлечены данные, и то, как эти таблицы должны быть соединены
`Where`	Ограничивает число строк в окончательном результирующем наборе
`Group by`	Используется для группировки строк по одинаковым значениям столбцов
`Having`	Ограничивает число строк в окончательном результирующем наборе с помощью группировки данных
`Order by`	Сортирует строки окончательного результирующего набора по одному или более столбцам

Все показанные в табл. 3.1 блоки включены в спецификацию ANSI. Кроме того, есть еще несколько блоков, используемых только в MySQL. Они будут рассмотрены в приложении B. В следующих разделах мы подробнее рассмотрим использование шести основных блоков запроса.

Блок select

Даже несмотря на то, что блок `select` является первым в выражении `select`, сервер БД обрабатывает его одним из последних. Причина в том, что прежде чем можно будет определить, что включать в окончательный результирующий набор, необходимо знать все столбцы, которые *могли бы* быть включены в этот набор. Поэтому, чтобы полностью понять роль блока `select`, надо немного разобраться с блоком `from`. Вот запрос для начала:

```
mysql> SELECT *
    -> FROM department;
+---------+----------------+
| dept_id | name           |
+---------+----------------+
|       1 | Operations     |
|       2 | Loans          |
|       3 | Administration |
+---------+----------------+
3 rows in set (0.04 sec)
```

В данном запросе в блоке `from` указана всего одна таблица (`department`), и блок `select` показывает, что в результирующий набор должны быть включены *все* столбцы (обозначено символом «*») таблицы `department`. Этот запрос можно перевести на естественный язык следующим образом:

Покажи мне все столбцы таблицы `department`.

Выбрать все столбцы можно не только с помощью символа звездочки, но и явно указав имена интересующих столбцов:

```
mysql> SELECT dept_id, name
    -> FROM department;
+---------+----------------+
| dept_id | name           |
+---------+----------------+
|       1 | Operations     |
|       2 | Loans          |
|       3 | Administration |
+---------+----------------+
3 rows in set (0.01 sec)
```

Результаты аналогичны первому запросу, поскольку в блоке `select` указаны все столбцы таблицы `department` (`dept_id` и `name`). А можно выбрать только некоторые из столбцов таблицы `department`:

```
mysql> SELECT name
    -> FROM department;
```

```
+----------------+
| name           |
+----------------+
| Operations     |
| Loans          |
| Administration |
+----------------+
3 rows in set (0.00 sec)
```

Таким образом, задача блока `select` заключается в следующем:

> Блок `select` определяет, какие из всех возможных столбцов должны быть включены в результирующий набор запроса.

Если бы приходилось выбирать столбцы только из таблицы или таблиц, указанных в блоке `from`, было бы скучновато. Хорошо, что можно добавить остроты, включив в блок `select` такие вещи, как:

- Литералы, например числа или строки
- Выражения, например `transaction.amount * -1`
- Вызовы встроенных функций, например `ROUND(transaction.amount, 2)`

Следующий запрос демонстрирует использование столбца таблицы, литерала, выражения и вызова встроенной функции в одном запросе к таблице `employee`:

```
mysql> SELECT emp_id,
    ->  'ACTIVE',
    ->  emp_id * 3.14159,
    ->  UPPER(lname)
    -> FROM employee;
+--------+--------+------------------+--------------+
| emp_id | ACTIVE | emp_id * 3.14159 | UPPER(lname) |
+--------+--------+------------------+--------------+
|      1 | ACTIVE |          3.14159 | SMITH        |
|      2 | ACTIVE |          6.28318 | BARKER       |
|      3 | ACTIVE |          9.42477 | TYLER        |
|      4 | ACTIVE |         12.56636 | HAWTHORNE    |
|      5 | ACTIVE |         15.70795 | GOODING      |
|      6 | ACTIVE |         18.84954 | FLEMING      |
|      7 | ACTIVE |         21.99113 | TUCKER       |
|      8 | ACTIVE |         25.13272 | PARKER       |
|      9 | ACTIVE |         28.27431 | GROSSMAN     |
|     10 | ACTIVE |         31.41590 | ROBERTS      |
|     11 | ACTIVE |         34.55749 | ZIEGLER      |
|     12 | ACTIVE |         37.69908 | JAMESON      |
|     13 | ACTIVE |         40.84067 | BLAKE        |
|     14 | ACTIVE |         43.98226 | MASON        |
|     15 | ACTIVE |         47.12385 | PORTMAN      |
|     16 | ACTIVE |         50.26544 | MARKHAM      |
|     17 | ACTIVE |         53.40703 | FOWLER       |
|     18 | ACTIVE |         56.54862 | TULMAN       |
```

```
+--------+--------+-----------------+--------------+
18 rows in set (0.05 sec)
```

Выражения и встроенные функции будут подробно рассмотрены позже, но я хотел дать представление о том, что может быть включено в блок `select`. Если требуется только выполнить встроенную функцию или вычислить простое выражение, можно вообще обойтись без блока `from`. Вот пример:

```
mysql> SELECT VERSION( ),
    ->   USER( ),
    ->   DATABASE( );
+-----------+-------------------+------------+
| VERSION() | USER()            | DATABASE( )|
+-----------+-------------------+------------+
| 4.1.11-nt | lrngsql@localhost | bank       |
+-----------+-------------------+------------+
1 row in set (0.02 sec)
```

Поскольку данный запрос просто вызывает три встроенные функции и не извлекает данные из таблиц, блок `from` здесь не нужен.

