САМОУЧИТЕЛЬ

bhv

УРОКИ MySQL

Анатолий Мотев

- Установка и настройка СУБД MySQL
- Проектирование и создание баз данных
- Основы языка SQL
- Создание приложений на PHP

+ CD

Создай базу данных своими руками!

Анатолий Мотев

УРОКИ MySQL

САМОУЧИТЕЛЬ

Санкт-Петербург
«БХВ-Петербург»
2006

УДК 681.3.06
ББК 32.973.26-018.2
М85

Мотев А. А.

М85 Уроки MySQL. Самоучитель. — СПб.: БХВ-Петербург, 2006. — 208 с.: ил.

ISBN 5-94157-658-7

Книга посвящена использованию СУБД MySQL для разработки интернет-проектов. В виде уроков рассмотрены все необходимые этапы работы с базами данных: от проектирования структуры до реализации приложений на языке PHP, позволяющих манипулировать данными. Изложенный материал сопровождается многочисленными примерами, комментариями и упражнениями. Показано, как создать гостевую книгу, форум, регистрацию пользователей, интернет-магазин и другие сложные элементы web-сайта. К книге прилагается компакт-диск, который содержит учебную базу данных и скрипты, описанные в книге, а также дистрибутивы MySQL и другие программы, распространяемые по лицензии GNU/GPL.

Для программистов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Татьяна Темкина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 22.03.06.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 16,77.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 5-94157-658-7

Оглавление

Предисловие

Большим компаниям необходимы базы данных, а также сложное и дорогое программное обеспечение для работы с ними. Такие программные продукты позволяют управлять огромными объемами информации целой компании, поскольку обладают полным набором функций для работы с данными.

Напротив, пользователи домашних компьютеров обычно не нуждаются в базах данных. Они хранят свою информацию (телефоны, адреса и т. д.) в небольших файлах, создаваемых с помощью специальных программ вроде записных книжек, электронных таблиц или телефонных справочников.

Но есть и промежуточная категория пользователей, которым требуется хранение и управление информацией средних объемов. К этой категории относятся небольшие предприятия и организации, филиалы больших компаний, а также пользователи домашних компьютеров, которым нужно поддерживать сложные персональные данные (например, размещаемый в Интернете каталог любимых фильмов с дополнительной информацией о сюжетах, актерах, съемочных группах и т. д.).

Таким пользователям как раз и адресована книга.

О чем эта книга

В данной книге рассматриваются возможности системы управления базами данных (СУБД) MySQL, вопросы создания баз данных в этой среде и их применения для реализации полноценных приложений в сети Интернет. Она познакомит читателя с основами баз данных, научит проектировать и создавать их с использованием MySQL, а также настраивать базу данных и управлять ею. СУБД MySQL достаточно проста в освоении и использовании, а множество примеров и практических заданий помогут вам лучше и быстрее усвоить материал.

Также в этой книге большое внимание уделено взаимодействию MySQL с программами на языке PHP, используемом для создания динамических сайтов в Интернете. Это позволит вам применить полученные знания и навыки на практике и создать гостевую книгу, форум и многие другие полезные и интересные приложения на основе баз данных для своего сайта или сайта вашей фирмы.

Примечание

При описании интерфейса для языка PHP предполагается наличие у читателя основных навыков работы с рассматриваемым языком.

Как пользоваться книгой

Книга разделена на четыре части.

Во *введении* я расскажу о том, что такое базы данных и где их можно использовать. Также в нем описаны модели построения базы данных.

Часть I "Проектирование базы данных" может сначала показаться не представляющей ценности, но на самом деле это одна из самых важных частей книги. Правильное проектирование важно для разработчика, если ставится задача создания приложения для работы с базами данных, достаточно легко масштабируемого в случае необходимости внесения изменений. Правильное проектирование базы данных также позволяет обеспечить высокую производительность.

Часть II "MySQL" описывает процесс инсталляции ядра MyQSL и использование установленных утилит для обращения к базе данных.

В *части III "Формирование запросов к БД. Язык SQL"* приведены важнейшие конструкции языка SQL, используемого для формирования структуры базы данных и манипулирования информацией, хранящейся в ней.

Часть IV "PHP и MySQL" предназначена для тех, кто уже имеет базовый опыт создания программ на языке PHP и хочет узнать, как "заставить" свои приложения общаться с базами данных MySQL.

Материал книги изложен в виде уроков, последовательно описывающих принципы работы с СУБД MySQL. В некоторых уроках есть контрольные задания, выполнение которых поможет вам оценить, насколько хорошо усвоен данный урок.

Приложения содержат справочные материалы — таблицу зарезервированных слов СУБД MySQL и список функций языка PHP, используемых для работы с MySQL, а также ответы на контрольные вопросы и примеры выполнения практических заданий, приведенных в книге, и описание сопроводительного компакт-диска книги.

Введение

Эта книга вводит вас в мир разработки малых и средних баз данных с помощью MySQL. MySQL *не является* базой данных. Это компьютерная программа, позволяющая пользователю создавать, поддерживать базы данных и управлять ими. Такой тип программного обеспечения называется *системой управления базами данных (СУБД)*. СУБД действует как посредник между физической базой данных и ее пользователями.

Создателем СУБД MySQL является Майкл Видениус (Michael Widenius, Monty) из шведской компании TcX. В 1979 г. он разработал для внутрифирменного использования средство управления базами данных и назвал его UNIREG. Впоследствии программа UNIREG была переписана на нескольких разных языках и расширена для поддержки больших баз данных.

В 1994 г. компания TcX начала разрабатывать приложения для Интернета, применяя для поддержки этого проекта программу UNIREG. К сожалению, динамическая генерация web-страниц с помощью этой программы приводила к большим накладным расходам. Поэтому разработчики из TcX, взяв UNIREG за основу, использовали утилиты сторонних разработчиков для mSQL и к маю 1995 г. у TcX имелась СУБД, удовлетворявшая внутренним потребностям компании, — MySQL 1.0.

Что же касается названия, то сам разработчик говорит об этом так: "До конца неясно, откуда идет название MySQL. В TcX базовый каталог, а также значительное число библиотек и утилит в течение десятка лет имели префикс "my". Вместе с тем мою дочь тоже зовут Май (My). Поэтому остается тайной, какой из двух источников дал название MySQL".

С момента публикации СУБД MySQL в Интернете она была перенесена на многие системы под управлением ОС UNIX, а также Win32 и OS/2. TcX считает, что сейчас MySQL используется примерно на 500 000 серверов.

Если у вас есть опыт работы с реляционными базами данных и их проектирования, вы можете сразу перейти к *части II* данной книги. Но если вы только

начали осваивать данную область, то информация, приведенная здесь и в *части I*, будет весьма полезной и интересной.

Что такое базы данных и где они используются

Для начала нужно выяснить, что представляют собой базы данных и для каких задач их можно, а иногда и необходимо применять.

Определение базы данных может звучать так:

Определение

База данных (БД) — это именованная совокупность данных, отражающая совокупность объектов и их отношения в предметной области.

Наверное, читателю не совсем понятно данное определение. Давайте же разберемся. *Предметной областью* для БД является то, что она описывает. Например, предметом для описания может быть магазин по продаже детских игрушек, склад товаров или список сотрудников большой компании, хранящийся в отделе кадров. То есть предметной областью для базы данных может быть все, что требует хранения довольно больших по объему и сложных по структуре данных.

Объекты — это структурные единицы, из которых состоит предметная область. Если в качестве предметной области взять магазин, то объектами будут являться товары, покупатели, продавцы, поставщики, заказы и т. д.

У каждого объекта есть набор *свойств*, который необходимо описать. Например, у товара могут быть следующие свойства:

1. Код (артикул).
2. Наименование.
3. Фирма-производитель.
4. Описание.
5. Цена.
6. Количество единиц на складе.

Все объекты, описанные в предметной области, связаны между собой отношениями (рис. В1).

То есть поставщики поставляют в магазин различные товары, покупатели делают заказы, заказы производятся на определенный товар и т. д.

И, наконец, совокупность (собрание) данных является именованной просто потому, что любая база данных имеет имя (название).

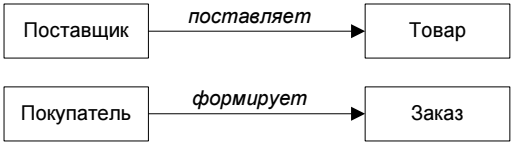


Рис. В1. Примеры отношений между объектами

Базы данных и приложения

Каждая база данных служит определенной цели. Простого наличия СУБД не достаточно, чтобы у вашей базы данных появилось назначение. Все зависит от того, как вы будете использовать хранимые данные. Если магазин не будет продавать товары, то какой смысл в организации и хранении товаров, ведении складского учета и т. д.? Кроме этого, нужна возможность изменения информации об имеющихся товарах и добавления информации о новых. Представьте себе компьютерный магазин, продающий в течение многих лет одни и те же процессоры, видеокарты и другие комплектующие. Я думаю, вскоре его станут называть "Антикварным".

Базы данных существуют для того, чтобы мы могли с ними взаимодействовать. Но взаимодействие происходит не непосредственно с базой данных, а косвенно — с помощью специально разработанного программного обеспечения. До появления Интернета базы данных обычно использовались в больших компаниях для поддержки всевозможных функций — бухгалтерии, финансов, контроля поставок, складского учета, учета персонала и т. п. Развитие Интернета и усложнение задач домашних вычислений способствовали перемещению потребностей в использовании БД за пределы больших компаний.

Базы данных и Интернет

Наиболее популярная область применения MySQL — разработка приложений для Интернета. В настоящее время спрос на сложные и надежные приложения достаточно велик. Соответственно вырос спрос и на базы данных. С их помощью можно реализовывать множество полезных функций web-сайта. Практически любым содержимым страницы можно управлять с помощью базы данных.

Возьмем в качестве примера интернет-магазин. Заходя на такой сайт, мы должны видеть список всех доступных товаров с их описанием и фотографиями. Если опубликовать каталог товаров в виде простой HTML-страницы, то придется вручную редактировать ее каждый раз, когда требуется добавить новый товар или изменить цену. Если же вместо этого хранить данные о товарах в базе данных, то появится возможность изменения web-страницы в

режиме реального времени, сразу же после внесения в базу данных сведений о новом товаре или изменений для имеющегося товара. Кроме этого, можно будет интегрировать сайт с какой-либо системой электронной коммерции.

Так как же web-страница взаимодействует с базой данных? База данных находится на вашем web-сервере. На web-странице вы размещаете форму, в которую пользователь вводит свой запрос (например, для поиска чего-либо) или те данные, которые нужно передать. После отправки данных из формы на сервер последний запускает написанную вами программу, которая извлекает данные, переданные пользователем. Далее программа формирует запрос на языке SQL *(см. часть III)* для выборки или изменения данных, а СУБД чудесным образом делает все остальное. Если пользователь запрашивал какие-то данные из БД, то ваша программа может оформить результат запроса в виде новой HTML-страницы и отправить обратно пользователю.

Обычно такие программы создаются в виде CGI-сценариев или серверных приложений на языке Java. Возможно также встраивание программы прямо в HTML-страницу.

Таким образом, использование базы данных для управления подобным сайтом дает очевидные преимущества как продавцу, так и покупателю.

Примечание

Подробнее взаимодействие базы данных и приложений описано в *части IV*. В ней я познакомлю вас с базовым синтаксисом языка PHP, а также расскажу о возможностях, предоставляемых данным языком для работы с базами данных.

Другие СУБД среднего масштаба

Первой СУБД среднего масштаба с поддержкой SQL была mSQL. Она недолго оставалась в одиночестве — в настоящий момент ее коллегами являются такие СУБД, как PostgreSQL, GNU SQL, Beagle и уже известная нам MySQL. Все эти продукты относятся к категории Open Source, т. е. поставляются с открытым исходным кодом (даже если за использование этого продукта необходимо заплатить).

PostgreSQL

В начале 1980-х доктор Майкл Стоунбрейкер (Michael Stonebreaker) из Калифорнийского университета в Беркли (University of California at Berkeley) создал систему управления базами данных Ingres, которая предвосхитила многие концепции, реализованные в современных СУБД. Ingres была некоммерческим проектом, который финансировался университетом.

Одна из компаний обратила внимание на коммерческий потенциал этого продукта и, зарегистрировав торговую марку Ingres, выпустила коммерческий

продукт. Исходная некоммерческая версия Ingres была переименована в University Ingres и в дальнейшем развивалась независимо от коммерческой версии.

Через некоторое время доктор Стоунбрейкер пошел в своих исследованиях дальше того, что предполагалось в начальных целях проекта Ingres. Он решил разработать совершенно новую систему управления базами данных, которая бы развила идеи, заложенные в Ingres. Эта СУБД стала известна как Postgres.

Postgres, как и Ingres, была открытым для всех проектом (он также финансировался университетом). Сейчас Postgres соперничает по популярности с MySQL и mSQL среди СУБД среднего масштаба.

Подробнее узнать о PostgreSQL можно на сайте **http://www.postgresql.org**.

GNU SQL

Проект GNU является для многих разработчиков символом свободы. Официальная лицензия на продукты GNU гарантирует свободный доступ и полную свободу для изменений исходного кода. До недавнего времени большим пробелом было отсутствие СУБД среди таких продуктов.

Институт системного программирования Российской академии наук работает над устранением этого недостатка. Недавно он выпустил первую открытую бета-версию GNU SQL — полнофункциональную СУБД с поддержкой языка SQL.

В настоящее время GNU SQL поддерживает многие развитые возможности — транзакции, вложенные запросы.

Подробнее узнать о GNU SQL можно на сайте **http://www.ispras.ru**.

Beagle

Проект Beagle был разработан и реализован Робертом Клейном (Robert Klein). Так же как и GNU SQL, Beagle был задуман как полностью SQL-совместимый сервер со всеми необходимыми функциями.

Домашняя страница Beagle расположена на сайте **http://www.beaglesql.org**.

Модели данных

Различают три модели, по которым можно построить БД:

- иерархическая;
- сетевая;
- реляционная.

В СУБД MySQL используется реляционная модель данных. Но все же про другие модели следует сказать несколько слов, чтобы читатель имел о них представление.

Иерархическая модель

В иерархической модели отношения между объектами строятся в виде дерева. Данная модель характеризуется такими параметрами, как уровни, узлы, связи. Принцип работы модели таков: несколько узлов более низкого уровня соединяются при помощи связи с одним узлом более высокого уровня. *Узел* (сегмент дерева) — информационная модель элемента, находящегося на данном уровне иерархии.

В качестве примера можно привести базу данных университета (рис. В2).

Рис. В2. Иерархическая модель

Обучение в университете разделено на курсы, на каждом курсе есть какое-то количество групп, в состав каждой группы входят конкретные студенты. Рассмотрев данный пример, мы можем выделить следующие свойства иерархической модели:

- несколько узлов низшего уровня могут быть связаны только с одним узлом высшего уровня;
- каждый узел имеет свое имя;
- у иерархического дерева имеется только одна вершина (корень), не подчиненная никакой другой вершине.

Сетевая модель

Сетевая модель базы данных похожа на иерархическую. Она имеет те же основные составляющие (узел, уровень, связь), однако характер их отношений принципиально иной. Сетевая модель основана на связях между *наборами данных (агрегатами)*.

В качестве примера можно привести базу данных преподавательского состава университета (рис. В3).

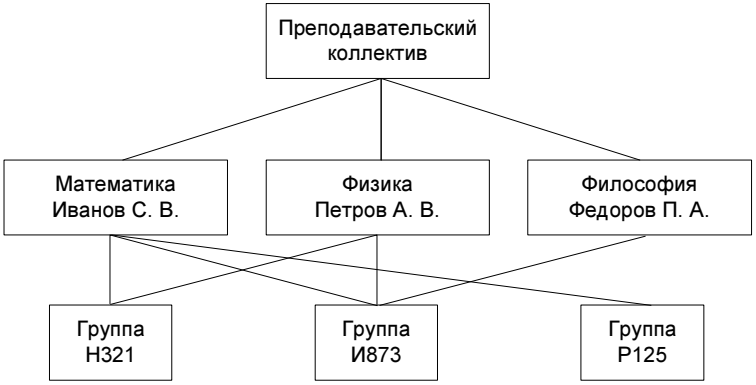


Рис. В3. Сетевая модель

Связь можно осуществить между элементами разных уровней.

Реляционная модель

Реляционная база данных представляется как совокупность таблиц, связанных друг с другом. Для наглядности приведу пример таблицы, которая может появиться в базе данных книжного магазина (табл. В1).

Таблица В1. *Книги*

Код	Название	Автор
00001	Основы JavaScript	Пол Уилтон
00002	Веб-дизайн	Дмитрий Кирсанов
00003	Введение в XML	Дэвид Хантер

Другая таблица (табл. В2) вполне может существовать в базе данных какого-нибудь компьютерного магазина.

Таблица В2. *Товары*

Артикул	Наименование	Модель	Цена (руб.)
M-0001	Модем ZyXEL	Omni 56K	1500
V-0033	Монитор LG	Flatron F700P	5000
A-0242	Колонки SVEN	SPS-611	800

В *части I* мы более подробно рассмотрим специфику таблиц, а пока обратим внимание на их некоторые особенности. Любая таблица состоит из строки заголовков столбцов и одной или более строк с данными. Эти столбцы и строки должны иметь следующие свойства:

- каждому столбцу таблицы присваивается имя, которое должно быть уникальным для этой таблицы;
- столбцы таблицы упорядочиваются слева направо (т. е. самый первый слева столбец таблицы — это столбец 1, второй слева — столбец 2, ..., самый правый — столбец *n*), хотя упорядоченными они являются только с точки зрения пользователя. Порядок, в котором определены имена столбцов, становится порядком, в котором в них должны вводиться данные;
- строки таблицы не упорядочены (их последовательность определяется лишь последовательностью ввода в таблицу);
- при выполнении операций с таблицей ее строки и столбцы можно обрабатывать в любом порядке безотносительно к их информационному содержанию;
- в поле на пересечении строки и столбца любой таблицы всегда имеется только одно значение данных и никогда не должно быть множества значений;
- всем строкам таблицы соответствует один и тот же набор столбцов, хотя любая строка может содержать пустые значения в некоторых столбцах;
- все строки таблицы обязательно отличаются друг от друга хотя бы одним значением, что позволяет однозначно определить любую строку такой таблицы.

Так почему же модель называется реляционной? Все просто, *отношение* (англ. relation) — это математический термин, используемый для обозначения неупорядоченного набора (совокупности) однотипных записей или таблиц определенного вида. Реляционные системы берут свое начало в математической теории множеств. Они были предложены сотрудником компании IBM Э. Ф. Коддом (E. F. Codd) в 1968 г.

Немного терминологии

Настало время определиться с терминами, используемыми при разработке БД. Основные термины и их описание:

- *сущность* (англ. entity) — то, что описано конкретной таблицей (пример: сущность "Книга");

- *поле* (англ. field) — свойство описываемой сущности (примеры: поле "Название", поле "Автор");
- *запись* (англ. record) — одна строка таблицы;
- *связь* (англ. relationship) — логическое отношение между двумя сущностями.

Итак, таблица описывает отдельную сущность. Сущность описывается полями. Между сущностями бывает организована связь (такие сущности называют связанными).

ЧАСТЬ I

Проектирование базы данных

Урок 1. Анализ предметной области, проблемы и пути их решения

Урок 2. Физическое проектирование таблиц, виды связей, нормализация

Урок 3. Типы данных и типы таблиц

Итак, теперь мы знаем, что такое база данных (БД) и для чего она может понадобиться. Настало время приступить к проектированию БД.

Рассмотрим этапы, которые нам необходимо пройти, прежде чем будут созданы БД и приложения, которые смогут использовать ее в своей работе:

1. Анализ предметной области.
2. Выбор модели данных (таблицы, связи).
3. Выбор СУБД.
4. Проектирование таблиц.
5. Проектирование приложений (модулей/программ), управляющих БД.
6. Реализация БД на компьютере.
7. Разработка средств администрирования БД (т. е. добавления, удаления, редактирования данных).
8. Эксплуатация БД.

Пункты 2 и 3 нами уже продуманы — мы используем реляционную модель данных, реализуемую в СУБД MySQL.

УРОК 1

Анализ предметной области, проблемы и пути их решения

Анализ предметной области — это анализ исходного набора данных. Предположим, перед нами стоит задача построения БД для какого-то магазина. Прежде всего, необходимо определить, какие данные нам понадобится хранить.

Допустим, нам нужно знать:

- дату продажи;
- фамилию, имя и отчество продавца;
- фамилию, имя и отчество покупателя;
- адрес покупателя;
- товар;
- сумму покупки (в рублях).

Представим все эти данные в виде таблицы (табл. 1.1).

На первый взгляд, данная таблица вполне нам подходит, т. к. содержит все необходимые данные. Если же рассмотреть ее более тщательно, то можно заметить, что такой способ хранения данных повлечет за собой ряд проблем и сложностей.

Во-первых, покупатель Петров встречается в таблице несколько раз — он постоянный клиент и довольно часто заходит в наш магазин. В итоге, при каждом его посещении нам приходится вносить одни и те же данные, например адрес его проживания. Следовательно, мы получаем избыточность данных — повторный ввод информации. Кроме этого, при очередном вводе данных о покупателе мы можем сделать ошибку, указав не тот номер дома или номер квартиры. Соответственно, у нас получится два разных покупателя.

Таблица 1.1. *База данных магазина, состоящая из одной таблицы*

Дата продажи	ФИО продавца	ФИО покупателя	Адрес покупателя	Товар	Сумма покупки (в рублях)
12.12.2004	Иванов Иван Иванович	Сергеев Андрей Васильевич	ул. Мира, д. 8, кв. 15	Телевизор SMK321	12000
12.12.2004	Васин Петр Сергеевич	Петров Иван Иванович	ул. Марата, д. 32, кв. 3	Лампа накаливания (60 Вт), удлинитель (10 м)	25
13.12.2004	Кузьмин Владимир Владимирович	Мухина Анна Викторовна	ул. Ленина, д. 21, кв. 14	Аудиосистема PS-911	4800
14.12.2004	Задорнов Виталий Петрович	Петров Иван Иванович	ул. Марата, д. 32, кв. 3	Чайник Tefal, сковорода Tefal	2340
14.12.2004	Соколов Сергей Александрович	Ванин Герасим Андреевич	пр. Науки, д. 45, кв. 26	Телефон Dect S115	1200
...	...	...	...	...	...

Во-вторых, если нам понадобится изменить данные, то придется менять их в нескольких местах. Допустим, в адрес закралась орфографическая ошибка или господин Петров решил его сменить. В этом случае нам придется изменить адрес столько раз, сколько упоминаний о нем есть в таблице. Напомню, что господин Петров заглядывает к нам довольно часто, чтобы сделать очередную покупку. И чем чаще это происходит, тем больше работы нам приходится делать, чтобы поддерживать непротиворечивость данных. Таким образом, получаем проблему обновления данных.

На этом список проблем не заканчивается — есть сложности и при удалении данных. Если мы захотим удалить из таблицы какого-нибудь покупателя, то вместе с покупателем будет удалено все, что с ним было связано. Например, будут удалены сведения о товарах, которые он когда-то приобретал.

Из всего этого следует, что структура хранения данных, представленная в табл. 1.1, нас совершенно не устраивает. Чтобы избавиться от всех этих сложностей, мы используем прием, называемый *нормализацией*.

УРОК 2

Физическое проектирование таблиц, виды связей, нормализация

Прежде чем приступить к нормализации, необходимо подробнее обсудить некоторые фундаментальные понятия реляционных баз данных. Данная модель состоит из трех основных элементов:

- сущность;
- атрибут;
- связь.

Сущности — это те вещи или объекты, данные о которых необходимо хранить. В модели данных сущность представляется в виде прямоугольника с заголовком. Заголовок является именем сущности или, если сказать проще, это название таблицы, хранящей данные. То есть сущность в БД — это таблица.

Атрибуты (поля таблицы) описывают те данные, которые нам нужно знать о сущности. Значение атрибута может быть числом, строкой символов, датой, временем или другим базовым значением данных.

Связями, как вы помните, называются логические взаимоотношения между сущностями. Но о связях мы поговорим позже.

В нашем примере база данных содержит ряд объектов: покупатель, продавец, наименование товара, дата продажи и т. д. Какие из них являются сущностями? Обратим внимание, что мы определили несколько видов данных (имя, адрес), относящихся к каждому покупателю. Без них невозможно описать покупателя. Поэтому покупатель является одним из объектов, которые мы хотели бы описать, т. е. сущностью. Давайте приступим к разработке модели данных с сущностью "Покупатель" (табл. 2.1).

Таблица 2.1. *Сущность "Покупатель"*

Покупатель

Примечание

Почему мы назвали нашу сущность "Покупатель", а не "Покупатели"? По общепринятому соглашению имя сущности должно быть в единственном числе, т. к. каждая сущность дает имя ее экземпляру. Например, "Петров Иван Иванович" является экземпляром сущности "Покупатель", а не "Покупатели".

У каждой сущности есть атрибуты, которые ее описывают. О покупателе нам могут понадобиться подробные сведения (табл. 2.2), например, если он покупает что-то в кредит.

Таблица 2.2. *Сущность "Покупатель" с атрибутами*

Покупатель		
ФИО покупателя	**Адрес**	**Телефон**
Сергеев Андрей Васильевич	ул. Мира, д. 8, кв. 15	362-23-32
Петров Иван Иванович	ул. Марата, д. 32, кв. 3	352-48-69
Ванин Герасим Андреевич	пр. Науки, д. 45, кв. 26	236-88-00
...	...	...

Вот теперь пришло время нормализации.

Впервые понятие "нормализация" ввел Е. Ф. Кодд, занимавшийся исследованиями в компании IBM. Целью нормализации является устранение из БД некоторых нежелательных характеристик. В частности, ставится задача избежать избыточности данных, приводящей к сложностям при операциях добавления, изменения и удаления данных.

Понятие нормализации включает пять нормальных форм. Мы рассмотрим три из них — этого достаточно, чтобы сделать структуру БД вполне работоспособной.

Первая нормальная форма (1НФ)

Сущность приведена к первой нормальной форме (1НФ), если все ее атрибуты имеют единственное значение. Если в каком-либо атрибуте есть повторяющиеся значения, то сущность не приведена к 1НФ. Посмотрев на нашу

базу данных (см. табл. 1.1), можно заметить, что в атрибуте "Товар" встречается больше одного значения, т. е. БД не находится в 1НФ. Это означает, что мы упустили, по крайней мере, еще одну сущность. Атрибут "Товар" описывает сведения о купленном товаре. Возможно, он тоже является сущностью. Давайте внесем его в нашу модель и добавим другие атрибуты (табл. 2.3).

Таблица 2.3. *Сущность "Товар" с атрибутами*

Товар		
Наименование	**Производитель**	**Цена (в рублях)**
Чайник	Tefal	1145
Телевизор SMK321	Sony	12 000
Телефон Dect S115	Dialon	1200
...	...	...

Теперь у нас есть сущность, приведенная к 1НФ.

Прежде чем переходить к рассмотрению второй нормальной формы, необходимо подробнее поговорить о связях между сущностями.

Ключи и связи

У каждого экземпляра сущности должен быть уникальный идентификатор, который будет однозначно определять каждую запись. Какой же из атрибутов может быть таким идентификатором? Нам необходимо выбрать атрибут, который подчиняется следующим правилам:

- он уникален для каждой записи (экземпляра сущности);
- для каждой записи он имеет значение, отличное от `NULL` (отсутствие данных);
- для каждой записи его значение не изменяется.

Такой атрибут называется *первичным ключом* (primary key). Если ни один из атрибутов не удовлетворяет этим правилам, то нужно ввести новый атрибут или создать первичный ключ из нескольких атрибутов. Рассмотрим в качестве примера таблицу, описывающую сущность "Покупка" (табл. 2.4).

В данной таблице ни один из атрибутов нельзя назначить ключевым, т. к. во всех полях данные могут повторяться. В каждом заказе может быть несколько товаров, каждый товар может присутствовать во многих заказах (исключение составляют магазины, торгующие уникальными товарами, но наш магазин таким не является). В этом случае можно задать *составной ключ*

(composite primary key), состоящий из полей "Номер заказа" и "Номер товара". Комбинация значений этих полей будет уникальной.

Таблица 2.4. *Сущность "Покупка" с атрибутами*

Покупка		
Номер заказа	**Номер товара**	**Количество единиц товара**
1	23	1
1	12	1
2	25	2
3	12	10
4	23	2

Выбор ключевого атрибута сущности играет очень важную роль при проектировании БД, т. к. он используется для моделирования связей. Если атрибут не удовлетворяет хотя бы одному из перечисленных правил, это может повлиять на всю модель данных.

Рассмотрим таблицу, описывающую покупателя (см. табл. 2.2). Можно попытаться выбрать в качестве ключевого поле "ФИО". Но что если покупатель изменит фамилию (это вполне возможно)? В этом случае нарушается третье правило для ключевых атрибутов. Кроме того, значения могут оказаться не уникальными — в каждом большом городе найдется несколько Петровых Иванов Ивановичей. Тогда нарушится первое правило. Наконец, возможно, что, вводя данные о покупателе, вы не будете знать его фамилию, имя и отчество. Тогда нарушается второе правило, которое гласит, что значение ключа должно быть отличным от `NULL`.

Исходя из этого, для таблицы "Покупатель" лучше задать новые поля (табл. 2.5).

Примечание

Для наглядности мы будем подчеркивать имя ключевого атрибута в каждой таблице.

Таблица 2.5. *Сущность "Покупатель" с ключевым атрибутом*

Покупатель			
<u>Номер покупателя</u>	**ФИО покупателя**	**Адрес**	**Телефон**

В эту таблицу мы ввели новое поле "Номер покупателя", которое будет уникально определять каждого конкретного покупателя нашего магазина.

Ключевые атрибуты сущностей (таблиц) позволяют моделировать связи, описывающие взаимоотношения между ними. Есть три типа связей:

- *один к одному* (1:1) — устанавливается между таблицами, если запись в первой таблице соответствует только одной записи во второй таблице;
- *один ко многим* (1:M) — устанавливается между таблицами, если запись в первой таблице соответствует одной или нескольким записям во второй таблице;
- *многие ко многим* (M:M) — устанавливается между таблицами, если одной записи в первой таблице соответствует несколько записей во второй таблице, а одной записи во второй таблице соответствует несколько записей в первой таблице.

Последний вид связи в реляционных таблицах не реализуется. Связь "1:1" встречается довольно редко. Если при проектировании таблиц вы столкнетесь с такой связью, следует еще раз внимательно рассмотреть свой проект. Возможно, сущности, между которыми установлена такая связь, являются на самом деле одной. В этом случае их следует объединить.

Рис. 2.1. Связь между таблицами "Заказ" и "Продавец"

Связь между таблицами, показанными на рис. 2.1, определена как "1:M". По номеру продавца мы можем узнать, какие заказы он обслуживал. Итак, таблицы "Продавец" и "Заказ" связаны по полю "Номер продавца".

Поле, которое указывает на запись в другой таблице, связанную с данной записью, называется *внешним ключом* (foreign key). Например, в таблице "Заказ" внешним ключом является поле "Номер продавца".

Иными словами, внешний ключ — это поле (или набор полей), значения которого совпадают с имеющимися значениями первичного ключа другой таблицы.

Ссылочная целостность

Итак, мы уже знаем, что значения в ключевом поле должны быть уникальными. Это является одним из правил *ссылочной целостности*. Некоторые СУБД могут контролировать уникальность первичных ключей — при попытке присвоить первичному ключу значение, уже имеющееся в другой записи, СУБД сгенерирует диагностическое сообщение. Это сообщение в дальнейшем может быть передано в приложение, при помощи которого пользователь управляет данными.

Если две таблицы являются связанными, то внешний ключ одной таблицы должен содержать только значения, имеющиеся среди значений первичного ключа другой таблицы. Допустим, если удалить запись из таблицы "Продавец", то в связанной с ней таблице "Заказ" будут присутствовать заказы, обслуженные несуществующим продавцом.

Вторая нормальная форма (2НФ)

Прочно связав таблицы, мы можем продолжить нормализацию.

Реляционная таблица приведена ко второй нормальной форме (2НФ), если она приведена к первой нормальной форме и ее неключевые поля полностью зависят от первичного ключа.

Таблица "Покупатель" (см. табл. 2.5) приведена ко второй нормальной форме. Но этого недостаточно — в этой таблице есть дополнительные зависимости. Например, если в таблице есть несколько покупателей с одним адресом (члены одной семьи), то при смене этого адреса потребуется изменить несколько записей.

Третья нормальная форма (3НФ)

Таблица приведена к третьей нормальной форме (3НФ), если она приведена ко второй нормальной форме и ни одно неключевое поле не зависит от других неключевых полей.

Чтобы перейти от второй нормальной формы к третьей, нужно выполнить следующие шаги:

1. Определить все поля (или группы полей), от которых зависят другие поля.
2. Создать новую таблицу для каждого такого поля (или группы полей) и переместить группы зависящих от него полей в эту таблицу. Поле (или группа полей), от которого зависят все остальные перемещенные поля, станет при этом первичным ключом новой таблицы.

3. Удалить перемещенные поля из исходной таблицы, оставив лишь те из них, которые станут внешними ключами.

Мы должны создать для адреса покупателя новую таблицу и переместить в нее поля из исходной таблицы (табл. 2.6 и 2.7).

Таблица 2.6. *Сущность "Покупатель" в 3НФ*

Покупатель		
Номер покупателя	**ФИО покупателя**	**Номер адреса**
1	Петров Иван Иванович	1
2	Сидоров Андрей Анатольевич	2
3	Ванин Алексей Сергеевич	3
4	Ванин Иван Алексеевич	3
...	...	...

Таблица 2.7. *Сущность "Адрес покупателя" в 3НФ*

Адрес покупателя		
Номер адреса	**Адрес**	**Телефон**
1	ул. Мира, д. 34, кв. 40	954-87-23
2	ул. Ленина, д. 123, кв. 23	941-85-25
3	ул. Московская, д. 12	654-78-22
...	...	...

Получившиеся таблицы связаны по полю "Номер адреса" (рис. 2.2).

У нас есть два покупателя, живущих по одному адресу (например, отец и сын). Если их адрес изменится, то сразу у обоих.

Итак, теперь мы знаем, как проектируются таблицы и устанавливаются связи между ними. Настало время для создания реальной БД.

В качестве предметной области возьмем "библиотеку". Нам необходимо провести анализ данной предметной области и разработать БД, которую мы будем использовать на протяжении всей книги. Для начала давайте подумаем, что нам необходимо хранить в нашей базе. Во-первых, мы должны хранить сведения о книге (табл. 2.8).

Рис. 2.2. Связь между таблицами "Покупатель" и "Адрес покупателя"

Таблица 2.8. *Сущность "Книга"*

Книга					
Номер книги	**Название**	**Год издания**	**Номер автора**	**Номер издательства**	**Номер раздела**
...	...	...	...	...	...

Номер книги будет уникально определять совокупность всех остальных ее атрибутов. Например, в нашу библиотеку поступила партия книг:

- автор: Томас Уилтон;
- название: "HTML 4.0. Справочник программиста";
- год издания: 2004;
- издательство: "Мир книг";
- раздел: "Компьютерная литература".

Допустим, партия включает 25 книг. Если бы номер присваивался каждому экземпляру книги, то нам пришлось бы вводить в таблицу "Книга" все эти данные двадцать пять раз. Поэтому номер книги будет определен непосредственно для партии этих книг. Но когда книга будет выдаваться на руки читателю, необходимо фиксировать номер конкретного экземпляра книги, а не номер партии. Поэтому необходимо хранить сведения о каждом экземпляре каждой книги (табл. 2.9).

Связь между этими таблицами будет осуществляться по номеру книги. То есть, зная номер экземпляра, мы сможем определить все остальные атрибуты книги.

Таблица 2.9. *Сущность "Экземпляр"*

Экземпляр	
Номер экземпляра	**Номер книги**
...	...

Поскольку наши писатели и поэты обычно не ограничиваются одним произведением, нам пришлось бы вводить сведения об авторе в таблицу "Книга" столько раз, сколько его книг есть в библиотеке. Поэтому в таблице "Книга" в качестве сведений об авторе зафиксирован номер автора (это куда лучше, чем вводить множество раз "Пушкин Александр Сергеевич"), а для хранения сведений об авторе мы создадим отдельную таблицу (табл. 2.10).

Таблица 2.10. *Сущность "Автор"*

Автор	
Номер автора	**ФИО автора**
...	...

Поле "ФИО автора" будет включать в себя его фамилию, имя и отчество. С таблицей "Книга" связь будет осуществляться по полю "Номер автора".

Кроме этого, нам нужно создать еще две таблицы, в которых будут храниться сведения об издательствах (табл. 2.11) и разделах (жанрах) (табл. 2.12).

Таблица 2.11. *Сущность "Издательство"*

Издательство	
Номер издательства	**Название издательства**
...	...

Таблица 2.12. *Сущность "Раздел"*

Раздел	
Номер раздела	**Название раздела**
...	...

Таблицы "Издательство" и "Раздел" будут связаны с таблицей "Книга" по полям "Номер издательства" и "Номер раздела" соответственно.

Итак, чтобы полностью описать книгу, нам потребовалось создать пять таблиц. Благодаря этому мы избежим избыточности данных. Но это еще не все. Нам нужно знать данные о читателях, посещающих библиотеку. Давайте создадим таблицу "Читатель" (табл. 2.13).

Таблица 2.13. *Сущность "Читатель"*

Читатель		
Номер читателя	**ФИО читателя**	**Номер адреса**
...	...	...

Также нам понадобится таблица с адресами читателей, чтобы избежать проблем в том случае, если будет несколько читателей, живущих по одному адресу (табл. 2.14).

Таблица 2.14. *Сущность "Адрес читателя"*

Адрес читателя		
Номер адреса	**Адрес**	**Телефон**
...	...	...

И, наконец, мы должны знать, какие книги находятся на руках у читателей. Для этого создадим новую таблицу и назовем ее "Абонемент" (табл. 2.15).

Таблица 2.15. *Сущность "Абонемент"*

Абонемент				
Номер записи	**Номер читателя**	**Номер экземпляра**	**Дата выдачи**	**Дата возврата**
...	...	...	...	...

Итак, мы полностью описали нашу БД, осталось только проставить связи между всеми таблицами (рис. 2.3).

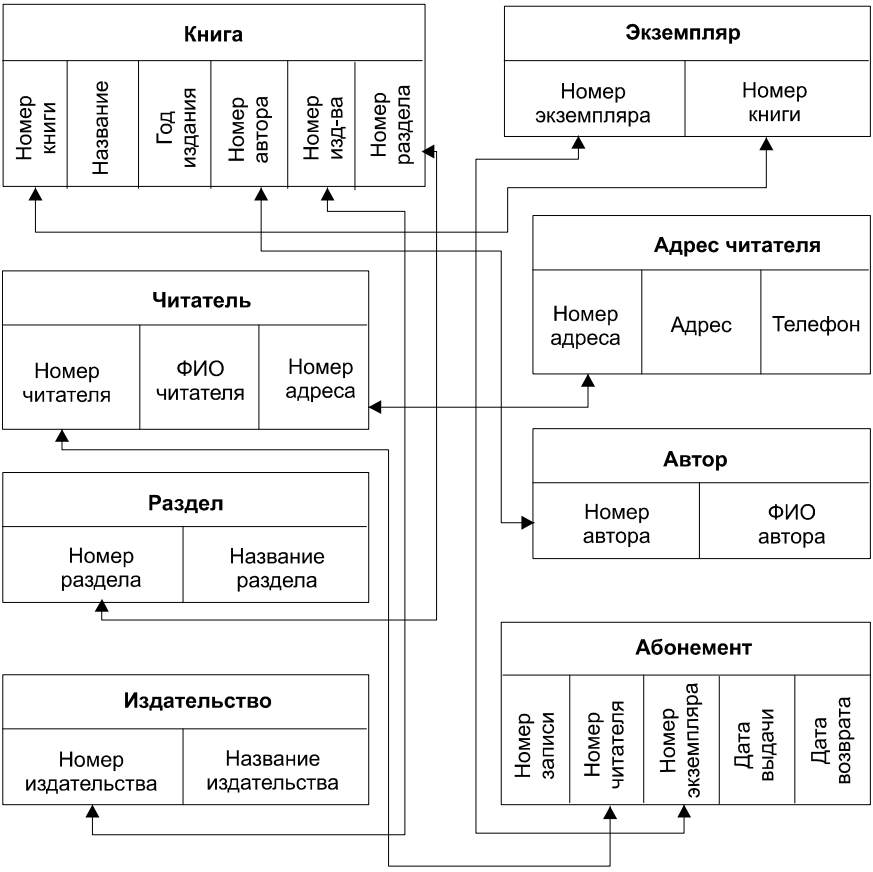


Рис. 2.3. Связи между сущностями в БД "Библиотека"

УРОК 3

Типы данных и типы таблиц

В таблицах, которые мы создали на предыдущем уроке, будут храниться различные данные (числа, строки, даты и т. д.). Пришло время поговорить о типах данных, используемых при создании таблиц. Для каждого поля таблицы необходимо описать тип данных, который оно будет содержать. Вы должны выбрать правильный (корректный) тип данных для каждого поля таблицы. Например, если вы введете числовое значение в текстовое поле, то не сможете в дальнейшем использовать это значение для математических вычислений.