Псевдонимы столбцов

Хотя инструмент *mysql* и генерирует имена для столбцов, возвращаемых в результате запроса, вы можете задавать эти имена самостоятельно. Кроме того, что при желании можно дать другое имя столбцу из таблицы (если у него «плохое» или неоднозначное имя), практически наверняка вы захотите по-своему назвать те столбцы результирующего набора, которые будут сформированы в результате выполнения выражения или встроенной функции. Сделать это можно добавлением *псевдонима столбца* после каждого элемента блока `select`. Вот предыдущий запрос к таблице `employee`, в котором для трех столбцов указаны псевдонимы:

```
mysql> SELECT emp_id,
    ->   'ACTIVE' status,
    ->   emp_id * 3.14159 empid_x_pi,
    ->   UPPER(lname) last_name_upper
    -> FROM employee;
+--------+--------+------------+-----------------+
| emp_id | status | empid_x_pi | last_name_upper |
+--------+--------+------------+-----------------+
|      1 | ACTIVE |    3.14159 | SMITH           |
|      2 | ACTIVE |    6.28318 | BARKER          |
|      3 | ACTIVE |    9.42477 | TYLER           |
|      4 | ACTIVE |   12.56636 | HAWTHORNE       |
|      5 | ACTIVE |   15.70795 | GOODING         |
|      6 | ACTIVE |   18.84954 | FLEMING         |
|      7 | ACTIVE |   21.99113 | TUCKER          |
|      8 | ACTIVE |   25.13272 | PARKER          |
```

```
|      9 | ACTIVE |   28.27431 | GROSSMAN        |
|     10 | ACTIVE |   31.41590 | ROBERTS         |
|     11 | ACTIVE |   34.55749 | ZIEGLER         |
|     12 | ACTIVE |   37.69908 | JAMESON         |
|     13 | ACTIVE |   40.84067 | BLAKE           |
|     14 | ACTIVE |   43.98226 | MASON           |
|     15 | ACTIVE |   47.12385 | PORTMAN         |
|     16 | ACTIVE |   50.26544 | MARKHAM         |
|     17 | ACTIVE |   53.40703 | FOWLER          |
|     18 | ACTIVE |   56.54862 | TULMAN          |
+--------+--------+------------+-----------------+
18 rows in set (0.00 sec)
```

Как видно из заголовков столбцов, второй, третий и четвертый столбцы теперь имеют осмысленные имена, а не обозначены просто функцией или выражением, сформировавшим этот столбец. Если посмотреть на блок `select`, можно увидеть, что псевдонимы `status`, `empid_x_pi` и `last_name_upper` добавлены после второго, третьего и четвертого столбцов. Думаю, все согласятся с тем, что с присвоенными псевдонимами столбцов выходные данные стали понятнее; кроме того, с ними легче работать программно, если запросы формируются из Java или C#, а не интерактивно посредством инструмента командной строки *mysql*.

Уничтожение дубликатов

В некоторых случаях запрос может возвратить дублирующие строки данных. Например, при выборе ID всех клиентов, имеющих счета, было бы представлено следующее:

```
mysql> SELECT cust_id
    -> FROM account;
+---------+
| cust_id |
+---------+
|       1 |
|       1 |
|       1 |
|       2 |
|       2 |
|       3 |
|       3 |
|       4 |
|       4 |
|       4 |
|       5 |
|       6 |
|       6 |
|       7 |
|       8 |
|       8 |
|       9 |
```

```
|       9 |
|       9 |
|      10 |
|      10 |
|      11 |
|      12 |
|      13 |
+---------+
24 rows in set (0.00 sec)
```

Поскольку у некоторых клиентов по нескольку счетов, один и тот же ID клиента будет выведен столько раз, сколько счетов имеет клиент. Но, очевидно, целью данного запроса является *выбор* клиентов, имеющих счета, а не получение ID клиента для каждой строки таблицы `account`. Добиться этого можно, поместив ключевое слово `distinct` (отличный) непосредственно после ключевого слова `select`, как в следующем примере:

```
mysql> SELECT DISTINCT cust_id
    -> FROM account;
+---------+
| cust_id |
+---------+
|       1 |
|       2 |
|       3 |
|       4 |
|       5 |
|       6 |
|       7 |
|       8 |
|       9 |
|      10 |
|      11 |
|      12 |
|      13 |
+---------+
13 rows in set (0.01 sec)
```

Теперь в результирующем наборе 13 строк, по одной для каждого клиента, а не 24 строки, по одной для каждого счета.

Если не требуется, чтобы сервер удалял дублирующие данные, или вы уверены, что в результирующем наборе их не будет, вместо `DISTINCT` можно указать ключевое слово `ALL` (все). Однако ключевое слово `ALL` применяется по умолчанию и в явном указании не нуждается, поэтому большинство программистов не включает `ALL` в запросы.

Запомните, что формирование набора уникальных значений требует сортировки данных, что в случае больших результирующих наборов может занять много времени. Не поддавайтесь

соблазну использовать DISTINCT только для того, чтобы гарантировать отсутствие дублирования; лучше потратьте некоторое время на осмысление данных, с которыми работаете, чтобы уже наверняка знать, где дублирование возможно.

Блок from

До сих пор мы рассматривали запросы, в блоках from которых была указана только одна таблица. Хотя большинство книг по SQL определяют блок from просто как список из одной или более таблиц, мне бы хотелось расширить это определение следующим образом:

> Блок from определяет таблицы, используемые запросом, а также средства связывания таблиц.

Это определение включает две разные, но взаимосвязанные концепции, которые будут изучены в следующих разделах.

Таблицы

При встрече с термином table большинство людей представляют себе набор взаимосвязанных строк, хранящихся в базе данных. Хотя один из типов таблиц действительно описывается именно так, мне бы хотелось использовать это слово в более общем значении – избавиться от любого упоминания о способах хранения данных, сосредоточившись только на наборе взаимосвязанных строк. Этому свободному определению соответствуют три разных типа таблиц:

- Постоянные таблицы (т. е. созданные с помощью выражения create table)
- Временные таблицы (т. е. строки, возвращенные подзапросом)
- Виртуальные таблицы (представления) (т. е. созданные с помощью выражения create view)

Каждый из этих типов таблиц может быть включен в блок from запроса. На данный момент вы уже вполне освоили включение постоянных таблиц, поэтому далее кратко описаны другие типы таблиц, которые могут использоваться в блоке from.

Таблицы, формируемые подзапросом

Подзапрос (subquery) – это запрос, содержащийся в другом запросе. Подзапросы заключаются в круглые скобки и могут располагаться в различных частях выражения select. Однако в рамках блока from подзапрос выполняет функцию формирования временной таблицы, видимой для всех остальных блоков запроса и способной взаимодействовать с другими таблицами, указанными в блоке from. Вот простой пример:

```
mysql> SELECT e.emp_id, e.fname, e.lname
    -> FROM (SELECT emp_id, fname, lname, start_date, title
```

```
    ->   FROM employee) e;
+--------+----------+-----------+
| emp_id | fname    | lname     |
+--------+----------+-----------+
|      1 | Michael  | Smith     |
|      2 | Susan    | Barker    |
|      3 | Robert   | Tyler     |
|      4 | Susan    | Hawthorne |
|      5 | John     | Gooding   |
|      6 | Helen    | Fleming   |
|      7 | Chris    | Tucker    |
|      8 | Sarah    | Parker    |
|      9 | Jane     | Grossman  |
|     10 | Paula    | Roberts   |
|     11 | Thomas   | Ziegler   |
|     12 | Samantha | Jameson   |
|     13 | John     | Blake     |
|     14 | Cindy    | Mason     |
|     15 | Frank    | Portman   |
|     16 | Theresa  | Markham   |
|     17 | Beth     | Fowler    |
|     18 | Rick     | Tulman    |
+--------+----------+-----------+
18 rows in set (0.00 sec)
```

Здесь подзапрос к таблице `employee` возвращает пять столбцов, а *основной запрос* (*containing query*) ссылается на три из пяти доступных столбцов. Запрос ссылается на подзапрос посредством псевдонима, в данном случае `e`. Это упрощенный, практически бесполезный пример подзапроса в блоке `from`; подробный рассказ о подзапросах можно найти в главе 9.

Представления

Представление (view) – это запрос, хранящийся в словаре данных (data dictionary). Оно выглядит и работает как таблица, но с представлением не ассоциированы никакие данные (вот почему я называю это *виртуальной* таблицей). При выполнении запроса к представлению запрос соединяется с описанием представления и создается окончательный запрос, который и будет выполнен.

Чтобы продемонстрировать это, приведу описание представления, запрашивающего таблицу `employee` и включающего вызов встроенной функции:

```
CREATE VIEW employee_vw AS
SELECT emp_id, fname, lname,
  YEAR(start_date) start_year
FROM employee;
```

После создания представления никакие дополнительные данные не создаются: сервер просто сохраняет выражение `select` для дальнейшего

использования. Теперь, когда представление существует, можно делать запросы к нему:

```
SELECT emp_id, start_year
FROM employee_vw;

 Emp_id  start_year
-------- ----------
      1        2001
      2        2002
      3        2000
      4        2002
      5        2003
      6        2004
      7        2004
      8        2002
      9        2002
     10        2002
     11        2000
     12        2003
     13        2000
     14        2002
     15        2003
     16        2001
     17        2002
     18        2002
```

Представления создаются по разным причинам, в том числе с целью скрыть столбцы от пользователей и упростить сложно устроенные БД.