Типы данных, поддерживаемые СУБД MySQL, можно разделить на четыре группы:

- числа (Numbers);
- текст (Text);
- дата и время (Date and Time);
- списки (Defined group).

В табл. 3.1 приведены все типы данных для каждой из четырех групп.

Таблица 3.1. *Типы данных в MySQL*

Группа типов	Типы
Числа	TINYINT, SMALLINT, MEDIUMINT, INT, BIGINT FLOAT, DOUBLE NUMERIC, DECIMAL
Текст	CHAR VARCHAR TINYTEXT, MEDIUMTEXT, TEXT, LONGTEXT TINYBLOB, MEDIUMBLOB, BLOB, LONGBLOB

Таблица 3.1 *(окончание)*

Группа типов	Типы
Дата и время	DATETIME TIMESTAMP DATE TIME YEAR
Списки	ENUM SET

Числа

СУБД MySQL различает целые числа (например 5 и 46) и вещественные числа (с дробной частью, например 123,5). Целые числа можно представить в десятичном или шестнадцатеричном формате. Также СУБД может работать с числами в экспоненциальной форме (например 1,34E+14 или 2,53e–3). Числовые типы данных ограничиваются диапазоном представляемых величин и размером в байтах. Вас не должно пугать их разнообразие. Я постараюсь максимально подробно описать каждый из типов данных.

Целые числа (типы *TINYINT*, *SMALLINT*, *MEDIUMINT*, *INT*, *BIGINT*)

Поля типов TINYINT, SMALLINT, MEDIUMINT, INT, BIGINT могут содержать положительные или отрицательные целые числа, а также ноль (0). Таким образом, любой из этих типов может быть знаковым (signed) или беззнаковым (unsigned).

- Знаковое поле может содержать положительные и отрицательные числа, а также ноль. Для обозначения отрицательных значений числа предваряются знаком "минус" (–).
- Беззнаковое поле может содержать только положительные числа и ноль.

По умолчанию поле будет знаковым. Если вы хотите, чтобы поле содержало только положительные числа и ноль, необходимо добавить атрибут UNSIGNED после описания типа поля.

```
field_name INT(length) UNSIGNED
```

Параметры: `field_name` — имя создаваемого поля, а `length` — размер поля в символах (количество знакомест). Если при объявлении типа параметр `length` не указан, будет использовано значение по умолчанию. Например, объявив

```
field_name INT
```

мы получим

```
field_name int(11)
```

Но, помните, что параметр `length` влияет лишь на количество отображаемых символов, но никоим образом *не влияет* на объем памяти, необходимый для хранения данных этого типа.

Давайте рассмотрим целые типы более подробно.

- `TINYINT(length)` — целое число от 0 до 255 для беззнаковых и от –128 до 127 для знаковых. Таким образом, максимальный размер данных, вносимых в поле такого типа, будет равен одному байту. Параметр `length`, как вы помните, ограничивает размер поля в символах. Если, например, для поля объявлен тип `TINYINT(2)`, то данное поле будет отображать значения от –9 до 99 для беззнаковых и от 0 до 99 для знаковых.
- `SMALLINT(length)` — целое число от 0 до 65 535 для знаковых и от –32 768 до 32 767 для беззнаковых.
- `MEDIUMINT(length)` — целое число от 0 до 16 777 215 для беззнаковых и от –8 338 608 до 8 338 607 для знаковых.
- `INT(length)` — целое число от 0 до 4 294 967 295 для беззнаковых и от –2 147 483 648 до 2 147 483 647 для знаковых.
- `BIGINT(length)` — целое число в пределах от 0 до 18 446 744 073 709 551 615 для беззнаковых и от –9 223 372 036 854 775 808 до 9 223 372 036 854 775 808 для знаковых. Под поле с таким типом будет выделено 8 байт.

Это все целочисленные типы данных, поддерживаемые СУБД MySQL. Если в числовое поле записать величину, превышающую максимальное возможное значение для данного типа, то MySQL ограничит значение до соответствующей граничной точки допустимого интервала и сохранит результат вместо исходного значения. То есть если, например, в поле, тип которого `TINYINT`, записать число –300, то в результате в него будет занесено значение –128 (предельное для данного типа). Аналогично, если попытаться записать в такое поле число 300, то в итоге получим 127.

Если поле `TINYINT` задано как беззнаковое, то при попытке записать значения –300 и 300 мы получим 0 и 255 соответственно.

Числа с плавающей точкой (типы *DOUBLE* и *FLOAT*)

Эти типы очень удобны для хранения всевозможных научных данных.

Тип `FLOAT(length, decimal)` — число с плавающей точкой обычной (одинарной) точности. Значения допустимы в пределах от –3.402823466E+38 до –1.175494351E–38 и от 1.175494351E–38 до 3.402823466E+38. Аргумент `length` задает число выводимых на экран (для пользователя) знаков, а `decimal` — количество разрядов, следующих за десятичной точкой. Например, поле типа `FLOAT(9,2)` будет отображать значения как, например:

```
999999.99
```

Если указан атрибут `UNSIGNED`, то отрицательные значения недопустимы. Также следует отметить, что использование типа данных `FLOAT` может привести к неожиданным проблемам, поскольку все вычисления в MySQL выполняются с удвоенной точностью.

Тип `DOUBLE(length, decimal)` — число с плавающей точкой удвоенной точности (8 байт). Допустимы значения от –1.7976931348623157E+308 до –2.2250738585072014E–308, 0 и от 2.2250738585072014E–308 до 1.7976931348623157E+308. Если указан атрибут `UNSIGNED`, то отрицательные значения недопустимы. Для типа `DOUBLE` есть синонимы — `DOUBLE PRECISION` и `REAL`.

Тип *NUMERIC* (*DECIMAL*)

Тип `NUMERIC(length, decimal)` или `DECIMAL(length, decimal)` — число с плавающей точкой, хранящееся в виде строки. Это может понадобиться для сохранения точности представления этих величин в десятичном виде. Максимальный интервал значений типа `DECIMAL` тот же, что и для типа `DOUBLE`, но действительный интервал для конкретного поля `DECIMAL` может быть ограничен выбором значений атрибутов `length` и `decimal`.

Если указан атрибут `UNSIGNED`, отрицательные значения недопустимы. Если параметр `decimal` не указан, его значение по умолчанию равно 0. Если не указан параметр `length`, его значение по умолчанию равно 10.

Требования к памяти для числовых типов приведены в табл. 3.2.

При выборе числового типа для поля подумайте о том, из какого диапазона оно может принимать значения. Обязательно выбирайте наименьший из приемлемых типов. Если выбирать большие типы, можно получить таблицу довольно большого размера. Например, для хранения возраста человека оптимальным будет использование типа `TINYINT`.

Таблица 3.2. *Требования к памяти для числовых типов*

Тип	Число байт, необходимых для хранения значения
TINYINT	1
SMALLINT	2
MEDIUMINT	3
INT	4
BIGINT	8
FLOAT	4
DOUBLE	8
DECIMAL(length, decimal) NUMERIC(length, decimal)	length + 2 (при decimal > 0) length + 1 (при decimal = 0) decimal + 2 (при length < decimal)

Текст

Текстовое поле может хранить любые символы. Чаще всего используются буквы, но символы пунктуации и числа также допустимы (например, в значении адреса — 'ул. Мира, д. 35, кв. 40'). Но если вы введете в текстовое поле число, то в дальнейшем его нельзя будет использовать в математических функциях. MySQL поддерживает четыре основных текстовых типа — CHAR, VARCHAR, TEXT и BLOB.

Тип *CHAR*

Тип CHAR(length) — строка фиксированной длины. Длина (параметр length) может принимать любое значение от 1 до 255 (в MySQL версии 3.23 — от 0 до 255). Любой текст меньшей длины будет дополнен пробелами в конце строки. Все конечные пробелы, добавленные MySQL или пользователями, будут удалены при выводе значения. Любой текст большей длины будет обрезан до заданной.

MySQL по умолчанию считает текст нечувствительным к регистру букв. Если требуется сделать текст чувствительным к регистру букв, то при описании поля нужно добавить атрибут BINARY:

```
field_name CHAR(length) BINARY
```

В этом случае значения в поле сортируются и сравниваются с учетом регистра в соответствии с порядком символов в ASCII-таблице на том компьютере, где работает сервер MySQL.

Тип *VARCHAR*

Тип VARCHAR(length) — строка переменной длины. Длина (параметр length) может принимать значение от 1 до 255 символов. Значение типа VARCHAR занимает столько байт, сколько символов ввел пользователь, плюс один байт для записи длины (но не больше 255). Концевые пробелы при сохранении удаляются. Строка, хранимая в поле типа VARCHAR, также нечувствительна к регистру букв. Здесь, если требуется сделать строку чувствительной к регистру букв, тоже необходимо применять атрибут BINARY, как в случае с типом CHAR.

В принципе, типы данных CHAR и VARCHAR очень схожи между собой, разница заключается в способе хранения и извлечения (табл. 3.3).

Таблица 3.3. *Отличия типов CHAR и VARCHAR*

Исходное значение	CHAR(4)		VARCHAR(4)	
	Значение	Требуется байт	Значение	Требуется байт
''	' '	4	''	1
'ab'	'ab '	4	'ab'	3
'abcd'	'abcd'	4	'abcd'	5
'abcdef'	'abcd'	4	'abcd'	5

Тип CHAR предпочтительнее использовать в том случае, если хранимые значения имеют одинаковую длину (VARCHAR требует дополнительный байт для хранения длины) или длина отличается незначительно. В таблицах типа MyISAM (тип таблиц по умолчанию) эффективнее обрабатываются строки фиксированной длины.

Извлеченные из столбцов CHAR(4) и VARCHAR(4) величины в каждом случае будут одними и теми же, поскольку при извлечении концевые пробелы из поля типа CHAR удаляются.

В некоторых случаях СУБД MySQL без уведомления изменяет тип поля при создании таблицы или изменении ее структуры.

- Поле типа VARCHAR меньше четырех символов длиной преобразуется в поле типа CHAR.
- В случае если таблица содержит любые поля переменной длины (VARCHAR, TEXT или BLOB), то все поля типа CHAR больше трех символов длиной преобразуются в поля типа VARCHAR. Это в любом случае не повлияет на использование полей. В MySQL поле типа VARCHAR представляет собой просто

другой способ хранения данных. MySQL выполняет данное преобразование, потому что оно позволяет сэкономить память и сделать табличные операции более быстрыми.

Типы *TEXT* и *BLOB*

Тип BLOB представляет собой двоичную строку переменной длины, чувствительную к регистру букв. У данного типа есть четыре модификации (TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB), которые отличаются максимальной длиной хранимого значения.

Тип TEXT — текстовая строка, нечувствительная к регистру букв. У данного типа тоже есть четыре модификации, соответствующие модификациям BLOB и имеющие ту же максимальную длину и требования к памяти.

Требуемая память для текстовых типов указана в табл. 3.4.

Таблица 3.4. *Требуемая память для текстовых типов*

Тип	Требуемая память для хранения значения
CHAR(length)	length байт
VARCHAR(length)	length + 1 байт
TINYBLOB, TINYTEXT	length + 1 байт (length < 256)
BLOB, TEXT	до 64 Кбайт
MEDIUMBLOB, MEDIUMTEXT	до 16 Мбайт
LONGBLOB, LONGTEXT	до 4 Гбайт

Если данные, вносимые в поля типа TEXT или BLOB, превышают максимальную допустимую длину, то они будут усечены соответствующим образом.

В принципе, поле типа TEXT можно рассматривать как поле типа VARCHAR без ограничения длины (аналогично, BLOB — как поле типа VARCHAR BINARY). Различия при этом следующие:

- в полях с типом BLOB и TEXT не удаляются концевые пробелы, как это делается для полей типа VARCHAR;
- для столбцов BLOB и TEXT не может быть задан атрибут DEFAULT — значения величин по умолчанию;
- поля BLOB и TEXT могут индексироваться в версии MySQL версии 3.23.2 и более.

Эти типы полей полезны, когда требуется хранить большие объемы данных, например всевозможные электронные документы или изображения.

Дата и время

В MySQL поддерживаются следующие типы данных для хранения даты и времени: `YEAR`, `DATE`, `TIME`, `DATETIME`, `TIMESTAMP`. Каждый из этих типов данных имеет интервал допустимых значений. В том случае если пользователь введет в такое поле недопустимое значение, оно будет заменено на `'0'`. Например, при попытке ввести в поле `DATE` значение `'300'` или `'hallo'` мы получим в результате `'0000-00-00'`. Следует отметить, что MySQL позволяет хранить некоторые не вполне верные значения даты, например `'1998-02-31'`. В этом случае проверкой вводимых данных должно заниматься приложение, в котором работает пользователь. MySQL выполняет только минимальную проверку вводимых данных. СУБД проверяет, находится ли месяц в интервале 0—12 и день в интервале 0—31. Данные интервалы начинаются с 0 для того, чтобы обеспечить для MySQL возможность хранить в столбцах `DATE` или `DATETIME` даты, в которых день или месяц равен нулю, например, `'1980-00-00'` или `'1980-01-00'` (это удобно в случае, если в поле вносится дата рождения, но известны лишь год и месяц).

Даты должны задаваться в порядке *год-месяц-день* (например, `'1980-09-21'`), а не в том порядке, как мы их обычно записываем (например, `'21-09-1980'`).

MySQL поддерживает двузначный ввод года:

- значения в диапазоне от 0 до 69 будут интерпретированы и сохранены как года от 2000 до 2069;
- значения в диапазоне от 70 до 99 будут интерпретированы и сохранены как года от 1970 до 1999.

Тип *YEAR*

Тип `YEAR` — это однобайтный тип данных для хранения значения года. Данные можно вводить в формате *YYYY*, либо *YY*. Диапазон значений года — от 1901 до 2155. Все недопустимые значения преобразуются в 0. Если не требуется хранить полную дату, то наиболее эффективным будет данный тип (например, для хранения значения "год рождения" или "год выпуска изделия").

Тип *TIME*

Тип `TIME` — тип данных для хранения времени. Значение всегда выводится в формате *HH:MM:SS*. Ввод данных может быть осуществлен в следующих форматах:

- *HH:MM:SS*;
- *HHMMSS* — строка интерпретируется как дата (например, ввод значения `'122210'` будет воспринят как `'12:22:10'`); если ввести недопустимую величину (например, `'129801'`), она преобразуется в `'00:00:00'`;
- *HHMM* — значение секунд равно нулю;
- *HH* — значения минут и секунд обнуляются.

Величины также можно вводить как число, например, величина `'100122'` будет воспринята как `'10:01:22'` (формат *HHMMSS*). Если величина типа `TIME` представлена как строка с разделителями, то необязательно вводить по два разряда для значений часов, минут и секунд меньше 10. То есть величина `'9:5:7'` эквивалентна `'09:05:07'`.

Примечание

Если значение, вводимое в поле типа `TIME`, не содержит разделителей, то оно будет воспринято не как время дня, а как истекшее время. Пример: вы вводите величину `'1225'`, подразумевая 12 часов 25 минут дня (`'12:25:00'`), а MySQL понимает ее как `'00:12:25'` (12 минут 25 секунд).

Значения в поле `TIME` могут лежать в пределах от `'-838:59:59'` до `'838:59:59'`. Почему же значения часов настолько велики? Дело в том, что тип `TIME` может использоваться не только для хранения времени дня, но и для представления истекшего времени или интервала между какими-нибудь событиями, который может превышать 24 часа или быть отрицательным.

Значения, лежащие за пределами допустимого интервала, будут соответствующим образом усечены. Например, введенные значения `'-900:00:00'` и `'900:00:00'` будут преобразованы в `'-838:59:59'` до `'838:59:59'` соответственно. Недопустимые значения будут приведены к виду `'00:00:00'`.

Типы *DATE*, *DATETIME* и *TIMESTAMP*

Тип данных `DATE` предназначен для хранения значений года, месяца и дня. Значения всегда выводятся в формате *YYYY-MM-DD*. Ввод данных может осуществляться в следующих форматах:

- *YYYY-MM-DD*;
- *YY-MM-DD*;
- *YYMMDD*;
- *YYMMDDHHMMSS* (часть, хранящая время, игнорируется).

Для значений, представленных как строки и содержащих разделительные знаки между частями даты, необязательно указывать два разряда для значений месяца или дня меньше 10. Так, величина `'1980-5-9'` эквивалентна вели-

чине `'1980-05-09'`. Данный тип поддерживает диапазон от `'1000-01-01'` до `'9999-12-31'`. Недопустимое значение преобразуется к виду `'0000-00-00'`.

При задании значения даты следует учитывать некоторые особенности MySQL.

- Упрощенный формат, который допускается для значений, заданных строками, может ввести в заблуждение. Например, такое значение, как `'10:11:12'`, благодаря разделителю `':'` могло бы оказаться значением времени, но, используемое в контексте даты, оно будет интерпретировано как год `'2010-11-12'`. В то же время величина `'12:55:35'` будет преобразована в `'0000-00-00'`, т. к. значение `'55'` является недопустимым значением для месяца.
- MySQL выполняет только начальную проверку подлинности даты: дни 00—31, месяцы 00—12, года 1000—9999. Любая дата вне этого диапазона преобразуется в `'0000-00-00'`. Следует также отметить, что хранить неверные даты, такие как `'2004-02-31'`, не запрещается. Чтобы убедиться в достоверности даты, необходимо выполнять проверку в приложении, с которым работает пользователь.
- Значения года, представленные двумя разрядами, допускают неоднозначное толкование, т. к. неизвестно столетие. MySQL интерпретирует двухразрядные значения года по следующим правилам:
  - значения года в интервале 00—69 преобразуются в 2000—2069;
  - значения года в интервале 70—99 преобразуются в 1970—1999.

Тип данных `DATETIME` используется для полей, хранящих информацию как о дате, так и о времени. Вывод значений СУБД MySQL осуществляется в формате *YYYY-MM-DD HH:MM:SS*. Для данного типа поддерживается диапазон значений от `'1000-01-01 00:00:00'` до `'9999-12-31 23:59:59'`.

Тип данных `TIMESTAMP(length)` можно использовать для автоматической записи текущих даты и времени при выполнении операций `INSERT` (ввод данных) и `UPDATE` (обновление данных). Дата начинается с 19700101000000 и заканчивается где-то в далеком 2037 году. Начальное значение — это "начало эпохи" ОС UNIX. При внесении значения `NULL` в поле запишется текущее время для сервера. Также можно внести значение вручную. Необязательный параметр `length` (длина) определяет формат вывода значения поля на экран.

Примечание

Понятие длины довольно условно. Параметр `length` задает длину отображения (т. е. количество символов для отображения значений столбца). Но если длина отображаемого значения содержит больше символов, чем задано в параметре `length`, то будет отображено все значение, а не его часть, обрезанная до `length` символов.

Полный формат TIMESTAMP составляет 14 десятичных разрядов, но можно создавать поля типа TIMESTAMP и меньшей длины (табл. 3.5).

Таблица 3.5. *Зависимость формата вывода от длины поля типа TIMESTAMP*

Тип поля	Формат
TIMESTAMP(14)	*YYYYMMDDHHMMSS*
TIMESTAMP(12)	*YYMMDDHHMMSS*
TIMESTAMP(10)	*YYMMDDHHMM*
TIMESTAMP(8)	*YYYYMMDD*
TIMESTAMP(6)	*YYMMDD*
TIMESTAMP(4)	*YYMM*
TIMESTAMP(2)	*YY*

Обычно значение параметра length устанавливают равным 6, 8, 12 или 14. При создании таблицы вы можете задать произвольную длину поля, но если она окажется равной 0 или превысит 14, то будет использоваться значение 14. Нечетные значения параметра length будут приведены к ближайшему большему четному числу. Недопустимые значения, внесенные в поле типа TIMESTAMP, будут преобразованы в 0. Тип TIMESTAMP можно использовать для хранения даты создания или даты последней модификации чего-либо. Если вам нужно хранить и то и другое, соответственно создайте два поля типа TIMESTAMP. Но, помните, что в этом случае автоматически будет изменяться только первое из них. Соответственно поле, хранящее дату модификации, должно быть первым, а поле с датой создания — вторым.

Внутренний размер данных, хранящихся в поле TIMESTAMP, не зависит от длины выводимого значения и всегда равен 4 байтам (табл. 3.6).

Величину одного типа даты можно в ряде случаев присвоить объекту другого типа даты, но в этом случае возможны некоторые изменения значения или потеря информации:

- если присвоить значение типа DATE объекту типа DATETIME или TIMESTAMP, то в результирующем значении временна́я часть будет установлена в '00:00:00', т. к. значение типа DATE не содержит информации о времени;
- если присвоить значение типа DATETIME или TIMESTAMP объекту типа DATE, то временна́я часть в результирующем значении будет удалена, т. к. тип DATE не включает информацию о времени.

Таблица 3.6. *Требования к памяти для типов даты и времени*

Тип	Число байт, необходимых для хранения значения
YEAR	1 байт
DATE	3 байта
TIME	3 байта
DATETIME	8 байт
TIMESTAMP	4 байта

Списки

Имеющиеся в MySQL типы списков: ENUM (перечисление), SET (множество). Давайте рассмотрим их более детально.

Тип *ENUM* (перечисление)

Поле типа ENUM может содержать одно значение из предварительно заданного набора (например, ENUM('яблоки', 'бананы', 'апельсины')). Данные могут быть введены как одно из текстовых значений набора или как число, соответствующее одному из элементов набора (номер первого элемента — 1). Этим значением также может быть пустая строка ('') или NULL. Для этого должны быть выполнены следующие условия.

- Если вводится некорректное значение в поле типа ENUM (т. е. строка, не присутствующая в списке допустимых), то вставляется пустая строка (это указывает на ошибочное значение). Данная строка отличается от "обычной" пустой строки в том, что она имеет цифровое значение, равное 0.
- Если значение в поле типа ENUM определяется как NULL, то NULL является допустимым значением такого поля, а также значением по умолчанию. Если значение в поле типа ENUM определяется как NOT NULL, то значением по умолчанию является первый элемент из списка допустимых значений.

Набор может содержать до 65 535 различных элементов. Если в наборе меньше 256 элементов, поле займет один байт, иначе его размер будет равен двум байтам (см. табл. 3.8).

Каждое из допустимых значений имеет индекс.

- Значения из списка допустимых величин, определенных при создании таблицы, индексируются, начиная с 1.
- Индексом пустой строки является 0.
- Индексом значения NULL является NULL.

Начиная с MySQL версии 3.23.51 при создании таблицы из значений этого поля автоматически удаляются оконечные пробелы.

Регистр букв не играет роли, когда вы делаете вставку в поле типа `ENUM`. Однако регистр букв в значениях, получаемых из этого столбца, совпадает с регистром букв в соответствующем значении, заданном во время создания таблицы.

Если вы вставляете число в поле типа `ENUM`, то это число воспринимается как индекс, и в таблицу записывается соответствующее этому индексу значение из перечисления.

Примечание

В поле типа `ENUM` не рекомендуется сохранять числа, т. к. это может привести к излишней путанице.

Значения перечисления сортируются в соответствии с их индексами. Например, для `ENUM('яблоки', 'бананы')` значение "яблоки" в отсортированном выводе будет упомянуто раньше, чем значение "бананы", а для `ENUM('бананы', 'яблоки')` значение "бананы" появится раньше значения "яблоки."

Пустые строки возвращаются перед непустыми строками, а значения `NULL` выводятся в самую первую очередь.

Тип *SET* (множество)

Поле типа `SET` может содержать любое количество предварительно заданных значений. Данные могут быть заданы как список разделенных запятыми значений (например, `SET('синий', 'зеленый', 'красный', 'желтый')`). Как следствие, сами элементы множества не могут содержать запятых.

Например, столбец, определенный как `SET('один', 'два') NOT NULL`, может принимать такие значения:

- `''`
- `'один'`
- `'два'`
- `'один,два'`

В одном наборе `SET` может быть до 64 различных элементов.

Начиная с MySQL версии 3.23.51 при создании таблицы из значений множества `SET` автоматически удаляются концевые пробелы.

MySQL сохраняет значения `SET` в численном виде, где младший бит сохраненной величины соответствует первому элементу множества. Если вы делаете выборку столбца `SET` в числовом контексте, полученное значение со-

держит соответствующим образом установленные биты, создающие значение столбца.

Если делается вставка в поле типа SET, биты, установленные в двоичном представлении числа, определяют элементы множества. Например, поле было определено как SET('a', 'b', 'c', 'd'). Представление элементов поля SET (установленные биты) для этого случая приведено в табл. 3.7.

Таблица 3.7. *Представление элементов поля SET*

Элемент	Числовое значение	Двоичное представление
a	1	0001
b	2	0010
c	4	0100
d	8	1000

Для значения, содержащего больше одного элемента, не играет никакой роли порядок, в котором эти элементы перечисляются в момент вставки значения. Также неважно, сколько раз встречается то или иное значение. Когда это значение выбирается, каждый элемент будет присутствовать только единожды, и элементы будут перечислены в том порядке, в котором они перечисляются при создании таблицы.

Например, если поле определено как SET('a', 'b', 'c', 'd'), тогда 'a,d', 'd,a' и 'd,a,a,d,d' будут представлены как 'a,d'.

Если вы вставляете в поле SET некорректную величину, это значение будет проигнорировано.

Значения в поле SET сортируются в соответствии с числовым представлением. NULL-значения идут в первую очередь.

Размер объекта SET определяется количеством отличающихся элементов множества. Если это количество равно N, то размер объекта вычисляется по формуле $(N + 7) / 8$ и полученное число округляется до 1, 2, 3, 4 или 8 байтов. В множество SET может входить максимум 64 элемента.

Размер объекта ENUM определяется количеством различных перечисляемых величин. Один байт используется для перечисления до 255 возможных величин. Используя два байта, можно перечислить до 65 535 величин.

Требования к памяти для типов ENUM и SET приведены в табл. 3.8.

Таблица 3.8. *Требования к памяти для типов* `ENUM` *и* `SET`

Тип	Число байт, необходимых для хранения значения
`ENUM('значение1', 'значение2', ...)`	1 или 2, в зависимости от количества перечисленных значений (максимум 65 535)
`SET('значение1', 'значение2', ...)`	1, 2, 3, 4 или 8 в зависимости от количества перечисленных элементов множества (максимум 64)

ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ

1. Какие четыре базовых текстовых типа поддерживает MySQL?
2. Сколько памяти требуется для хранения значения типа `SMALLINT`?
3. Каков диапазон допустимых значений для типа `YEAR`?

Итак, мы рассмотрели все типы данных, поддерживаемые СУБД MySQL. Теперь нам необходимо определить типы для каждого поля всех таблиц, имеющихся в учебной базе "Библиотека". Но прежде следует сказать несколько слов о правилах выбора типа данных для полей, а также об именовании таблиц и полей.

Выбор типа данных для поля

Поговорим немного о том, чем нужно руководствоваться при выборе типа поля. При создании баз данных наиболее часто используются строковые поля, которые хранят любые значения. Может возникнуть вопрос — а почему бы не сделать все поля строковыми? Можно поступить и так, но это создаст довольно много проблем. Во-первых, при этом неэффективно используется память, во-вторых, числа и строки обрабатываются по-разному. Например, при сортировке или операциях сравнения нам потребуется каждый раз преобразовывать строковое поле в числовой формат. Все это окажет влияние на общую производительность системы.

На первый взгляд, все очевидно: числа хранятся в числовых полях, строки — в строковых, дата и время — в полях календарного типа. Но на самом деле не все так просто, есть множество нюансов, о которых следует помнить. Одно дело, если вы разрабатываете БД для своих нужд, другое — если для заказчика. Во втором случае вы должны узнать довольно точно, в каком виде предстоит хранить и выводить значения.

Например, если требуется хранить параметры каких-то товаров (ширину, высоту и т. д.), то можно это сделать различными способами. Допустим, вы мо-

жете хранить высоту, как строку `'2/20'`, т. е. 2 метра 20 сантиметров. Это значение легко для понимания, но его нельзя использовать для математических операций. Можно создать два поля, одно будет хранить значение в метрах, другое в сантиметрах. Так вы сможете использовать значения для различных вычислений, но все же это не настолько удобно, как если бы параметр хранился в одном поле. Значит, надо создать одно поле и хранить высоту товара в миллиметрах. Возможно, это не очень понятно для пользователя, но зато удобно для хранения в БД.

Что же касается восприятия пользователем, то любые данные при выводе можно отформатировать и представить в более понятной форме. Форматированием должно заниматься приложение, с которым работает пользователь (например, web-интерфейс, написанный вами на языке PHP).

Другой пример. Если вам нужно хранить цену товара, то здесь тоже есть несколько вариантов. Типы `FLOAT` и `DOUBLE` не очень годятся на эту роль, т. к. они округляются, а все, что касается денег, требует особой точности. Тогда можно хранить величины в полях типа `DECIMAL(length,2)`, выбрав подходящую длину (значение параметра `length`). В этом случае значения не округляются, но операции со строками, как правило, менее эффективны, чем с числами. Есть еще один вариант: можно хранить одно число — значение в копейках, но в этом случае при вводе/выводе потребуется привести его к необходимому формату вывода.

После выбора типа данных надо решить, какой диапазон значений будет храниться, — будут ли значения находиться в пределах от 0 до 100 или достигать миллионов. Вы, конечно, можете использовать самый большой тип (например, `BIGINT`, для числовых значений) и ни о чем не беспокоиться. Однако следует использовать минимальный из возможных типов, для того чтобы сократить объем дискового пространства, занимаемого таблицей. Это позволит повысить производительность (помните, что на жестком диске сервера вы не одиноки). Если диапазон уж совсем неизвестен, то выбирайте тип `BIGINT`. Если выбранный тип оказался слишком маленьким, и его не хватает для хранения величин, то это не так страшно — впоследствии все можно поправить с помощью оператора `ALTER TABLE`. Вы также должны помнить о том, что числовые поля по умолчанию допускают ввод отрицательных чисел и вмещают в положительном диапазоне только половину возможных значений. Чтобы решить эту проблему, используйте атрибут `UNSIGNED`.

При создании строковых полей нужно также определиться с длиной хранимого значения. Если строки не будут превышать 256 символов, то подойдет тип `CHAR`, `VARCHAR`, `TINYTEXT` или `TINYBLOB`. Для строк большей длины используйте тип `TEXT` или `BLOB`. Если строка — это набор фиксированных величин, то используйте тип `SET` или `ENUM`, — они позволяют выполнять цифровые операции и экономят память.

Имена баз данных, таблиц и полей

Для имен баз данных, таблиц и полей используются одни и те же правила (они были изменены начиная с версии MySQL 3.23.6).

Имена баз данных

В MySQL есть только два правила именования баз данных — имя БД может иметь длину до 64 символов и может содержать любые символы, кроме прямого слэша (/), обратного слэша (\) и точки (.).

Имена таблиц

Имена таблиц (как и имена полей, содержащихся в таблице) в вашей базе данных должны "говорить" о том, какие данные в них хранятся. Очень подробное имя (например, `shop_customer_information`) может быть очень информативным, но с другой стороны, слишком сложным для ввода в командной строке. Для большинства людей имя `shop_cust` будет вполне понятным и не вызовет проблем при наборе в командной строке.

Имя таблицы MySQL может иметь длину до 64 символов и не может включать символы прямого слэша (/), обратного слэша (\) и точки (.).

В MySQL есть зарезервированные слова, которые не рекомендуется использовать в качестве имен таблиц и полей *(см. приложение 1)*.

Имена полей и обращение к полю

Имя поля может включать любые символы (до 64). Не рекомендуется использовать имена, подобные `1e`, т. к. выражение вида `1e+1` является неоднозначным — его можно интерпретировать как выражение и как число. Если имя поля относится к служебным словам или содержит специальные символы, то его следует заключить в апострофы при использовании в выражениях:

```
SELECT * FROM 'select' WHERE 'select'.id_sel>20;
```

В табл. 3.9 приведены разрешенные в MySQL формы ссылки на поле таблицы.

Префиксы *`db_name`* и *`table_name`* следует указывать, если ссылка на поле может быть неоднозначной. Например, при выборке из таблиц `table1` и `table2`, содержащих поле `name`, следует уточнить, какая из таблиц имеется в виду, т. е. указать `table1.name` и `table2.name`.

При описании любого из полей таблицы необходимо указать его имя, тип, размер, а также дополнительные атрибуты (их мы рассмотрим позже).

Таблица 3.9. *Допустимые формы ссылки на поле таблицы*

Ссылка	Описание
column_name	Поле *column_name* из любой используемой в запросе таблицы
table_name.column_name	Поле *column_name* из таблицы *table_name* текущей БД
db_name.table_name.column_name	Поле *column_name* из таблицы *table_name* базы данных *db_name* (эта форма доступна начиная с MySQL версии 3.22)
'*column_name*'	Имя поля является зарезервированным словом или содержит специальные символы

Чувствительность имен к регистру букв

В MySQL имена баз данных и таблиц соответствуют каталогам (папкам) и файлам в каталогах. Из этого следует, что чувствительность к регистру букв операционной системы, под которой работает сервер MySQL, определяет чувствительность к регистру букв для имен баз данных и таблиц. Имена баз данных и таблиц нечувствительны к регистру букв в ОС Windows, а в большинстве версий UNIX проявляют чувствительность к регистру букв.

Но, хотя в Windows имена нечувствительны к регистру, не нужно ссылаться на конкретную базу данных или таблицу, используя различные регистры внутри одного и того же запроса. Следующий запрос не будет выполнен, т. к. в нем одна и та же таблица указана и как `reader`, и как `READER`:

```
SELECT * FROM reader WHERE READER.id_reader=1;
```

Если вы не хотите держать в памяти регистр букв имен баз данных и таблиц при их создании, то рекомендую при именовании использовать только строчные буквы.

Типы таблиц

В MySQL есть несколько типов таблиц (табл. 3.10). При создании новой таблицы можно указать MySQL, какой тип таблицы для нее использовать. Для таблицы и определений полей MySQL всегда создает файл с расширением FRM. Индекс и данные хранятся в других файлах (расширения их имен зависят от типа таблицы).

Типом, принятым по умолчанию для таблиц в MySQL, является `MyISAM`. Преобразовывать таблицы из одного типа в другой можно при помощи оператора `ALTER TABLE`. Например:

```
ALTER TABLE table_name TYPE = MyISAM;
```

Таблица 3.10. *Поддерживаемые MySQL типы таблиц*

Тип таблицы	Описание
ISAM	Оригинальное хранилище (устаревший тип таблиц)
MyISAM	Бинарное переносимое хранилище (замена для ISAM)
HEAP	Хранятся только в памяти
BDB или BerkeleyDB	Поддержка транзакций
InnoDB	Поддержка транзакций с блокировкой строк
MERGE	Совокупность нескольких таблиц MyISAM, используемых как одна

Следует отметить, что MySQL поддерживает два различных типа таблиц — транзакционные (InnoDB и BDB) и без поддержки транзакций (HEAP, ISAM, MERGE и MyISAM).

Преимущества *транзакционных таблиц* (Transaction-safe tables, TST):

- надежность — даже если произойдет сбой в работе MySQL или возникнут проблемы с оборудованием, вы сможете восстановить свои данные либо методом автоматического восстановления, либо при помощи резервной копии и журнала транзакций;
- можно сочетать несколько операторов и принимать все эти операторы с помощью одной команды COMMIT;
- можно запустить команду ROLLBACK, чтобы отменить внесенные изменения;
- если произойдет сбой во время обновления, все изменения будут восстановлены (в таблицах без поддержки транзакций все внесенные изменения не могут быть отменены).

Преимущества таблиц без поддержки транзакций (Non-transaction-safe tables, NTST):

- работа с ними происходит намного быстрее, т. к. не выполняются дополнительные транзакции;
- для них требуется меньше дискового пространства, т. к. не применяются дополнительные транзакции;
- для обновлений используется меньше памяти.

Давайте чуть подробнее рассмотрим каждый из типов таблиц.

ISAM

Этот тип является устаревшим, но его еще можно использовать в MySQL версии до 5.0. Индекс хранится в файле с расширением ISM, а данные — в файле с расширением ISD.

Свойства таблиц ISAM:

- ключи сжатой и фиксированной длины;
- фиксированная и динамическая длина записи;
- максимальная длина ключа 256 (по умолчанию);
- данные хранятся в машинном формате, благодаря этому обеспечивается скорость, но возникает зависимость от ОС.

Основные отличия таблиц ISAM от MyISAM:

- таблицы ISAM не являются переносимыми в двоичном виде с одной ОС на другую;
- невозможна работа с таблицами размером больше 4 Гбайт;
- динамические таблицы больше фрагментируются;
- таблицы сжимаются при помощи утилиты PACK_ISAM, а не MYISAMPACK.

MyISAM

Тип таблиц MyISAM принят по умолчанию в MySQL версии 3.23. Он основан на коде ISAM и обладает в сравнении с ним большим количеством полезных дополнений.

Индекс хранится в файле с расширением MYI (MYIndex), а данные — в файле с расширением MYD (MYData). Таблицы типа MyISAM можно проверять/восстанавливать при помощи утилиты MYISAMCHK. Таблицы типа MyISAM можно сжимать при помощи команды MYISAMPACK, после чего они будут занимать намного меньше места.

HEAP

Для HEAP-таблиц используются хэш-индексы; эти таблицы хранятся в памяти. Благодаря этому их обработка осуществляется очень быстро, однако в случае сбоя MySQL будут утрачены все данные, которые в них хранились. Тип HEAP хорошо подходит для временных таблиц, например, для подсчета и вывода статистики на web-страницу.

BDB или BerkeleyDB

Поддержка таблиц типа BDB включена в дистрибутив исходного кода MySQL начиная с версии 3.23.

Тип BerkeleyDB обеспечивает транзакционный обработчик таблиц для MySQL. Использование типа BerkeleyDB повышает для ваших таблиц шанс

уцелеть после сбоев, а также предоставляет возможность осуществлять операции COMMIT (фиксация) и ROLLBACK (откат) для транзакций. Информацию о данном типе таблиц можно найти на сайте **http://www.sleepycat.com**.

InnoDB

Таблицы InnoDB разработаны компанией Innobase Oy (**http://www.innodb.com**). Эти таблицы в MySQL снабжены обработчиком, обеспечивающим безопасные транзакции с возможностями фиксации транзакции, отката и восстановления после сбоя. Для таблиц InnoDB осуществляется блокировка на уровне строки. Перечисленные функции позволяют улучшить взаимную совместимость и повысить производительность в многопользовательском режиме.

Тип InnoDB предназначается для получения максимальной производительности при обработке больших объемов данных. По эффективности использования процессора этот тип намного превосходит другие модели реляционных баз данных с памятью на дисках.

Таблицы InnoDB могут иметь любой размер даже в тех операционных системах, где размер файла ограничен двумя гигабайтами.

MERGE

Таблицы типа MERGE (объединение) появились в версии MySQL 3.23.25.

Таблица типа MERGE (или MRG_MyISAM) представляет собой коллекцию идентичных таблиц MyISAM, которые могут использоваться как одна таблица.

Под идентичными таблицами подразумеваются таблицы, созданные с одинаковой структурой и ключами. Нельзя объединять таблицы, в которых столбцы сжаты разными методами или не совпадают, либо ключи расположены в другом порядке.

При создании таблицы типа MERGE будут образованы файлы определений таблиц (расширение FRM) и списка таблиц (расширение MRG). MRG-файл содержит список индексных файлов (файлы с расширением MYI), с которыми нужно работать как с единым файлом.

Вот некоторые возможности, обеспечиваемые таблицами типа MERGE:

- увеличение скорости работы;
- более эффективный поиск — если точно известно, что вы ищете, можно производить поиск по определенным запросам только в одной из составляющих MERGE-таблицу таблиц, одновременно используя таблицу MERGE для других запросов;

- более простое восстановление;
- быстрая обработка большого количества файлов как одного;
- обход ограничения на размер файлов в операционных системах.

Но у них есть и недостатки:

- для создания таблицы типа `MERGE` можно использовать только идентичные таблицы типа `MyISAM`;
- поля `AUTO_INCREMENT` (автоматически вычисляемые) не обновляются автоматически при применении команды `INSERT`;
- не работает команда `REPLACE`;
- ключи считываются медленнее — при чтении ключа обработчику таблицы типа `MERGE` необходимо прочитать все базовые таблицы, чтобы выяснить, какая из них больше всего соответствует указанному ключу.

ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ

4. Какие типы таблиц MySQL вы можете использовать?
5. Какой тип определен по умолчанию?
6. Какой тип можно использовать для хранения временных данных?