MySQL до версии 5.0.1 не поддерживает представления. Однако они широко используются другими серверами БД, поэтому тот, кто планирует работать с MySQL, должен помнить о них.

Поскольку MySQL версии 4.1.11 не включает представления, в предыдущем запросе намеренно не показано приглашение *mysql>* и обычное форматирование результирующего набора. Этот же прием применяется в других главах книги при описании возможности SQL, еще не реализованной в MySQL.

Связи таблиц

Второе отступление от определения простого блока `from`: если в блоке `from` присутствует более одной таблицы, обязательно должны быть включены и условия, используемые для *связывания* (*link*) таблиц. Это не требование MySQL или какого-то другого сервера БД, а утвержденный ANSI метод соединения нескольких таблиц, и это способ, наиболее переносимый между серверами БД. Соединение нескольких таблиц будет подробно рассматриваться в главах 5 и 10; здесь приведен лишь простой пример для утоления любопытства:

```
mysql> SELECT employee.emp_id, employee.fname,
    ->   employee.lname, department.name dept_name
```

```
    -> FROM employee INNER JOIN department
    ->   ON employee.dept_id = department.dept_id;
+--------+----------+-----------+----------------+
| emp_id | fname    | lname     | dept_name      |
+--------+----------+-----------+----------------+
|      1 | Michael  | Smith     | Administration |
|      2 | Susan    | Barker    | Administration |
|      3 | Robert   | Tyler     | Administration |
|      4 | Susan    | Hawthorne | Operations     |
|      5 | John     | Gooding   | Loans          |
|      6 | Helen    | Fleming   | Operations     |
|      7 | Chris    | Tucker    | Operations     |
|      8 | Sarah    | Parker    | Operations     |
|      9 | Jane     | Grossman  | Operations     |
|     10 | Paula    | Roberts   | Operations     |
|     11 | Thomas   | Ziegler   | Operations     |
|     12 | Samantha | Jameson   | Operations     |
|     13 | John     | Blake     | Operations     |
|     14 | Cindy    | Mason     | Operations     |
|     15 | Frank    | Portman   | Operations     |
|     16 | Theresa  | Markham   | Operations     |
|     17 | Beth     | Fowler    | Operations     |
|     18 | Rick     | Tulman    | Operations     |
+--------+----------+-----------+----------------+
18 rows in set (0.05 sec)
```

Предыдущий запрос выводит данные из таблиц `employee` (`emp_id`, `fname`, `lname`) и `department` (`name`), поэтому обе таблицы включены в блок `from`. Механизм связывания двух таблиц (называемый *соединением* (*join*)) заключается в присоединении данных об отделе, в котором работает сотрудник, хранящихся в таблице `employee`. Таким образом, серверу БД отдается распоряжение использовать значение столбца `dept_id` таблицы `employee` для поиска соответствующего названия отдела в таблице `department`. Условия соединения находятся в подблоке `on` блока `from`. В данном случае условие соединения: `ON e.dept_id = d.dept_id`. Всестороннее обсуждение соединения нескольких таблиц также можно найти в главе 5.

Определение псевдонимов таблиц

При соединении нескольких таблиц в одном запросе вам понадобится идентифицировать таблицу, на которую делается ссылка при указании столбцов в блоках `select`, `where`, `group by`, `having` и `order by`. Дать ссылку на таблицу вне блока `from` можно одним из двух способов:

- Использовать полное имя таблицы, например `employee.emp_id`.
- Присвоить каждой таблице псевдоним и использовать его в запросе.

В предыдущем запросе я решил использовать в блоках `select` и `on` полное имя таблицы. А вот как выглядит этот же запрос с применением псевдонимов:

```
SELECT e.emp_id, e.fname, e.lname,
  d.name dept_name
FROM employee e INNER JOIN department d
  ON e.dept_id = d.dept_id;
```

Если внимательнее посмотреть на блок `from`, видно, что таблица `employee` получила псевдоним `e`, а таблица `department` – псевдоним `d`. Затем эти псевдонимы используются в блоке `on` при описании условия соединения, а также в блоке `select` при задании столбцов, которые должны быть включены в результирующий набор. Надеюсь, все согласятся, что использование псевдонимов делает выражение более компактным, не приводя к путанице (при условии разумного выбора псевдонимов).

Блок where

До сих пор запросы, приводимые в данной главе, осуществляли выбор всех строк из таблиц `employee`, `department` или `account` (кроме примера с ключевым словом `distinct`). Однако чаще всего извлекать *все* строки таблицы не требуется, и нужен способ, позволяющий отфильтровывать строки, не представляющие интереса. Это работа для блока `where`.