Итак, настало время вернуться к нашей учебной БД "Библиотека" (`library`). Мы будем использовать таблицы типа `MyISAM` (этот тип установлен в MySQL по умолчанию). Определим типы данных полей для наших таблиц (табл. 3.11—1.18).

Таблица 3.11. *Поля таблицы `book` (сущность "Книга")*

Имя поля	Тип поля	Комментарии
`id_book`	`MEDIUMINT(5) UNSIGNED`	Номер книги (до 99 999)
`title`	`VARCHAR(50)`	Название книги (до 50 символов)
`year_issue`	`YEAR`	Год издания
`id_author`	`SMALLINT(4) UNSIGNED`	Номер автора (предполагается, что число авторов не превысит 9999)
`id_publisher`	`SMALLINT(3) UNSIGNED`	Номер издательства (до 999)
`id_section`	`TINYINT(2) UNSIGNED`	Номер раздела библиотеки (до 99)

Таблица 3.12. *Поля таблицы* `unit` *(сущность "Экземпляр")*

Имя поля	Тип поля	Комментарии
id_unit	MEDIUMINT(6) UNSIGNED	Номер экземпляра (до 999 999)
id_book	MEDIUMINT(5) UNSIGNED	Номер книги (до 99 999)

Таблица 3.13. *Поля таблицы* `author` *(сущность "Автор")*

Имя поля	Тип поля	Комментарии
id_author	SMALLINT(4) UNSIGNED	Номер автора (до 9999)
author	VARCHAR(60)	Фамилия, имя и отчество автора (60 символов для отображения)

Таблица 3.14. *Поля таблицы* `publisher` *(сущность "Издательство")*

Имя поля	Тип поля	Комментарии
id_publisher	SMALLINT(2) UNSIGNED	Номер издательства (до 999)
publisher	VARCHAR(30)	Название издательства (до 30 символов)

Таблица 3.15. *Поля таблицы* `section` *(сущность "Раздел")*

Имя поля	Тип поля	Комментарии
id_section	TINYINT(2) UNSIGNED	Номер раздела (до 99)
section	VARCHAR(30)	Название раздела (до 30 символов)

Таблица 3.16. *Поля таблицы* `reader` *(сущность "Читатель")*

Имя поля	Тип поля	Комментарии
id_reader	MEDIUMINT(5) UNSIGNED	Номер читателя (до 99 999)
reader	VARCHAR(60)	Фамилия, имя и отчество читателя (до 60 символов)
id_addr	MEDIUMINT(5) UNSIGNED	Номер адреса (значение этого поля не может превышать число читателей, каждому читателю — по отдельной квартире!)

Таблица 3.17. *Поля таблицы* `address` *(сущность "Адрес читателя")*

Имя поля	Тип поля	Комментарии
id_addr	MEDIUMINT(5) UNSIGNED	Номер адреса (значение этого поля не может превышать число читателей)
address	VARCHAR(80)	Адрес читателя (до 80 символов)
phone	VARCHAR(19)	Номер телефона (с учетом кода и пробелов)

Таблица 3.18. *Поля таблицы* `abonement` *(сущность "Абонемент")*

Имя поля	Тип поля	Комментарии
id_note	MEDIUMINT(5) UNSIGNED	Номер записи (до 99 999)
id_reader	MEDIUMINT(5) UNSIGNED	Номер читателя
id_unit	MEDIUMINT(6) UNSIGNED	Номер экземпляра
get_date	DATE	Дата выдачи
exp_date	DATE	Дата возврата

Все числовые поля мы представили как беззнаковые, добавив при описании атрибут `UNSIGNED`. Это не позволит в данных полях появляться отрицательному значению. При создании всех этих таблиц на компьютере к некоторым полям будут добавлены еще несколько дополнительных атрибутов, например `AUTO_INCREMENT`, `NOT NULL` и др. Но об этом позже.

То, что у нас получилось в итоге, вы можете увидеть на рис. 3.1.

Все таблицы нашей базы данных спроектированы, теперь нужно создать эти таблицы непосредственно на компьютере. Как это делается, вы узнаете в следующей части книги, повествующей о языке SQL.

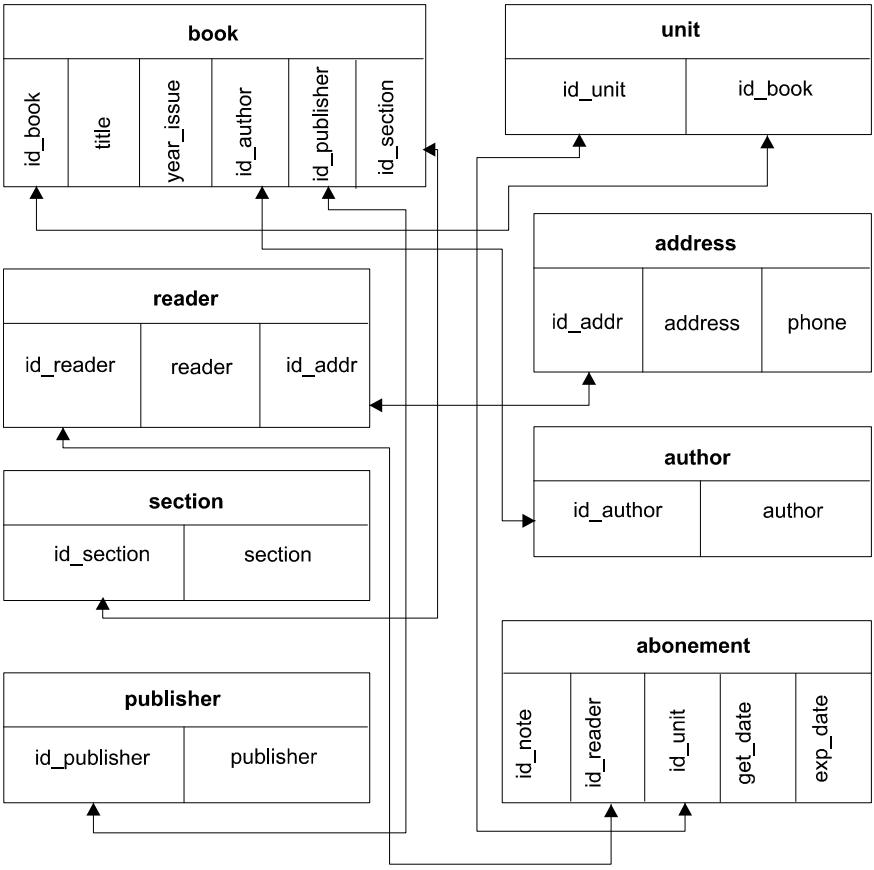
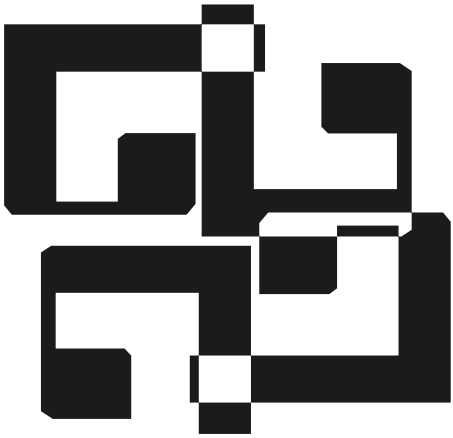


Рис. 3.1. Структура базы данных `library` ("Библиотека")



ЧАСТЬ II

MySQL

Урок 4. Установка MySQL под Windows

Урок 5. Утилиты MySQL

Урок 6. Использование командной строки для обращения к БД

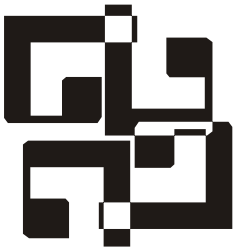
Для установки и использования на вашем компьютере СУБД MySQL под управлением ОС Windows требуется:

- ❒ операционная система Windows 95/98/ME/NT/2000/XP;
- ❒ программа-архиватор для распаковки ZIP-архивов;
- ❒ достаточно свободного места на жестком диске для распаковки, установки и создания базы.

Если у вас нет программы-архиватора, то можно загрузить временную версию архиватора WinZip с сайта **http://www.winzip.com**.

Самой СУБД MySQL для инсталляции требуется около 100 Мбайт дискового пространства.

УРОК 4



Установка MySQL под Windows

Дистрибутив MySQL вы можете загрузить из Интернета (**http://www.mysql.com/downloads/index.html** или **http://www.mysql.ru**). Откройте браузер, перейдите по указанному адресу и выберите рекомендованную версию MySQL — **MySQL x.x -- Production release (recommended)** (на момент написания книги рекомендованной являлась MySQL версии 4.1).

Загрузите дистрибутив на свой жесткий диск в какой-нибудь каталог (папку). После окончания загрузки в выбранном каталоге появится архив MYSQL-X.X.X-WIN32.ZIP. Его нужно распаковать и запустить файл SETUP.EXE. Откроется окно программы установки MySQL Servers and Clients (рис. 4.1).

Нажмите кнопку **Next >** для продолжения установки.

На втором этапе необходимо выбрать тип установки (рис. 4.2). Выберите переключатель **Custom** (выборочная установка) и нажмите кнопку **Next >**.

В следующем окне (рис. 4.3) нужно указать каталог для установки MySQL. Для удобства использования установим MySQL в корневой каталог диска C:. Нажмите кнопку **Change** и задайте C:\MYSQL\ в качестве каталога для установки.

Чтобы перейти к следующему шагу установки, нажмите кнопку **Next >**. В открывшемся окне (рис. 4.4) нажмите кнопку **Install**, чтобы начать процесс установки.

На следующем этапе (рис. 4.5) предлагается создать новую учетную запись на сайте **MySQL.com**. Мы не будем этого делать и выберем переключатель **Skip Sign-Up** (пропустить). Для продолжения установки нажмем кнопку **Next >**.

После окончания установки предлагается настроить MySQL Server (рис. 4.6). Нажмите кнопку **Finish**.

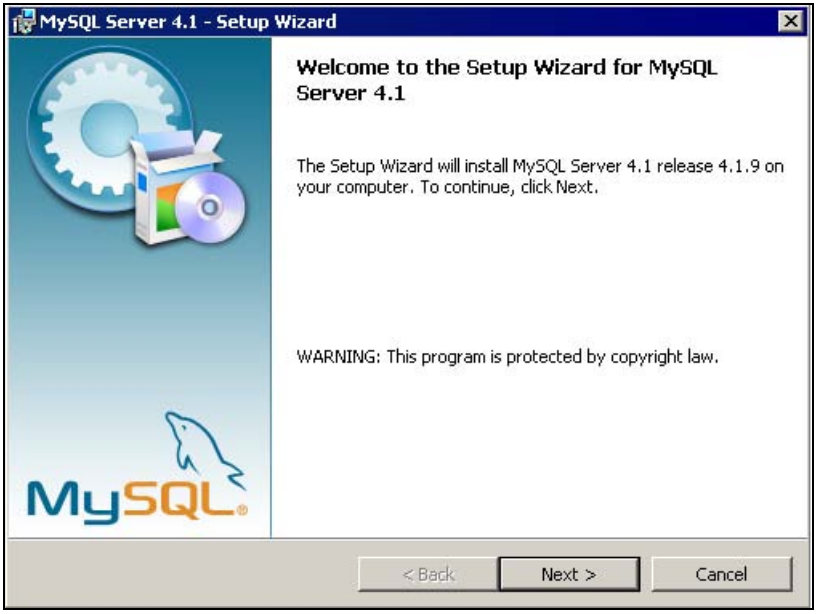


Рис. 4.1. Установка, этап 1

Рис. 4.2. Установка, этап 2

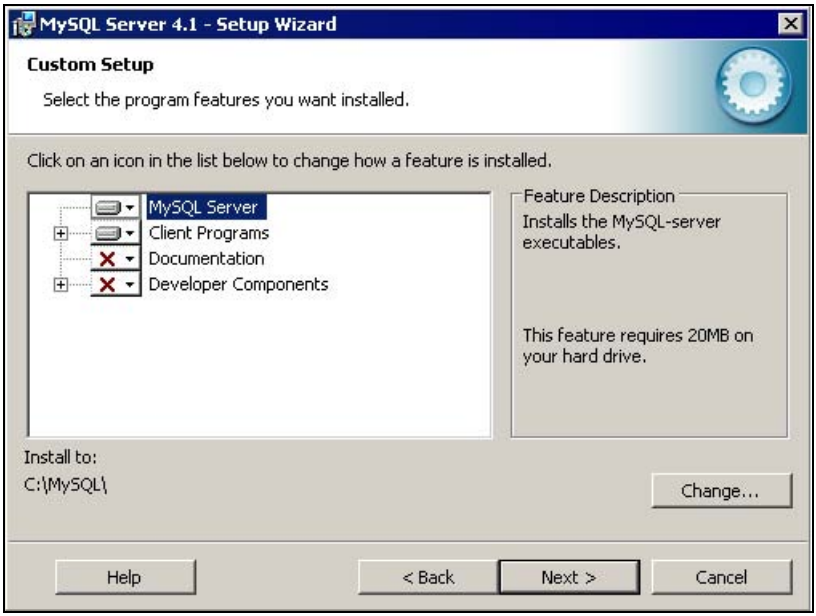


Рис. 4.3. Установка, этап 3

Рис. 4.4. Установка, этап 4

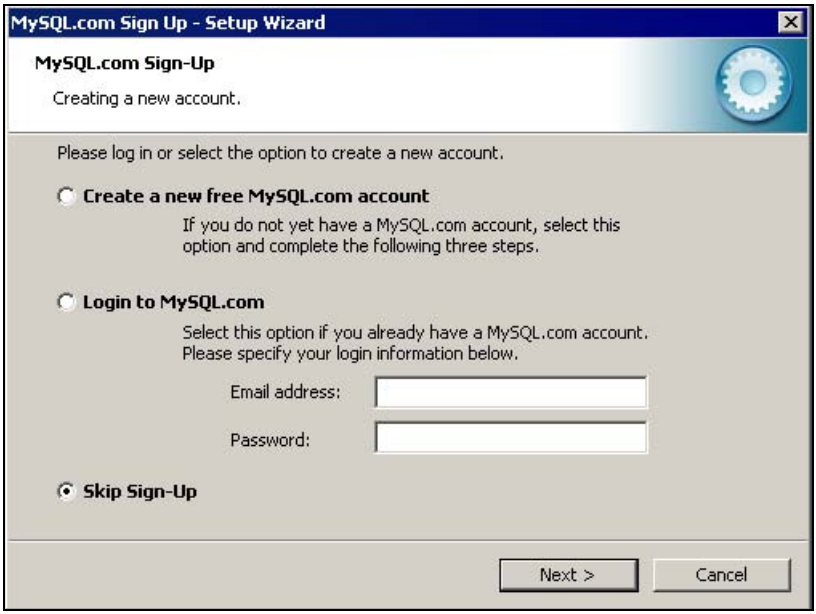


Рис. 4.5. Установка, этап 5

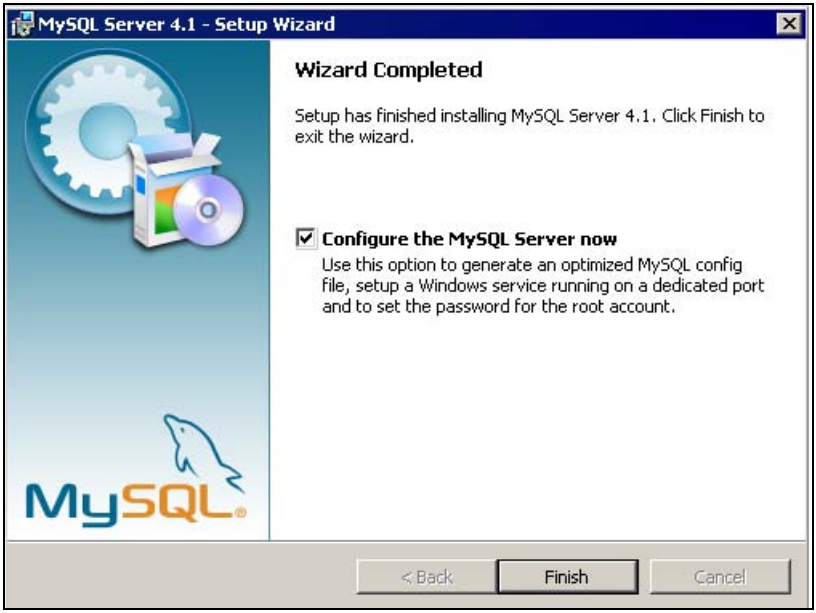


Рис. 4.6. Установка, этап 6 (последний)

Программа сервера базы данных расположена на том компьютере, где хранится БД. Она получает запросы клиентов (программ, передающих запросы) через сеть и осуществляет доступ к содержимому БД, для предоставления запрошенной информации.

Итак, мы решили настроить сервер. Программа установки запустила мастер конфигурации (рис. 4.7). Для продолжения нажмите кнопку **Next >**.

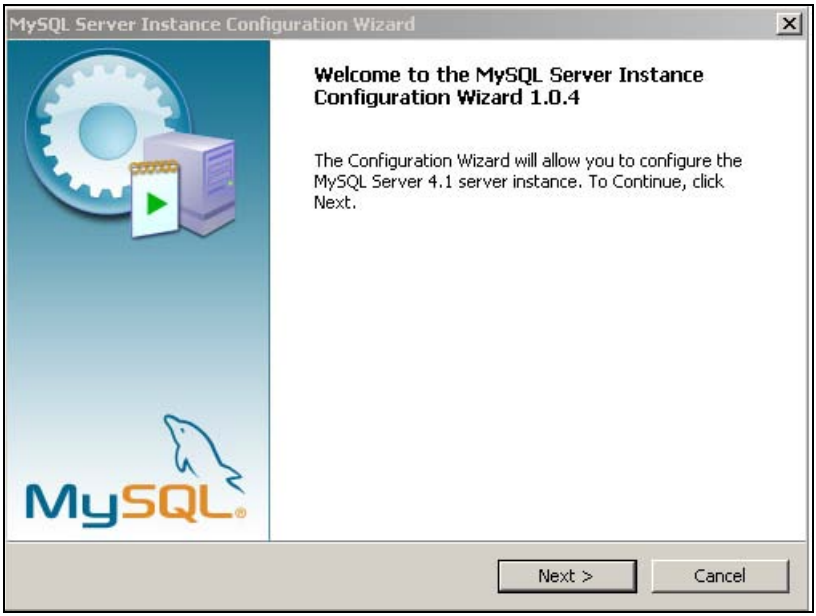


Рис. 4.7. Настройка, этап 1

На втором этапе конфигурации (рис. 4.8) вам предложат выбрать один из двух типов конфигурации. Выберите переключатель **Standard Configuration** и нажмите кнопку **Next >** для продолжения настройки.

На следующем этапе (рис. 4.9) вы можете установить флажок **Install As Windows Service**, если хотите запускать сервер автоматически при загрузке Windows. Нажмите кнопку **Next >** для продолжения.

Мастер конфигурации готов к внесению изменений (рис. 4.10). Нажмите кнопку **Execute**.

Конфигурация закончена (рис. 4.11), все настройки внесены в файл MY.INI.

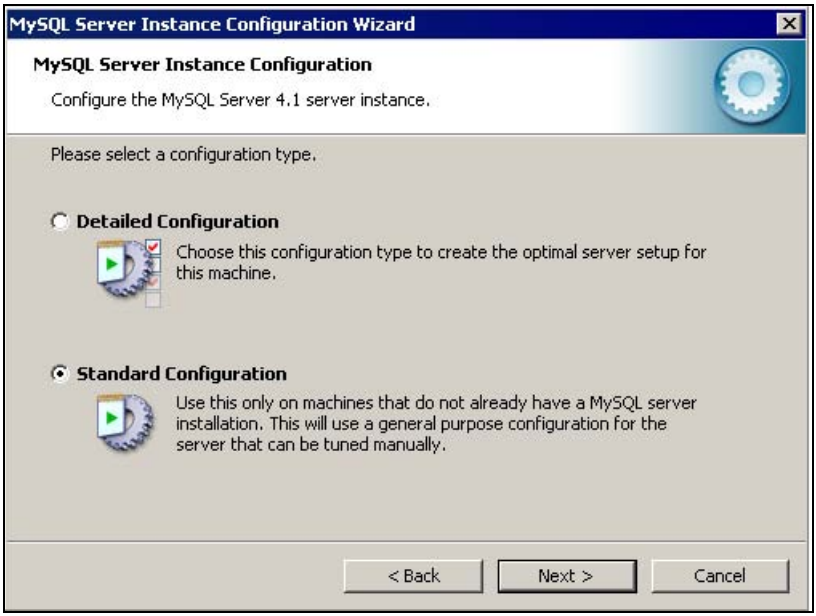


Рис. 4.8. Настройка, этап 2

Рис. 4.9. Настройка, этап 3

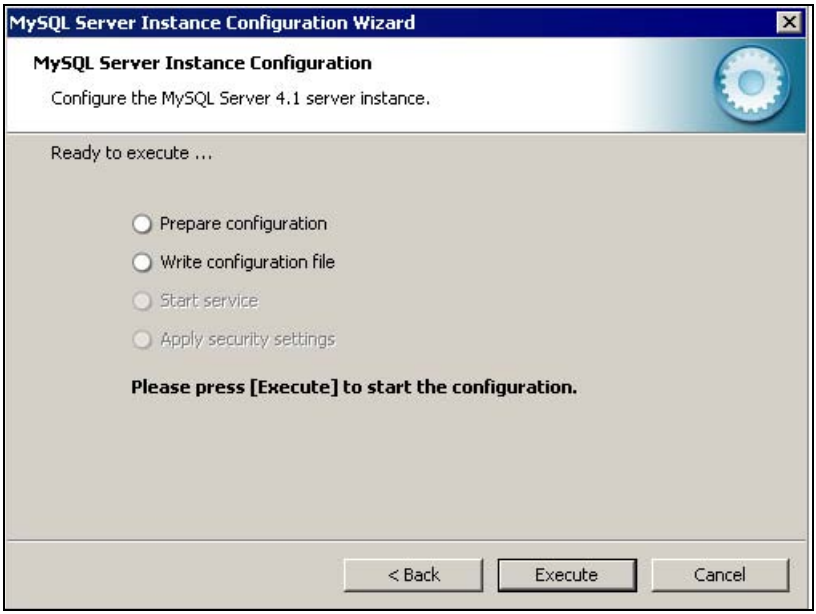


Рис. 4.10. Настройка, этап 4

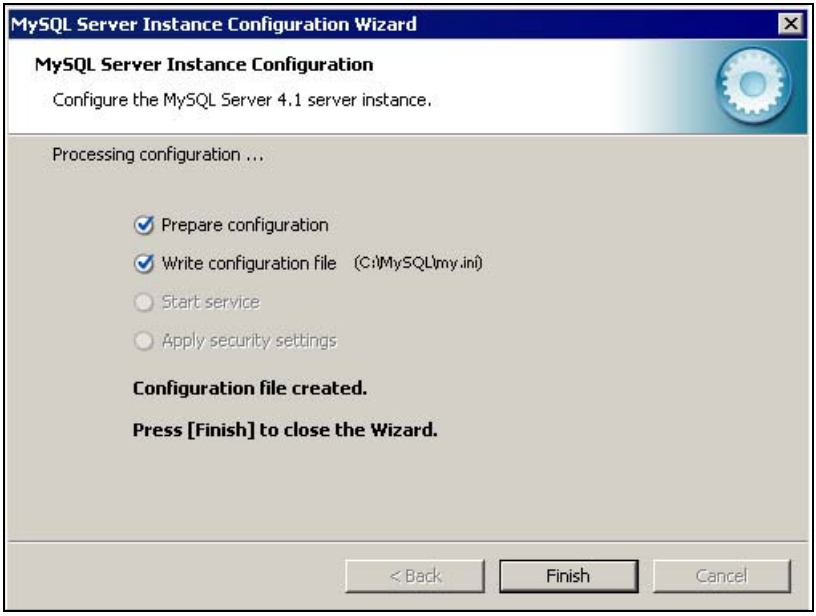


Рис. 4.11. Настройка, этап 5 (последний)

УРОК 5

Утилиты MySQL

Теперь у вас установлена СУБД MySQL. Запустите Проводник Windows и откройте папку C:\MYSQL\BIN\ (рис. 5.1). Здесь вы найдете утилиты, некоторые из них мы рассмотрим далее.

Утилита MYSQLD.EXE предназначена для запуска сервера MySQL. Запуск сервера — первое, что нужно сделать перед началом работы с БД.

Сделайте двойной щелчок на имени файла MYSQLD.EXE в окне **bin** (рис. 5.1) — и сервер запущен. После этого можно использовать остальные утилиты.

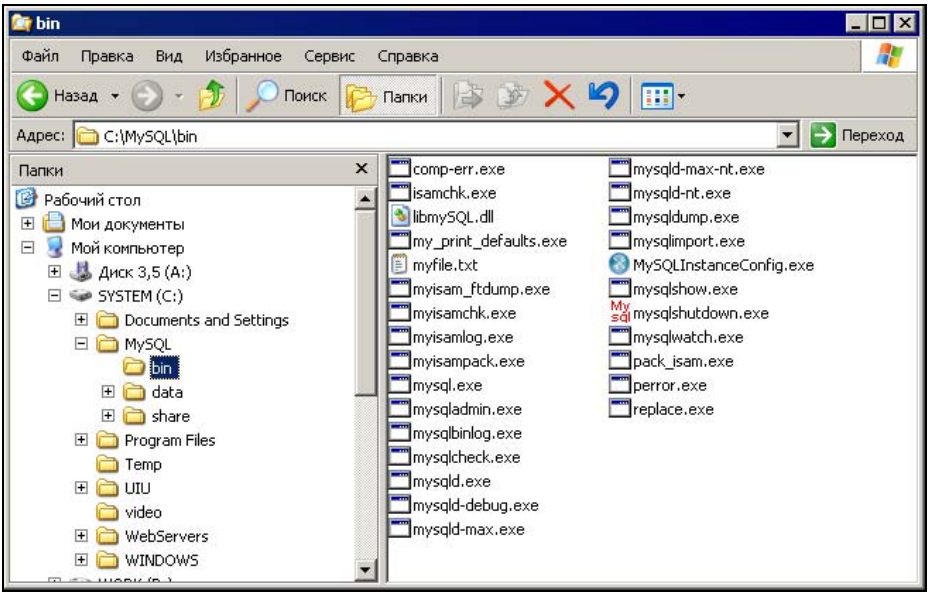


Рис. 5.1. Утилиты в папке C:\MYSQL\BIN\

Утилита MYSQLSHOW.EXE при запуске без параметров выводит список баз данных, хранящихся на сервере. Если при запуске утилиты указать имя БД, то будет выведен список таблиц, имеющихся в этой БД. При указании имени конкретной таблицы выводится информация о ее структуре.

Формат запуска утилиты с параметрами:

```
mysqlshow.exe [опции] [имя_БД [имя_таблицы]]
```

Примечание

[ ] — условное обозначение того, что параметр не является обязательным.

Допустимые опции: `-hхост`, `-uпользователь`, `-pпароль`. Например:

```
mysqlshow.exe -hlocalhost -uroot -p12345
```

УРОК 6

Использование командной строки для обращения к БД

Для того чтобы увидеть, как работают описанные утилиты, перейдем в режим DOS. Нажмите кнопку **Пуск** (**Start**) и выберите пункт **Выполнить** (**Run**). В открывшемся окне введите команду `cmd` (или `command`) и нажмите кнопку **OK**. Введите `cd \mysql\bin` в командной строке открывшегося окна и нажмите клавишу <Enter> (рис. 6.1).

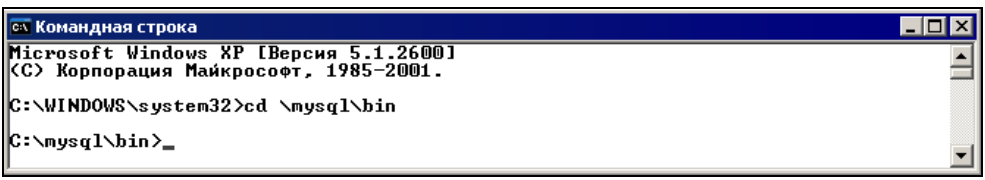


Рис. 6.1. Переход в каталог C:\MYSQL\BIN

Итак, давайте попробуем запустить утилиту MYSQLSHOW без указания каких-либо параметров (рис. 6.2).

В MySQL есть две базы данных, имена которых вы видите в окне. База данных `test` не содержит таблиц, поэтому для просмотра таблиц воспользуемся базой данных `mysql`.

Введите в командной строке **`mysqlshow mysql`**.

После запуска утилиты вы увидите список таблиц, имеющихся в БД (рис. 6.3). Теперь можно просмотреть структуру любой из этих таблиц. Для этого введите в командной строке **`mysqlshow mysql func`** (рис. 6.4).

Утилита MYSQL.EXE предназначена для запуска консоли (командной строки MySQL). Введите в командной строке **`mysql`** и нажмите клавишу <Enter>.

```
Командная строка
Microsoft Windows XP [Версия 5.1.2600]
(C) Корпорация Майкрософт, 1985-2001.

C:\Documents and Settings\ZeroTool>cd \mysql\bin

C:\MySQL\bin>mysqlshow
+------------+
| Databases  |
+------------+
| mysql      |
| test       |
+------------+

C:\MySQL\bin>_
```

Рис. 6.2. Результат запуска утилиты MYSQLSHOW без указания параметров

```
Командная строка
C:\Documents and Settings\ZeroTool>cd \mysql\bin

C:\MySQL\bin>mysqlshow mysql
Database: mysql
+---------------------------+
|          Tables           |
+---------------------------+
| columns_priv              |
| db                        |
| func                      |
| help_category             |
| help_keyword              |
| help_relation             |
| help_topic                |
| host                      |
| tables_priv               |
| time_zone                 |
| time_zone_leap_second     |
| time_zone_name            |
| time_zone_transition      |
| time_zone_transition_type |
| user                      |
+---------------------------+

C:\MySQL\bin>_
```

Рис. 6.3. Результат запуска утилиты MYSQLSHOW с указанием имени БД

Для просмотра команд, допустимых в консоли, введите в командной строке `help`. Будет выведен список команд и их сокращенных вариантов (shortcut), начинающихся с обратного слэша (\) (рис. 6.5).

Каждая команда должна заканчиваться символом точки с запятой (`;`). Некоторые команды выполняются и без этого символа (например, `help`). Но если команда требует обязательного наличия точки с запятой, а вы ее не ввели и нажали клавишу <Enter>, то система не выполнит команду, а будет ждать ввода символа `';'` (рис. 6.6).

```
Командная строка

C:\MySQL\bin>mysqlshow mysql func
Database: mysql  Table: func  Rows: 0
+-------+-----------------------------+-----------+------+-----+----------+----
---+-----------------------------+---------+
| Field | Type                        | Collation | Null | Key | Default  | Ext
ra | Privileges                  | Comment |
+-------+-----------------------------+-----------+------+-----+----------+----
---+-----------------------------+---------+
| name  | char(64)                    | utf8_bin  |      | PRI |          |
   | select,insert,update,references |         |
| ret   | tinyint(1)                  | NULL      |      |     | 0        |
   | select,insert,update,references |         |
| dl    | char(128)                   | utf8_bin  |      |     |          |
   | select,insert,update,references |         |
| type  | enum('function','aggregate') | utf8_bin  |      |     | function |
   | select,insert,update,references |         |
+-------+-----------------------------+-----------+------+-----+----------+----
---+-----------------------------+---------+

C:\MySQL\bin>_
```

Рис. 6.4. Результат запуска утилиты MYSQLSHOW с указанием имени таблицы

```
Командная строка - mysql
mysql> help

For the complete MySQL Manual online, visit:
   http://www.mysql.com/documentation

For info on technical support from MySQL developers, visit:
   http://www.mysql.com/support

For info on MySQL books, utilities, consultants, etc., visit:
   http://www.mysql.com/portal

List of all MySQL commands:
Note that all text commands must be first on line and end with ';'
?         (\?) Synonym for `help'.
clear     (\c) Clear command.
connect   (\r) Reconnect to the server. Optional arguments are db and host.
delimiter (\d) Set query delimiter.
ego       (\G) Send command to mysql server, display result vertically.
exit      (\q) Exit mysql. Same as quit.
go        (\g) Send command to mysql server.
help      (\h) Display this help.
notee     (\t) Don't write into outfile.
print     (\p) Print current command.
prompt    (\R) Change your mysql prompt.
quit      (\q) Quit mysql.
rehash    (\#) Rebuild completion hash.
source    (\.) Execute a SQL script file. Takes a file name as an argument.
status    (\s) Get status information from the server.
tee       (\T) Set outfile [to_outfile]. Append everything into given outfile.
use       (\u) Use another database. Takes database name as argument.

For server side help, type 'help contents'

mysql> _
```

Рис. 6.5. Список всех команд MySQL

После того как вы введете `';'`, система выполнит команду. Команда **`show tables;`** выведет список таблиц из БД `mysql`, которую мы выбрали ранее с помощью команды **`use`** (рис. 6.7).

```
Командная строка - mysql

C:\MySQL\bin>mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 21 to server version: 4.1.9

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

mysql> use mysql;
Database changed
mysql> show tables
    ->
```

Рис. 6.6. Ожидание ввода `';'`

```
Командная строка - mysql
mysql> use mysql;
Database changed
mysql> use mysql;
Database changed
mysql> show tables
    -> ;
+---------------------------+
| Tables_in_mysql           |
+---------------------------+
| columns_priv              |
| db                        |
| func                      |
| help_category             |
| help_keyword              |
| help_relation             |
| help_topic                |
| host                      |
| tables_priv               |
| time_zone                 |
| time_zone_leap_second     |
| time_zone_name            |
| time_zone_transition      |
| time_zone_transition_type |
| user                      |
+---------------------------+
15 rows in set (0.00 sec)

mysql>
```

Рис. 6.7. Выполнение команды

Для выхода из режима консоли MySQL введите в командной строке команду **`exit`** или **`quit`** (рис. 6.8).

```
Командная строка
mysql> exit
Bye

C:\MySQL\bin>_
```

Рис. 6.8. Выход из режима консоли

Итак, продолжим рассмотрение утилит MySQL.

Утилита MYSQLADMIN.EXE предоставляет административные функции сервера. Полный список команд можно просмотреть, запустив утилиту без параметров.

Вот некоторые из них:

- `create` *`имя_БД`* — создать базу данных;
- `drop` *`имя_БД`* — удалить базу данных;
- `reload` — перезапустить сервер;
- `shutdown` — окончание работы сервера;
- `status` — информация о состоянии сервера.

Формат запуска утилиты с параметрами (опции те же, что и у MYSQLSHOW.EXE):

`mysqladmin.exe [`*`опции`*`] [`*`команды`*`]`

Утилита MYSQLDUMP.EXE предназначена для сохранения базы данных или ее таблицы (рис. 6.9). Формат запуска утилиты:

`mysqldump.exe [`*`опции`*`] [`*`имя_БД [имя_таблицы]`*`]`

```
Командная строка
--
-- Table structure for table `user`
--

DROP TABLE IF EXISTS `user`;
CREATE TABLE `user` (
  `Host` varchar(60) collate utf8_bin NOT NULL default '',
  `User` varchar(16) collate utf8_bin NOT NULL default '',
  `Password` varchar(41) collate utf8_bin NOT NULL default '',
  `Select_priv` enum('N','Y') collate utf8_bin NOT NULL default 'N',
  `Insert_priv` enum('N','Y') collate utf8_bin NOT NULL default 'N',
  `Update_priv` enum('N','Y') collate utf8_bin NOT NULL default 'N',
  `Delete_priv` enum('N','Y') collate utf8_bin NOT NULL default 'N',
  `Create_priv` enum('N','Y') collate utf8_bin NOT NULL default 'N',
  `Drop_priv` enum('N','Y') collate utf8_bin NOT NULL default 'N',
  `Reload_priv` enum('N','Y') collate utf8_bin NOT NULL default 'N',
  `Shutdown_priv` enum('N','Y') collate utf8_bin NOT NULL default 'N',
  `Process_priv` enum('N','Y') collate utf8_bin NOT NULL default 'N',
  `File_priv` enum('N','Y') collate utf8_bin NOT NULL default 'N',
  `Grant_priv` enum('N','Y') collate utf8_bin NOT NULL default 'N',
  `References_priv` enum('N','Y') collate utf8_bin NOT NULL default 'N',
  `Index_priv` enum('N','Y') collate utf8_bin NOT NULL default 'N',
  `Alter_priv` enum('N','Y') collate utf8_bin NOT NULL default 'N',
  `Show_db_priv` enum('N','Y') collate utf8_bin NOT NULL default 'N',
  `Super_priv` enum('N','Y') collate utf8_bin NOT NULL default 'N',
  `Create_tmp_table_priv` enum('N','Y') collate utf8_bin NOT NULL default 'N',
  `Lock_tables_priv` enum('N','Y') collate utf8_bin NOT NULL default 'N',
  `Execute_priv` enum('N','Y') collate utf8_bin NOT NULL default 'N',
  `Repl_slave_priv` enum('N','Y') collate utf8_bin NOT NULL default 'N',
  `Repl_client_priv` enum('N','Y') collate utf8_bin NOT NULL default 'N',
  `ssl_type` enum('','ANY','X509','SPECIFIED') collate utf8_bin NOT NULL default
 '',
  `ssl_cipher` blob NOT NULL,
  `x509_issuer` blob NOT NULL,
  `x509_subject` blob NOT NULL,
  `max_questions` int(11) unsigned NOT NULL default '0',
  `max_updates` int(11) unsigned NOT NULL default '0',
  `max_connections` int(11) unsigned NOT NULL default '0',
  PRIMARY KEY  (`Host`,`User`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_bin COMMENT='Users and global
privileges';

--
-- Dumping data for table `user`
--
```

Рис. 6.9. Результат работы утилиты MYSQLDUMP (структура одной из таблиц)

Здесь, кроме уже известных вам опций, можно упомянуть еще одну:

`--add-drop-table` — добавляет в текстовый файл, содержащий структуру и данные БД, инструкцию для автоматического удаления существующей таблицы с этим же именем (`DROP TABLE IF EXISTS '`*`имя_таблицы`*`'`).

Примечание

Эта опция используется в ранних версиях MySQL, а в версии 4.1, установка которой рассмотрена в этой книге, данная инструкция добавляется автоматически.

Вы также можете перенаправить вывод в файл: введите в командной строке **`mysqldump mysql>dump.txt`** и нажмите клавишу <Enter>. В каталоге C:\MYSQL\BIN будет создан файл DUMP.TXT, содержащий структуру и данные БД `mysql`. Таким способом можно выполнять резервное копирование вашей базы.

Кроме этого данный файл можно перенести на другой компьютер для создания на нем такой же БД (рис. 6.10). В этом случае после запуска сервера нужно выполнить следующие команды:

`mysqladmin create db_name` *(Создаем БД)*

`mysql db_name<file_name` *(Заносим данные из файла)*

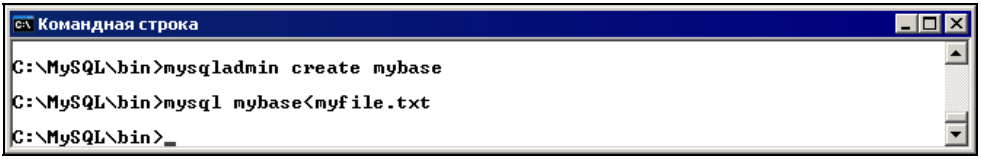


Рис. 6.10. Создание БД

Напоминаю, что удалить БД можно с помощью команды **`drop`**:

`mysqladmin drop db_name`

Также вы можете создать БД, запустив команду **`create database db_name`** в режиме консоли (рис. 6.11).

Запустите режим консоли, введите команду **`create database`** и укажите имя БД (например, `mybase`). Поставьте точку с запятой (`;`) и нажмите клавишу <Enter>. Система создаст БД с указанным именем. После этого вы сможете просмотреть список баз данных, выполнив команду **`show databases;`**. Для удаления БД введите команду **`drop database db_name;`** (рис. 6.12).