Блок `where` – это механизм отсеивания нежелательных строк из результирующего набора.

Например, требуется извлечь из таблицы `employee` данные, но только для сотрудников, нанятых в качестве старших операционистов (head tellers). В следующем запросе блок `where` служит для извлечения *только* четырех старших операционистов:

```
mysql> SELECT emp_id, fname, lname, start_date, title
    -> FROM employee
    -> WHERE title = 'Head Teller';
+--------+---------+---------+------------+-------------+
| emp_id | fname   | lname   | start_date | title       |
+--------+---------+---------+------------+-------------+
|      6 | Helen   | Fleming | 2004-03-17 | Head Teller |
|     10 | Paula   | Roberts | 2002-07-27 | Head Teller |
|     13 | John    | Blake   | 2000-05-11 | Head Teller |
|     16 | Theresa | Markham | 2001-03-15 | Head Teller |
+--------+---------+---------+------------+-------------+
4 rows in set (0.00 sec)
```

В данном случае блоком `where` были отсеяны 14 из 18 строк. Этот блок `where` содержит всего одно *условие фильтрации* (*filter condition*), но этих условий может быть столько, сколько потребуется. Условия разделяются с помощью таких операторов, как `and`, `or` и `not` (подробно блок `where` и условия фильтрации обсуждаются в главе 4). Вот расширенный вариант предыдущего запроса со вторым условием – должны быть включены только сотрудники, принятые на работу после 1 января 2002 года:

```
mysql> SELECT emp_id, fname, lname, start_date, title
    -> FROM employee
    -> WHERE title = 'Head Teller'
    ->   AND start_date > '2002-01-01';
+--------+-------+---------+------------+-------------+
| emp_id | fname | lname   | start_date | title       |
+--------+-------+---------+------------+-------------+
|      6 | Helen | Fleming | 2004-03-17 | Head Teller |
|     10 | Paula | Roberts | 2002-07-27 | Head Teller |
+--------+-------+---------+------------+-------------+
2 rows in set (0.00 sec)
```

По первому условию (`title = 'Head Teller'`) было отфильтровано 14 из 18 строк, а по второму (`start_date > '2002-01-01'`) – еще 2. В итоге в результирующем наборе осталось 2 строки. Давайте посмотрим, что произойдет, если заменить разделяющий условия оператор `and` оператором `or`:

```
mysql> SELECT emp_id, fname, lname, start_date, title
    -> FROM employee
    -> WHERE title = 'Head Teller'
    ->   OR start_date > '2002-01-01';
+--------+----------+-----------+------------+--------------------+
| emp_id | fname    | lname     | start_date | title              |
+--------+----------+-----------+------------+--------------------+
|      2 | Susan    | Barker    | 2002-09-12 | Vice President     |
|      4 | Susan    | Hawthorne | 2002-04-24 | Operations Manager |
|      5 | John     | Gooding   | 2003-11-14 | Loan Manager       |
|      6 | Helen    | Fleming   | 2004-03-17 | Head Teller        |
|      7 | Chris    | Tucker    | 2004-09-15 | Teller             |
|      8 | Sarah    | Parker    | 2002-12-02 | Teller             |
|      9 | Jane     | Grossman  | 2002-05-03 | Teller             |
|     10 | Paula    | Roberts   | 2002-07-27 | Head Teller        |
|     12 | Samantha | Jameson   | 2003-01-08 | Teller             |
|     13 | John     | Blake     | 2000-05-11 | Head Teller        |
|     14 | Cindy    | Mason     | 2002-08-09 | Teller             |
|     15 | Frank    | Portman   | 2003-04-01 | Teller             |
|     16 | Theresa  | Markham   | 2001-03-15 | Head Teller        |
|     17 | Beth     | Fowler    | 2002-06-29 | Teller             |
|     18 | Rick     | Tulman    | 2002-12-12 | Teller             |
+--------+----------+-----------+------------+--------------------+
15 rows in set (0.00 sec)
```

Посмотрев на выходные данные, можно увидеть, что в результирующий набор включены все четыре старших операциониста (Head Teller), а также все остальные сотрудники, приступившие к работе в банке после 1 января 2002 года. Для 15 из 18 сотрудников из таблицы `employee` выполняется по крайней мере одно из двух условий. Таким образом, чтобы строка попала в результирующий набор, когда условия разделяются оператором `and`, для нее должны выполняться *все* условия; а при использовании оператора `or` достаточно, чтобы выполнялось только *одно* из условий.