Итак, теперь вы знаете, как создается база данных. Предлагаю вам создать БД `library`, с которой мы и будем работать.

```
Ярлык для cmd.exe - mysql
mysql> create database mybase;
Query OK, 1 row affected (0.01 sec)

mysql> show databases;
+----------+
| Database |
+----------+
| mybase   |
| mysql    |
| test     |
+----------+
3 rows in set (0.00 sec)

mysql> _
```

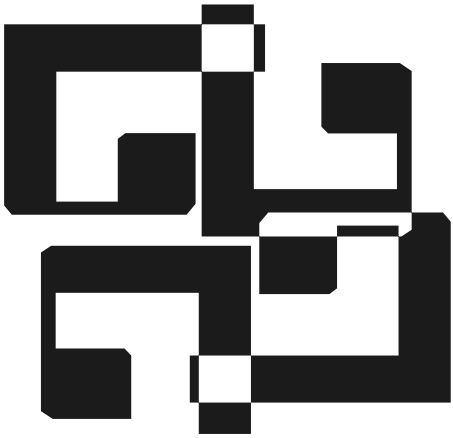
Рис. 6.11. Создание БД в режиме консоли MySQL

```
Командная строка - mysql
mysql> drop database mybase;
Query OK, 0 rows affected (0.00 sec)

mysql> show databases;
+----------+
| Database |
+----------+
| mysql    |
| test     |
+----------+
2 rows in set (0.00 sec)

mysql> _
```

Рис. 6.12. Удаление БД



ЧАСТЬ III

Формирование запросов к БД. Язык SQL

Для чтения и записи в базах данных MySQL используется структурированный язык запросов (Structured Query Language, SQL). С помощью SQL можно осуществлять ввод данных, их изменение, удаление, а также выборку. SQL является основополагающим инструментом, необходимым для взаимодействия с MySQL и другими СУБД. Даже если для доступа к БД вы пользуетесь каким-то приложением или графическим интерфейсом пользователя (GUI), где-то "в глубине" это приложение генерирует SQL-команды.

Язык SQL "структурирован" в том отношении, что он придерживается определенного набора правил. Он очень не похож на традиционные языки программирования, такие как C++, Java, PHP или Perl, близок к "естественному языку". Пример SQL-запроса:

```
SELECT title FROM book WHERE year_issue=2003
```

Как видите, это напоминает фразу на ломаном английском: "Выбрать название из книг, где год выпуска равен 2003". У такого подхода есть как преимущества, так и недостатки.

Операторы языка SQL можно разбить на несколько групп:

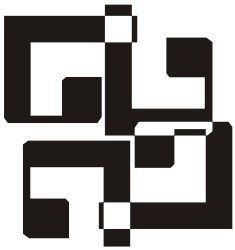
Создание, удаление, изменение и выборка БД	**Создание, удаление и изменение таблиц и индексов**	**Изменение данных в таблицах**
CREATE DATABASE, DROP DATABASE, ALTER DATABASE, USE	CREATE TABLE, CREATE INDEX, DROP TABLE, DROP INDEX, ALTER TABLE	INSERT, DELETE, UPDATE, LOAD DATA, REPLACE

Выполнение транзакций	**Просмотр информации о таблицах и БД**	**Администрирование**	**Выборка данных из таблиц**
BEGIN, COMMIT, ROLLBACK, SET AUTOCOMMIT	DESCRIBE (DESC), SHOW	GRANT, FLUSH, REVOKE	SELECT, UNION

Здесь приведены только операторы, поддерживаемые MySQL. Но следует заметить, что СУБД MySQL постоянно совершенствуется и с каждой новой версией в нее добавляются все новые и новые возможности.

Что касается чувствительности SQL-команд к регистру, то все зависит от части этой команды и от ОС. Например, ключевые слова и имена функций не зависят от регистра и могут вводиться как угодно, а имена таблиц и баз данных зависят от ОС, под управлением которой работает сервер (об этом мы уже говорили в *части II*). Имена полей и индексов не чувствительны к регистру, а псевдонимы — чувствительны (псевдоним можно объявить в любом регистре, но в дальнейшем нужно обращаться к нему так, как он был объявлен). В этой книге я записывал ключевые слова и названия функций в верхнем регистре, а все остальное (имена таблиц, полей и т. д.) — в нижнем.

УРОК 7



Создание и удаление таблиц

После создания общей структуры базы данных нам необходимо приступить к созданию таблиц. В этом уроке мы рассмотрим операторы языка SQL, позволяющие создавать и удалять таблицы.

Создание таблиц

Создать таблицу можно с помощью оператора `CREATE TABLE`. Общий формат записи оператора `CREATE TABLE`:

```
CREATE TABLE имя_таблицы (описание_поля, ...)
```

Параметры: *имя_таблицы* — имя создаваемой таблицы, *описание_поля* — описание поля таблицы. В MySQL 3.23 и более поздних версий в операторе `CREATE TABLE` можно использовать ключевые слова `IF NOT EXISTS`:

```
CREATE TABLE IF NOT EXISTS имя_таблицы (описание_поля, ...)
```

В этом случае не возникнет ошибка, если такая таблица уже существует. Но нужно учитывать, что идентичность структуры таблиц не проверяется. Во время создания таблицы можно установить ее тип *(см. часть II)*, например:

```
CREATE TABLE mytable (...) TYPE=INNODB;
```

Если тип таблицы не задан, то сервер создает, как вы помните, таблицу типа `MyISAM`.

При описании поля указываются его имя, тип, а также дополнительные инструкции (если они нужны):

- `PRIMARY KEY` — данная инструкция говорит о том, что поле является первичным ключом. Поле с такой инструкцией должно быть определено как `NOT NULL`. В таблице может быть только один первичный ключ. Если ключ

составной, то для каждого поля, входящего в составной ключ, не нужно указывать PRIMARY KEY (необходимо указать только NOT NULL), вместо этого применяется синтаксис PRIMARY KEY(*имя_поля*1,*имя_поля*2);

- NULL — поле с неопределенным значением (если не указывается ни NULL, ни NOT NULL, то поле интерпретируется так, как будто указано NULL);
- NOT NULL — непустое поле (при внесении записи в таблицу должно быть обязательно заполнено). Эта инструкция указывается для первичного ключа;

Примечание

Инструкции NULL и NOT NULL не применяются к полям типа TIMESTAMP. При установке такого поля в значение NULL в него будут записаны текущие дата и время.

- KEY или INDEX — поле будет индексированным (в таких полях быстрее выполняются операции SELECT);
- ZEROFILL — эта инструкция добавляет к значению нули слева, чтобы его длина совпала с длиной поля. Например, если поле имеет тип smallint(5) и вы заносите в него значение 1, то в результате получаете 00001;
- UNSIGNED — беззнаковое поле (для числовых типов);
- AUTO_INCREMENT — указывается для целочисленных полей. При записи величины NULL (рекомендуется) или 0 в поле AUTO_INCREMENT получим значение value+1, где value представляет собой наибольшее для этого поля значение в таблице на момент записи. Последовательность AUTO_INCREMENT начинается с 1. В таблице может быть только одно поле AUTO_INCREMENT, и оно должно быть индексировано. Кроме того, версия MySQL 3.23 будет правильно работать только с положительными величинами поля AUTO_INCREMENT. В случае внесения отрицательного числа оно интерпретируется как очень большое положительное число;
- DEFAULT '*значение*' — значение по умолчанию. Величина DEFAULT должна быть константой, она не может быть функцией или выражением. Если для данного поля не задается никакой величины DEFAULT, то MySQL автоматически назначает ее. Если поле может принимать NULL как допустимую величину, то по умолчанию ему присваивается значение NULL. Если поле объявлено как NOT NULL, то значение по умолчанию зависит от типа поля:
  - для числовых типов, за исключением объявленных с атрибутом AUTO_INCREMENT, значение по умолчанию равно 0. Для поля AUTO_INCREMENT значением по умолчанию является следующее значение в последовательности для этого поля;

- для типов даты и времени, отличных от TIMESTAMP, значение по умолчанию равно соответствующей нулевой величине для данного типа. Для первого поля TIMESTAMP в таблице значение по умолчанию представляет собой текущее значение даты и времени;
- для строковых типов, кроме ENUM, значением по умолчанию будет пустая строка. Для ENUM значение по умолчанию равно первой перечисляемой величине (если явно не задано другое значение по умолчанию с помощью директивы DEFAULT);

- BINARY — поле, чувствительное к регистру букв (для типов CHAR и VARCHAR);
- UNIQUE — поле с уникальными значениями. Если попытаться внести строку, совпадающую с уже существующей строкой, то возникнет ошибка выполнения команды.

Подробнее об индексировании

Есть множество методов оптимизации работы вашей системы, но самым эффективным остается индексирование. Индексирование таблиц позволит уменьшить время выполнения запросов. Рассмотрим неиндексированную таблицу (табл. 7.1).

Таблица 7.1. *Неиндексированная таблица*

reader		
id_reader	reader	id_addr
12	Иванов	2
13	Петров	3
14	Васечкин	4
15	Смирнова	5
16	Дворников	6
17	Иванова	2
18	Мусин	7
19	Смирнов	5

Данная таблица не имеет индекса, поэтому при поиске читателей, живущих по определенному адресу, нужно будет проверить все строки на совпадение с нужным значением. Для этого потребуется просмотреть всю таблицу (она может быть в сотни раз больше), что займет значительное время. В табл. 7.2 представлена таблица индексов для поля id_addr таблицы reader.

Таблица 7.2. *Таблица индексов для поля* `id_addr`

id_addr
2
2
3
4
5
5
6
7

Соответственно, если потребуется найти читателей с номером адреса 2, то просмотр таблицы вернет нам два значения. Поиск можно прекратить по достижению следующего номера (3), т. к. после него нужных нам значений нет. Если нужно найти какое-то значение, находящееся где-то в середине, то для этого есть специальные алгоритмы позиционирования, позволяющие быстро найти нужное значение без просмотра всех записей.

Механизм индексирования еще более полезен при запросе данных из нескольких таблиц. Допустим, при запросе к одной неиндексированной таблице, состоящей из 1000 записей, нам нужно просмотреть и проанализировать только 1000 строк. В случае, если таких таблиц две, получится $1000 \times 1000 = 1\ 000\ 000$ записей. А если их пять?!

Индексация таблиц позволяет СУБД MySQL ускорить поиск записей, удовлетворяющих условию, описанному в предложении `WHERE`. Кроме этого индексирование позволяет ускорить поиск минимального и максимального значений в поле, сделать более быстрыми операции сортировки и группировки значений.

Недостатки

У индексирования есть и недостатки, но они столь незначительны, что с лихвой перекрываются преимуществами. Индексный файл занимает определенное место на жестком диске сервера и поэтому при большом количестве индексов он может быстро достичь максимального размера. Кроме этого индексы ускоряют поиск данных, но замедляют операции добавления, удаления и обновления в индексированных полях.

Создание индекса

В MySQL можно проиндексировать столбцы всех типов. Максимальное количество ключей и максимальная длина индексов зависят от типа таблицы (для типа `MyISAM` — по умолчанию 32 ключа в одной таблице и 500 байт на ключ). Первичный ключ индексируется автоматически.

Таблицу можно индексировать как по одному, так и по нескольким полям. Индекс может содержать как уникальные, так и повторяющиеся значения.

Для полей типа `CHAR` и `VARCHAR` можно индексировать префикс поля (несколько первых символов), это намного быстрее и требует меньше места на диске, нежели индексация всего поля. Для полей типа `TEXT` и `BLOB` также необходимо индексировать префикс поля (до 255 байт). Нельзя индексировать поле целиком.

Синтаксис инструкции для индексации префикса поля при создании таблицы может выглядеть примерно так:

```
KEY имя_индекса (имя_поля(длина))
```

где *имя_поля(длина)* — имя поля с указанием количества индексируемых символов.

Пример запроса с применением такого индексирования показан на рис. 7.1.

```
Командная строка - mysql test
mysql> CREATE TABLE test(title VARCHAR(100), KEY ind_title(title(10)));
Query OK, 0 rows affected (0.00 sec)

mysql>
```

Рис. 7.1. Создание индекса для первых 10 символов поля `title`

Просмотреть индексы вы можете с помощью следующей команды:

```
SHOW INDEX FROM имя_таблицы
```

Можно создавать индексы нескольких типов:

- *обычный индекс* — позволяет содержать дублирующиеся значения;
- *уникальный индекс* — запрещает наличие дублирующихся значений;
- *индекс* `FULLTEXT` — предназначен для осуществления контекстного поиска.

Кроме оператора `CREATE TABLE`, для создания индексов можно использовать операторы `CREATE INDEX` и `ALTER TABLE` (это позволит создавать индексы в уже существующих таблицах).

Синтаксис инструкции создания обычного индекса:

```
ALTER TABLE имя_таблицы ADD INDEX имя_индекса (имя_поля);
```

Синтаксис инструкции создания уникального индекса:

```
ALTER TABLE имя_таблицы ADD UNIQUE имя_индекса (имя_поля);
```

Синтаксис инструкции создания первичного ключа:

```
ALTER TABLE имя_таблицы ADD PRIMARY KEY (имя_поля);
```

Синтаксис инструкции создания индекса типа `FULLTEXT`:

```
ALTER TABLE имя_таблицы ADD FULLTEXT (имя_поля);
```

Здесь *имя_поля* — имя индексируемого поля (полей).

С помощью оператора `CREATE INDEX` вы можете добавить все перечисленные индексы (кроме `PRIMARY KEY`):

```
CREATE INDEX имя_индекса ON имя_таблицы (имя_поля);
CREATE FULLTEXT INDEX имя_индекса ON имя_таблицы (имя_поля);
CREATE UNIQUE INDEX имя_индекса ON имя_таблицы (имя_поля);
```

Что же касается индексации по префиксу поля, то, как показано на рис. 7.1, вы должны указать количество символов (*n*).

Приведу еще один пример:

```
CREATE TABLE  mytable (first_name VARCHAR(30), last_name VARCHAR(30),
  INDEX (first_name(), last_name(5)));
```

Примечание

Учтите, что длина префикса это количество *байт*, а не *символов*. То есть для однобайтовых кодировок это одно и то же, а для многобайтовых — нет.

Индексирование по префиксу поля накладывает определенные ограничения на изменение длины поля. Нельзя делать длину такого поля меньше, чем длина объявленного индекса. Вам придется удалить индекс, изменить длину поля, а затем воссоздать индекс снова с более коротким префиксом. Поля с индексами типа `FULLTEXT` не могут иметь префикса.

Удаление индекса

Удалить индекс можно с помощью оператора `DROP INDEX` или `ALTER TABLE`:

```
DROP INDEX имя_индекса ON имя_таблицы;
ALTER TABLE имя_таблицы DROP INDEX имя_индекса;
```

Удаление индекса `PRIMARY KEY`:

```
ALTER TABLE имя_таблицы DROP PRIMARY KEY;
```

Если вы удаляете поле из таблицы, вы тем самым удаляете это поле из индекса. Если удалить все индексированные поля, то удалится весь индекс.

Правильный выбор поля для индексирования

Итак, в предыдущих разделах мы узнали, что такое индексы, для чего они нужны и как с ними работать. Но знания синтаксиса операторов недостаточно, чтобы эффективно применять индексирование для ускорения работы системы. Давайте поговорим немного о том, какие поля лучше всего подходят на роль индекса.

В качестве индексируемых нужно выбирать те поля, по которым будут производиться сортировка и группировка, а не те, которые будут выбираться. То есть индексируйте поля, которые участвуют в предложении `WHERE`, `ORDER BY` или `GROUP BY`.

Рассмотрим пример запроса:

```
mysql> SELECT title, year_issue FROM book WHERE id_author>3;
```

Поля `title` и `year_issue` не стоит индексировать, а вот поле `id_author` является кандидатом на индексирование.

Примечание

Подробное описание оператора `SELECT` приведено в *уроке 12*.

Индексы лучше работают со столбцами, содержащими уникальные значения или значения, которые повторяются редко. Например, если у вас есть поле, хранящее данные о возрасте сотрудников компании, то значения в этом поле будут повторяться нечасто. Если данные повторяются очень часто (например, какое-то значение повторяется в 40% записей), то индекс будет, в принципе, бесполезен. Например, поле, содержащее сведения о половой принадлежности сотрудников (`Male` или `Female`), не нуждается в индексировании, т. к. все равно придется просматривать всю таблицу.

С другой стороны, не стоит индексировать все, что попадется под руку. Помните, каждый созданный вами индекс занимает место на диске и замедляет операции записи. При изменениях в таблице определенные изменения происходят и с индексами, и чем их больше, тем больше времени это занимает.

Удаление и переименование таблиц

С удалением таблиц дело обстоит гораздо проще. Для удаления таблицы используется следующий оператор:

```
DROP TABLE имя_таблицы [,имя_таблицы...]
```

Например, команды

```
mysql> DROP TABLE reader;
```

достаточно, чтобы удалить таблицу `reader`.

С помощью данного оператора можно удалить как одну, так и несколько таблиц, перечислив их имена через запятую. Здесь также можно использовать ключевые слова `IF EXISTS` для предупреждения возникновения ошибки в случае, если таблица с указанным именем не существует.

На рис. 7.2 приведен пример удаления таблицы. Сначала для БД `test` (стандартная БД) был включен режим консоли, а затем создана произвольная таблица с именем `mytable`, содержащая два поля. С помощью команды `SHOW TABLES` мы просмотрели список таблиц, а затем удалили только что созданную таблицу. Следующая команда `SHOW TABLES` вывела для нас сообщение о том, что список таблиц пуст.

```
Командная строка - mysql test

C:\mysql\bin>mysql test
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7 to server version: 3.23.58-max-debug

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

mysql> CREATE TABLE mytable(id_note TINYINT(2) PRIMARY KEY NOT NULL, note VARCHAR(50));
Query OK, 0 rows affected (0.06 sec)

mysql> SHOW TABLES;
+----------------+
| Tables_in_test |
+----------------+
| mytable        |
+----------------+
1 row in set (0.00 sec)

mysql> DROP TABLE mytable;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW TABLES;
Empty set (0.00 sec)

mysql> _
```

Рис. 7.2. Удаление таблицы

```
Командная строка - mysql test
mysql> CREATE TABLE mytable(id_note TINYINT(2) PRIMARY KEY NOT NULL, note VARCHAR(50));
Query OK, 0 rows affected (0.01 sec)

mysql> SHOW TABLES;
+----------------+
| Tables_in_test |
+----------------+
| mytable        |
+----------------+
1 row in set (0.00 sec)

mysql> RENAME TABLE mytable TO mynewtable;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW TABLES;
+----------------+
| Tables_in_test |
+----------------+
| mynewtable     |
+----------------+
1 row in set (0.00 sec)

mysql>
```

Рис. 7.3. Переименование таблицы

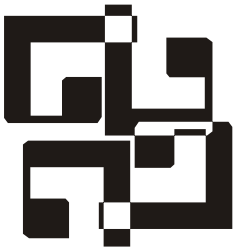
Также можно переименовать уже созданную таблицу (рис. 7.3):

```
RENAME TABLE старое_имя_таблицы TO новое_имя_таблицы
```

ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ

1. Какие операторы можно использовать для удаления индекса?
2. Какие типы индексов можно создавать в MySQL?

УРОК 8



Изменение структуры таблицы

Возможность *изменять структуру* таблицы обеспечивает оператор `ALTER TABLE`. С его помощью вы можете, например, добавлять и удалять поля таблицы, переименовывать поля или изменять их тип. Во время выполнения оператора `ALTER TABLE` MySQL создает временную копию таблицы (кроме команды `RENAME`) и все изменения производятся с копией. Затем исходная таблица удаляется, а временная переименовывается. Благодаря оператору `ALTER TABLE` можно изменять уже существующую и работающую БД. Например, некоторые типы, назначенные для полей, могут оказаться недостаточными для хранения всех значений или, наоборот, слишком большими. Или может появиться необходимость добавить еще несколько полей в таблицу.

Общий синтаксис оператора:

`ALTER TABLE` *`имя_таблицы операция1`* `[,`*`операция2`*`...]`

Операции, которые можно выполнить над таблицей:

- `ADD` *`описание_поля`* — добавляет поле к таблице (параметр *`описание_поля`* имеет такой же формат, как в операторе `CREATE TABLE`). По умолчанию поле добавляется после последнего поля таблицы. Если указать ключевое слово `FIRST`, то поле будет вставлено первым. Если указать предложение `AFTER` *`имя_поля`*, то новое поле будет вставлено после поля *`имя_поля`*. При помощи операции `ADD` вы можете добавить различные индексы:
  - `ADD FULLTEXT (`*`имя_поля`*`)` — индекс типа `FULLTEXT` (работает, начиная с MySQL 3.23.23);
  - `ADD INDEX (`*`имя_поля`*`)` — добавляет обычный индекс (неуникальный);
  - `ADD PRIMARY KEY` — добавляет первичный ключ;
  - `ADD UNIQUE` — добавляет индекс с уникальным значением;

- `ALTER` *`имя_поля`* `SET DEFAULT` *`значение`* `| DROP DEFAULT` — изменяет значение поля, установленное по умолчанию, или удаляет значение по умолчанию. Например:

  `mysql>` **`ALTER TABLE address ALTER city SET DEFAULT 'Санкт-Петербург';`**

  или

  `mysql>` **`ALTER TABLE address ALTER city DROP DEFAULT;`**

- `CHANGE` *`старое_имя_поля`* *`описание_поля`* — изменяет имя и описание поля (*`старое_имя_поля`* — имя изменяемого поля, *`описание_поля`* — его новое описание, формат такой же, как в операторе `CREATE TABLE`);
- `DROP` *`имя_поля`* — удаляет определенное поле. Также можно удалять индексы:
  - `DROP INDEX` *`имя_индекса`*;
  - `DROP PRIMARY KEY`;
- `MODIFY` *`описание_поля`* — изменяет описание поля;
- `RENAME` *`новое_имя_таблицы`* — переименовывает таблицу.