А как быть, если вам нужно задействовать в блоке `where` оба оператора – `and` и `or`? Рад, что спросили. Необходимо сгруппировать условия с помощью круглых скобок. Следующий запрос составлен так, что в результирующий набор должны попасть только те сотрудники, которые являются старшими операционистами (Head Teller) и начали работать в компании позже 1 января 2002 года, *или* простые операционисты (Teller), начавшие работать после 1 января 2003 года:

```
mysql> SELECT emp_id, fname, lname, start_date, title
    -> FROM employee
    -> WHERE (title = 'Head Teller' AND start_date > '2002-01-01')
    ->   OR (title = 'Teller' AND start_date > '2003-01-01');
+--------+----------+---------+------------+-------------+
| emp_id | fname    | lname   | start_date | title       |
+--------+----------+---------+------------+-------------+
|      6 | Helen    | Fleming | 2004-03-17 | Head Teller |
|      7 | Chris    | Tucker  | 2004-09-15 | Teller      |
|     10 | Paula    | Roberts | 2002-07-27 | Head Teller |
|     12 | Samantha | Jameson | 2003-01-08 | Teller      |
|     15 | Frank    | Portman | 2003-04-01 | Teller      |
+--------+----------+---------+------------+-------------+
5 rows in set (0.00 sec)
```

Для разделения групп условий при использовании различных операторов всегда следует применять круглые скобки, чтобы автор запроса, сервер БД и любой специалист, который позже будет работать с этим кодом, понимали, что происходит.

Блоки group by и having

Все рассмотренные до сих пор запросы извлекали необработанные строки данных, не выполняя над ними никаких действий. Однако иногда вам захочется выявить в данных общие направления, для чего серверу БД придется немного поколдовать над ними, прежде чем предоставить вам результирующий набор. Одним из средств для этого является блок `group by`, предназначенный для группировки данных по значениям столбцов. Например, вместо списка сотрудников и отделов, в которых они числятся, нужен список отделов с числом сотрудников, работающих в каждом из них. С блоком `group by` также можно использовать блок `having`, позволяющий фильтровать данные групп аналогично блоку `where`, позволяющему фильтровать необработанные данные.

Я хотел лишь слегка коснуться этих двух блоков, чтобы в дальнейшем они не были неожиданностью для читателей, но они немного сложнее, чем другие четыре блока выражения `select`. Поэтому прошу дождаться главы 8, где полностью описано, как и когда использовать `group by` и `having`.

Блок order by

В общем случае строки результирующего набора запроса возвращаются в произвольном порядке. Если требуется упорядочить результирующий набор определенным образом, необходимо предписать серверу сортировать результаты с помощью блока `order by`:

> Блок `order by` – это механизм сортировки результирующего набора на основе данных столбцов, или выражений, использующих данные столбцов.

Вот, к примеру, еще один взгляд на приведенный ранее запрос к таблице `account`:

```
mysql> SELECT open_emp_id, product_cd
    -> FROM account;
+-------------+------------+
| open_emp_id | product_cd |
+-------------+------------+
|          10 | CHK        |
|          10 | SAV        |
|          10 | CD         |
|          10 | CHK        |
|          10 | SAV        |
|          13 | CHK        |
|          13 | MM         |
|           1 | CHK        |
|           1 | SAV        |
|           1 | MM         |
|          16 | CHK        |
|           1 | CHK        |
|           1 | CD         |
|          10 | CD         |
|          16 | CHK        |
|          16 | SAV        |
|           1 | CHK        |
|           1 | MM         |
|           1 | CD         |
|          16 | CHK        |
|          16 | BUS        |
|          10 | BUS        |
|          16 | CHK        |
|          13 | SBL        |
+-------------+------------+
24 rows in set (0.00 sec)
```

Если требуется проанализировать данные каждого сотрудника, полезно было бы отсортировать результаты по столбцу `open_emp_id`. Для этого просто добавляем этот столбец в блок `order by`:

```
mysql> SELECT open_emp_id, product_cd
    -> FROM account
    -> ORDER BY open_emp_id;
```

```
+-------------+------------+
| open_emp_id | product_cd |
+-------------+------------+
|           1 | CHK        |
|           1 | SAV        |
|           1 | MM         |
|           1 | CHK        |
|           1 | CD         |
|           1 | CHK        |
|           1 | MM         |
|           1 | CD         |
|          10 | CHK        |
|          10 | SAV        |
|          10 | CD         |
|          10 | CHK        |
|          10 | SAV        |
|          10 | CD         |
|          10 | BUS        |
|          13 | CHK        |
|          13 | MM         |
|          13 | SBL        |
|          16 | CHK        |
|          16 | CHK        |
|          16 | SAV        |
|          16 | CHK        |
|          16 | BUS        |
|          16 | CHK        |
+-------------+------------+
24 rows in set (0.00 sec)
```

Теперь легче увидеть, какие типы счетов были открыты каждым сотрудником. Однако было бы гораздо лучше, если бы типы счетов для каждого отдельного сотрудника выводились в определенном порядке; это осуществляется путем добавления в блок `order by` столбца `product_cd` после `open_emp_id`:

```
mysql> SELECT open_emp_id, product_cd
    -> FROM account
    -> ORDER BY open_emp_id, product_cd;
+-------------+------------+
| open_emp_id | product_cd |
+-------------+------------+
|           1 | CD         |
|           1 | CD         |
|           1 | CHK        |
|           1 | CHK        |
|           1 | CHK        |
|           1 | MM         |
|           1 | MM         |
|           1 | SAV        |
|          10 | BUS        |
```

```
|          10 | CD          |
|          10 | CD          |
|          10 | CHK         |
|          10 | CHK         |
|          10 | SAV         |
|          10 | SAV         |
|          13 | CHK         |
|          13 | MM          |
|          13 | SBL         |
|          16 | BUS         |
|          16 | CHK         |
|          16 | CHK         |
|          16 | CHK         |
|          16 | CHK         |
|          16 | SAV         |
+-------------+-------------+
24 rows in set (0.00 sec)
```

Теперь результирующий набор отсортирован сначала по ID сотрудников, а затем по типу счета. Учитывается порядок размещения столбцов в блоке `order by`.

Сортировка по возрастанию и убыванию

При сортировке можно задать порядок *по возрастанию* (*ascending*) или *по убыванию* (*descending*) с помощью ключевых слов `asc` и `desc`. По умолчанию выполняется сортировка по возрастанию, поэтому добавлять придется только ключевое слово `desc` – если требуется сортировка по убыванию. Например, по следующему запросу выводится список всех счетов, отсортированный по доступному остатку, начиная с самого большого:

```
mysql> SELECT account_id, product_cd, open_date, avail_balance
    -> FROM account
    -> ORDER BY avail_balance DESC;
+------------+------------+------------+---------------+
| account_id | product_cd | open_date  | avail_balance |
+------------+------------+------------+---------------+
|         24 | SBL        | 2004-02-22 |      50000.00 |
|         23 | CHK        | 2003-07-30 |      38552.05 |
|         20 | CHK        | 2002-09-30 |      23575.12 |
|         13 | CD         | 2004-12-28 |      10000.00 |
|         22 | BUS        | 2004-03-22 |       9345.55 |
|         18 | MM         | 2004-10-28 |       9345.55 |
|         10 | MM         | 2004-09-30 |       5487.09 |
|         14 | CD         | 2004-01-12 |       5000.00 |
|         15 | CHK        | 2001-05-23 |       3487.19 |
|          3 | CD         | 2004-06-30 |       3000.00 |
|          4 | CHK        | 2001-03-12 |       2258.02 |
|         11 | CHK        | 2004-01-27 |       2237.97 |
|          7 | MM         | 2002-12-15 |       2212.50 |
|         19 | CD         | 2004-06-30 |       1500.00 |
```

```
|          1 | CHK        | 2000-01-15 |       1057.75 |
|          6 | CHK        | 2002-11-23 |       1057.75 |
|          9 | SAV        | 2000-01-15 |        767.77 |
|          8 | CHK        | 2003-09-12 |        534.12 |
|          2 | SAV        | 2000-01-15 |        500.00 |
|         16 | SAV        | 2001-05-23 |        387.99 |
|          5 | SAV        | 2001-03-12 |        200.00 |
|         17 | CHK        | 2003-07-30 |        125.67 |
|         12 | CHK        | 2002-08-24 |        122.37 |
|         21 | BUS        | 2002-10-01 |          0.00 |
+------------+------------+------------+---------------+
24 rows in set (0.01 sec)
```

Сортировка по убыванию обычно применяется в ранжирующих запросах вроде «покажи мне пять самых больших доступных остатков». MySQL включает блок `limit` (предел), позволяющий сортировать данные и затем отбрасывать все, кроме первых *X* строк. Блок `limit` обсуждается в приложении B вместе с другими расширениями, не входящими в стандарт ANSI.

Сортировка с помощью выражений

Сортировать результаты по данным столбца легко и приятно, но иногда может потребоваться сортировка по какому-то признаку, который не хранится в БД и, возможно, отсутствует в запросе. Чтобы справиться с этой ситуацией, можно добавить в блок `order by` выражение. Например, требуется сортировать данные клиентов по последним трем разрядам их федерального ID (это либо номер социальной страховки для физических лиц, либо корпоративный ID для юридических лиц):

```
mysql> SELECT cust_id, cust_type_cd, city, state, fed_id
    -> FROM customer
    -> ORDER BY RIGHT(fed_id, 3);
+---------+--------------+------------+-------+-------------+
| cust_id | cust_type_cd | city       | state | fed_id      |
+---------+--------------+------------+-------+-------------+
|       1 | I            | Lynnfield  | MA    | 111-11-1111 |
|      10 | B            | Salem      | NH    | 04-1111111  |
|       2 | I            | Woburn     | MA    | 222-22-2222 |
|      11 | B            | Wilmington | MA    | 04-2222222  |
|       3 | I            | Quincy     | MA    | 333-33-3333 |
|      12 | B            | Salem      | NH    | 04-3333333  |
|      13 | B            | Quincy     | MA    | 04-4444444  |
|       4 | I            | Waltham    | MA    | 444-44-4444 |
|       5 | I            | Salem      | NH    | 555-55-5555 |
|       6 | I            | Waltham    | MA    | 666-66-6666 |
|       7 | I            | Wilmington | MA    | 777-77-7777 |
|       8 | I            | Salem      | NH    | 888-88-8888 |
|       9 | I            | Newton     | MA    | 999-99-9999 |
+---------+--------------+------------+-------+-------------+
13 rows in set (0.01 sec)
```

В этом запросе используется встроенная функция `right()`, которая извлекает последние три символа значения столбца `fed_id` и сортирует строки на основании этого значения.

Сортировка с помощью числовых заместителей

При сортировке с использованием столбцов, перечисленных в блоке `select`, можно ссылаться на столбцы не по имени, а по их *порядковому номеру*. Например, если требуется выполнить сортировку по второму или пятому столбцу, возвращаемому запросом, можно сделать следующее:

```
mysql> SELECT emp_id, title, start_date, fname, lname
    -> FROM employee
    -> ORDER BY 2, 5;
+--------+--------------------+------------+----------+-----------+
| emp_id | title              | start_date | fname    | lname     |
+--------+--------------------+------------+----------+-----------+
|     13 | Head Teller        | 2000-05-11 | John     | Blake     |
|      6 | Head Teller        | 2004-03-17 | Helen    | Fleming   |
|     16 | Head Teller        | 2001-03-15 | Theresa  | Markham   |
|     10 | Head Teller        | 2002-07-27 | Paula    | Roberts   |
|      5 | Loan Manager       | 2003-11-14 | John     | Gooding   |
|      4 | Operations Manager | 2002-04-24 | Susan    | Hawthorne |
|      1 | President          | 2001-06-22 | Michael  | Smith     |
|     17 | Teller             | 2002-06-29 | Beth     | Fowler    |
|      9 | Teller             | 2002-05-03 | Jane     | Grossman  |
|     12 | Teller             | 2003-01-08 | Samantha | Jameson   |
|     14 | Teller             | 2002-08-09 | Cindy    | Mason     |
|      8 | Teller             | 2002-12-02 | Sarah    | Parker    |
|     15 | Teller             | 2003-04-01 | Frank    | Portman   |
|      7 | Teller             | 2004-09-15 | Chris    | Tucker    |
|     18 | Teller             | 2002-12-12 | Rick     | Tulman    |
|     11 | Teller             | 2000-10-23 | Thomas   | Ziegler   |
|      3 | Treasurer          | 2000-02-09 | Robert   | Tyler     |
|      2 | Vice President     | 2002-09-12 | Susan    | Barker    |
+--------+--------------------+------------+----------+-----------+
18 rows in set (0.03 sec)
```

Скорее всего, вы редко будете использовать эту возможность, поскольку если добавить столбец в блок `select` и не изменить порядковые номера в блоке `order by`, результаты будут непредсказуемыми.

Упражнения

Следующие упражнения разработаны для закрепления понимания выражения `select` и его блоков. Решения приведены в приложении C.

3.1

Извлеките ID, имя и фамилию всех банковских сотрудников. Выполните сортировку по фамилии, а затем по имени.

3.2

Извлеките ID счета, ID клиента и доступный остаток всех счетов, имеющих статус `'ACTIVE'` (активный) и доступный остаток более 2500 долларов.

3.3

Напишите запрос к таблице `account`, возвращающий ID сотрудников, открывших счета (используйте столбец `account.open_emp_id`). Результирующий набор должен включать по одной строке для каждого сотрудника.

3.4

В этом запросе к нескольким наборам данных заполните пробелы (обозначенные как *<число>*) так, чтобы получить результат, приведенный ниже:

```
mysql> SELECT p.product_cd, a.cust_id, a.avail_balance
    -> FROM product p INNER JOIN account <1>
    ->   ON p.product_cd = <2>
    -> WHERE p.<3> = 'ACCOUNT';
+------------+---------+---------------+
| product_cd | cust_id | avail_balance |
+------------+---------+---------------+
| CD         |       1 |       3000.00 |
| CD         |       6 |      10000.00 |
| CD         |       7 |       5000.00 |
| CD         |       9 |       1500.00 |
| CHK        |       1 |       1057.75 |
| CHK        |       2 |       2258.02 |
| CHK        |       3 |       1057.75 |
| CHK        |       4 |        534.12 |
| CHK        |       5 |       2237.97 |
| CHK        |       6 |        122.37 |
| CHK        |       8 |       3487.19 |
| CHK        |       9 |        125.67 |
| CHK        |      10 |      23575.12 |
| CHK        |      12 |      38552.05 |
| MM         |       3 |       2212.50 |
| MM         |       4 |       5487.09 |
| MM         |       9 |       9345.55 |
| SAV        |       1 |        500.00 |
| SAV        |       2 |        200.00 |
| SAV        |       4 |        767.77 |
| SAV        |       8 |        387.99 |
+------------+---------+---------------+
21 rows in set (0.02 sec)
```