Давайте рассмотрим некоторые операции на примере. Создадим в БД `test` произвольную таблицу (рис. 8.1) и поэкспериментируем над ней.

```
Командная строка - mysql test
mysql> CREATE TABLE myuser(id_user SMALLINT(4) UNSIGNED PRIMARY KEY NOT NULL AUTO_INCREMENT, login V
ARCHAR(20));
Query OK, 0 rows affected (0.01 sec)

mysql>
```

Рис. 8.1. Создание экспериментальной таблицы

Далее добавим к нашей таблице новое поле с помощью опции `ADD` (рис. 8.2). Введите команду:

`mysql>` **`ALTER TABLE myuser ADD description text;`**

```
Командная строка - mysql test
mysql> ALTER TABLE myuser ADD description TEXT;
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC myuser;
+-------------+----------------------+------+-----+---------+----------------+
| Field       | Type                 | Null | Key | Default | Extra          |
+-------------+----------------------+------+-----+---------+----------------+
| id_user     | smallint(4) unsigned |      | PRI | NULL    | auto_increment |
| login       | varchar(20)          | YES  |     | NULL    |                |
| description | text                 | YES  |     | NULL    |                |
+-------------+----------------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)

mysql> _
```

Рис. 8.2. Добавление нового поля к таблице

С помощью команды DESC *имя_таблицы* (сокращение от DESCRIBE) можно просмотреть структуру таблицы и определить, правильно ли выполнилась какая-либо команда.

Примечание

Не забывайте ставить в конце команды символ ';'.

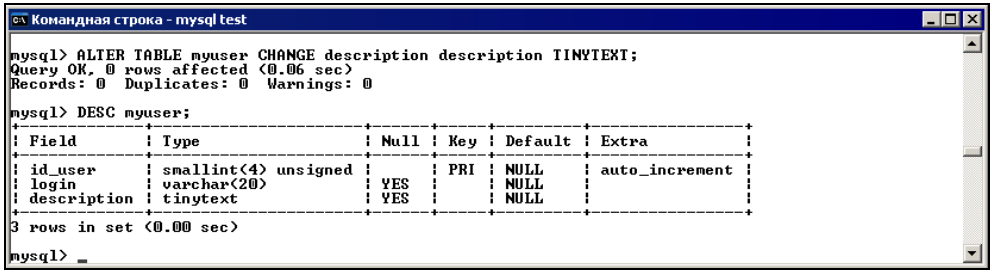
Команды языка SQL не чувствительны к регистру (я использую верхний регистр просто для удобства). То есть можно ввести команду и так: desc *имя_таблицы*;

Если требуется указать местоположение поля, используйте ключевые слова FIRST и AFTER:

```
mysql> ALTER TABLE myuser ADD description text FIRST;
mysql> ALTER TABLE myuser ADD description text AFTER id_user;
```

А сейчас мы попробуем с помощью опции CHANGE внести изменения в только что добавленное поле. Например, изменим его тип:

```
mysql> ALTER TABLE myuser CHANGE description description TINYTEXT;
```



```
Командная строка - mysql test
mysql> ALTER TABLE myuser CHANGE description description TINYTEXT;
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC myuser;
+-------------+----------------------+------+-----+---------+----------------+
| Field       | Type                 | Null | Key | Default | Extra          |
+-------------+----------------------+------+-----+---------+----------------+
| id_user     | smallint(4) unsigned |      | PRI | NULL    | auto_increment |
| login       | varchar(20)          | YES  |     | NULL    |                |
| description | tinytext             | YES  |     | NULL    |                |
+-------------+----------------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)

mysql> _
```

Рис. 8.3. Изменение типа поля description

Здесь обязательно дважды указать имя изменяемого поля. Сначала указывается имя поля, которое будет изменяться, а затем указывается полное описание поля, включая его имя.

Если ввести команду

```
mysql> ALTER TABLE myuser CHANGE description comment TINYTEXT;
```

то изменится не только тип поля, но и его имя. Если вы не собираетесь изменять имя поля, то вместо опции CHANGE можно использовать опцию MODIFY. В этом случае не придется дважды указывать имя поля.

```
mysql> ALTER TABLE myuser MODIFY description TINYTEXT;
```

Пришло время для удаления. Опция DROP позволяет удалить поле:

```
mysql> ALTER TABLE myuser DROP desrcription;
```

Этим запросом мы удалим из таблицы `myuser` поле `description`. Результат выполнения запроса представлен на рис. 8.4.

```
Командная строка - mysql test
mysql> DESC myuser;
+-------------+----------------------+------+-----+---------+----------------+
| Field       | Type                 | Null | Key | Default | Extra          |
+-------------+----------------------+------+-----+---------+----------------+
| id_user     | smallint(4) unsigned |      | PRI | NULL    | auto_increment |
| login       | varchar(20)          | YES  |     | NULL    |                |
| description | tinytext             | YES  |     | NULL    |                |
+-------------+----------------------+------+-----+---------+----------------+
3 rows in set (0.02 sec)

mysql> ALTER TABLE myuser DROP description;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE myuser DROP login;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESC myuser;
+---------+----------------------+------+-----+---------+----------------+
| Field   | Type                 | Null | Key | Default | Extra          |
+---------+----------------------+------+-----+---------+----------------+
| id_user | smallint(4) unsigned |      | PRI | NULL    | auto_increment |
+---------+----------------------+------+-----+---------+----------------+
1 row in set (0.00 sec)

mysql> ALTER TABLE myuser DROP id_user;
ERROR 1090: You can't delete all columns with ALTER TABLE. Use DROP TABLE instead
mysql> _
```

Рис. 8.4. Удаление полей

Если таблица содержит только одно поле, то его нельзя удалить. Вместо этого вы можете удалить таблицу, воспользовавшись командой `DROP TABLE`.

Наконец, можно переименовать таблицу с помощью опции `RENAME` оператора `ALTER TABLE`:

`ALTER TABLE` *`старое_имя_таблицы`* `RENAME` *`новое_имя_таблицы`*;

Для переименования можно также воспользоваться оператором `RENAME TABLE`:

`RENAME TABLE` *`старое_имя_таблицы`* `TO` *`новое_имя_таблицы`*;

```
Командная строка - mysql test
mysql> SHOW TABLES;
+----------------+
| Tables_in_test |
+----------------+
| myuser         |
+----------------+
1 row in set (0.00 sec)

mysql> ALTER TABLE myuser RENAME mynewuser;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW TABLES;
+----------------+
| Tables_in_test |
+----------------+
| mynewuser      |
+----------------+
1 row in set (0.01 sec)

mysql> _
```

Рис. 8.5. Переименование таблицы при помощи опции `RENAME`

В отличие от переименования с помощью оператора `ALTER TABLE`, оператор `RENAME TABLE` позволяет переименовать сразу несколько таблиц:

```
RENAME TABLE таблица1 TO врем_табл, таблица2 TO таблица1, врем_табл
  TO таблица2;
```

Примечание

Нельзя переименовать таблицу, если ее новое имя совпадает с именем другой существующей таблицы.

Итак, теперь вы можете создавать, удалять и редактировать таблицы. Нам вновь нужно вернуться к учебной БД и создать все таблицы, которые мы спроектировали. Чтобы немного ускорить процесс, создайте в каталоге C:\MYSQL\BIN текстовый файл (например, LIBRARY.TXT) или скопируйте его с сопроводительного компакт-диска книги. В нем мы опишем набор команд необходимый для создания таблиц, а потом с помощью утилиты MYSQL.EXE выполним их все за один раз. В коде можно оставлять однострочные комментарии, используя символы `'--'`. В листинге 8.1 приведен код, создающий все таблицы для БД `library`.

Листинг 8.1

```
CREATE TABLE abonement (
  id_note MEDIUMINT(5) UNSIGNED PRIMARY KEY NOT NULL AUTO_INCREMENT,
  id_reader MEDIUMINT(5) UNSIGNED,
  id_unit mediumint(6) UNSIGNED,
  get_date DATE,
  exp_date DATE);

CREATE TABLE address (
  id_addr MEDIUMINT(5) UNSIGNED PRIMARY KEY NOT NULL AUTO_INCREMENT,
  address VARCHAR(80),
  phone VARCHAR(19));

CREATE TABLE reader (
  id_reader MEDIUMINT(5) UNSIGNED PRIMARY KEY NOT NULL AUTO_INCREMENT,
  reader VARCHAR(60),
  id_addr MEDIUMINT(5));

CREATE TABLE section (
  id_section TINYINT(2) UNSIGNED PRIMARY KEY NOT NULL AUTO_INCREMENT,
  section VARCHAR(30));
```

```
CREATE TABLE publisher (
  id_publisher SMALLINT(3) UNSIGNED PRIMARY KEY NOT NULL AUTO_INCREMENT,
  publisher VARCHAR(30));

CREATE TABLE author (
  id_author SMALLINT(4) UNSIGNED PRIMARY KEY NOT NULL AUTO_INCREMENT,
author VARCHAR(60));

CREATE TABLE unit (
  id_unit MEDIUMINT(6) UNSIGNED PRIMARY KEY NOT NULL AUTO_INCREMENT,
  id_book MEDIUMINT(5));

CREATE TABLE book (
  id_book MEDIUMINT(5) UNSIGNED PRIMARY KEY NOT NULL AUTO_INCREMENT,
  title VARCHAR(50),
  year_issue YEAR,
  id_author SMALLINT(4) UNSIGNED,
  id_publisher SMALLINT(3) UNSIGNED,
  id_section TINYINT(2));
```

После этого нужно выполнить все команды, занесенные в файл, с помощью утилиты MYSQL.EXE (рис. 8.6). Для выхода из режима консоли используйте уже знакомую вам команду `exit`:

```
mysql> exit
Bye
C:\mysql\bin>
```

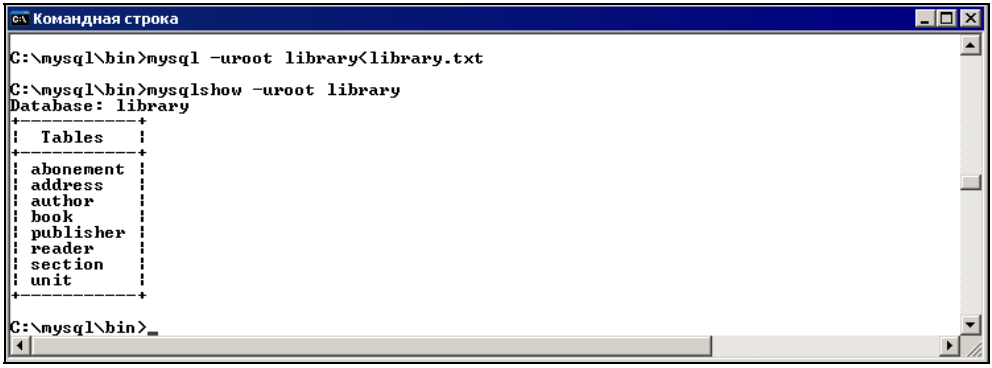


Рис. 8.6. Выполнение команд из файла LIBRARY.TXT

Утилита MYSQLSHOW.EXE поможет вам убедиться, что все выполнилось правильно.

После создания БД рекомендую сделать дамп (сохранение) на случай сбоя операционной системы, нечаянного удаления каких-либо файлов и других стихийных бедствий:

```
C:\mysql\bin> mysqldump -uroot --add-drop-table library>db_library.txt
```

В листинге 8.2 приведено содержимое файла DB_LIBRARY.TXT.

Листинг 8.2

```
-- MySQL dump 8.23
--
-- Host: localhost    Database: library
--------------------------------------------------------
-- Server version  3.23.58-max-debug

--
-- Table structure for table 'abonement'
--

DROP TABLE IF EXISTS abonement;
CREATE TABLE abonement (
  id_note mediumint(5) unsigned NOT NULL auto_increment,
  id_reader mediumint(5) unsigned default NULL,
  id_unit mediumint(6) unsigned default NULL,
  get_date date default NULL,
  exp_date date default NULL,
  PRIMARY KEY  (id_note)
) TYPE=MyISAM;

--
-- Dumping data for table 'abonement'
--

--
-- Table structure for table 'address'
--

DROP TABLE IF EXISTS address;
CREATE TABLE address (
  id_addr mediumint(5) unsigned NOT NULL auto_increment,
  address varchar(80) default NULL,
  phone varchar(19) default NULL,
  PRIMARY KEY  (id_addr)
) TYPE=MyISAM;
```

```
--
-- Dumping data for table 'address'
--

--
-- Table structure for table 'author'
--

DROP TABLE IF EXISTS author;
CREATE TABLE author (
  id_author smallint(4) unsigned NOT NULL auto_increment,
  author varchar(60) default NULL,
  PRIMARY KEY  (id_author)
) TYPE=MyISAM;

--
-- Dumping data for table 'author'
--

--
-- Table structure for table 'book'
--

DROP TABLE IF EXISTS book;
CREATE TABLE book (
  id_book mediumint(5) unsigned NOT NULL auto_increment,
  title varchar(50) default NULL,
  year_issue year(4) default NULL,
  id_author smallint(4) unsigned default NULL,
  id_publisher smallint(3) unsigned default NULL,
  id_section tinyint(2) default NULL,
  PRIMARY KEY  (id_book)
) TYPE=MyISAM;

--
-- Dumping data for table 'book'
--

--
-- Table structure for table 'publisher'
--

DROP TABLE IF EXISTS publisher;
CREATE TABLE publisher (
  id_publisher smallint(3) unsigned NOT NULL auto_increment,
```

```
  publisher varchar(30) default NULL,
  PRIMARY KEY  (id_publisher)
) TYPE=MyISAM;

--
-- Dumping data for table 'publisher'
--

--
-- Table structure for table 'reader'
--

DROP TABLE IF EXISTS reader;
CREATE TABLE reader (
  id_reader mediumint(5) unsigned NOT NULL auto_increment,
  reader varchar(60) default NULL,
  id_addr mediumint(5) default NULL,
  PRIMARY KEY  (id_reader)
) TYPE=MyISAM;

--
-- Dumping data for table 'reader'
--

--
-- Table structure for table 'section'
--

DROP TABLE IF EXISTS section;
CREATE TABLE section (
  id_section tinyint(2) unsigned NOT NULL auto_increment,
  section varchar(30) default NULL,
  PRIMARY KEY  (id_section)
) TYPE=MyISAM;

--
-- Dumping data for table 'section'
--

--
-- Table structure for table 'unit'
--

DROP TABLE IF EXISTS unit;
CREATE TABLE unit (
  id_unit mediumint(6) unsigned NOT NULL auto_increment,
```

```
  id_book mediumint(5) default NULL,
  PRIMARY KEY  (id_unit)
) TYPE=MyISAM;

--
-- Dumping data for table 'unit'
--
```

Как видите, СУБД MySQL установила для полей значение по умолчанию (`NULL`), проставила тип таблицы `MyISAM`, а также, благодаря установленной опции `--add-drop-table`, перед каждой таблицей появилась команда `DROP TABLE IF EXISTS` *`имя_таблицы`*. Если в дальнейшем вы измените содержимое файла и выполните его, то MySQL удалит таблицу с таким именем и заменит ее той, что указана в файле. То есть вы можете вносить любые изменения в структуру таблиц и заносить их в БД, выполняя этот файл при помощи утилиты MYSQL.EXE.

Примечание

Напоминаю, что необходимость в опции `--add-drop-table` отпадает при работе с MySQL версии 4.*xx* и выше.

Мы научились работать с таблицами, но база данных с пустыми таблицами вряд ли кому-то нужна. Пришло время узнать, как заносить данные в таблицу. Мы рассмотрим это в следующем уроке.

ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ

1. Какая опция оператора `ALTER TABLE` позволяет удалить поле таблицы?
2. Какие ключевые слова используются для указания позиции добавляемого поля?

УРОК 9

Добавление данных в таблицу

Добавлять записи в таблицу можно с помощью команд `LOAD DATA` и `INSERT`. Команда `INSERT` позволяет добавить в таблицу одну запись, а команда `LOAD DATA` — сразу несколько записей, загружаемых из текстового файла. Рассмотрим эти команды подробнее.

Общий синтаксис команды `INSERT`:

```
INSERT [INTO] [IGNORE] [LOW PRIORITY | DELAYED]
  имя_таблицы [(имя_поля,...)] VALUES (выражение,...);
```

Параметры: *имя_таблицы* — таблица, в которую добавляется запись; `(`*выражение*`,...)` — список добавляемых значений (значения перечисляются в том же порядке, что и поля таблицы); `(`*имя_поля*`,...)` — список полей, в которые вносятся данные. Эта часть параметров указана в квадратных скобках (`[]`), что говорит о необязательности ее присутствия в команде. Список полей указывается только тогда, когда таблица заполняется не полностью, в ином случае эта часть не присутствует. То есть если у нас есть таблица `myuser`:

myuser		
id_user	login	password
...	...	...

и мы хотим заполнить только два поля из трех (`id_user` и `login`), то команда будет выглядеть так:

```
mysql> INSERT INTO myuser(id_user,login) VALUES(1,'login1');
```

В поля таблицы, не указанные в списке, вставляется значение по умолчанию (свое для каждого поля). Если заполняются все поля, то перечислять их имена совсем не обязательно:

```
mysql> INSERT INTO myuser VALUES(1,'login1','password1');
```

Ключевое слово `INTO` не является обязательным (его необходимо использовать только в MySQL 3.22.5). Если указать ключевое слово `LOW PRIORITY`, то оператор не выполняется, пока хотя бы один пользователь читает данные из таблицы. При задании ключевого слова `DELAYED` строки будут помещены в очередь для более поздней вставки, и оператор прекратит выполнение, что позволит клиенту продолжить работу.

Если при заполнении таблицы явно не задается значение какого-либо поля, то в него будет занесено значение по умолчанию, определенное вами или MySQL при создании таблицы.

В том случае, если вносятся дублирующие значения в поля `PRIMARY KEY` или `UNIQUE`, действие команды прекращается и выдается предупреждение (рис. 9.1). Ключевое слово `IGNORE` позволит проигнорировать вставку дублирующих значений.

```
Командная строка - mysql library

mysql> DESC reader;
+-----------+----------------------+------+-----+---------+----------------+
| Field     | Type                 | Null | Key | Default | Extra          |
+-----------+----------------------+------+-----+---------+----------------+
| id_reader | mediumint(5) unsigned |      | PRI | NULL    | auto_increment |
| reader    | varchar(60)          | YES  |     | NULL    |                |
| id_addr   | mediumint(5)         | YES  |     | NULL    |                |
+-----------+----------------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)

mysql> INSERT INTO reader VALUES(1,'Вася',1);
Query OK, 1 row affected (0.02 sec)

mysql> INSERT INTO reader VALUES(1,'Вася',1);
ERROR 1062: Duplicate entry '1' for key 1
mysql>
```

Рис. 9.1. Ошибка при дублировании значений

Кроме этого, предупреждения выдаются в следующих ситуациях:

- внесение значения `NULL` в поле, которое было объявлено как `NOT NULL`; в данное поле будет установлено значение, заданное по умолчанию;
- установка числового поля в значение, которое находится за пределами его допустимого диапазона; данное значение будет усечено до соответствующей конечной точки этого диапазона;
- внесение в числовое поле такой величины, как `'32 ba'`; конечные данные удаляются, и вносится только оставшаяся числовая часть; если вводимую величину невозможно интерпретировать как число, то значение поля устанавливается в 0;
- внесение в поле типа `CHAR`, `VARCHAR`, `TEXT` или `BLOB` строки, длина которой превышает максимальную длину поля; данная величина усекается до максимальной длины поля;
- внесение в поле даты или времени строки, недопустимой для данного типа столбца; это поле будет установлено в нулевое значение, соответствующее данному типу.

Для внесения данных в определенные поля также можно использовать синтаксис:

```
INSERT INTO имя_таблицы SET имя_поля=значение[, имя_поля=значение,...];
```

Здесь оператор `INSERT` вставляет в поля, определенные в предложении `SET`, заданные значения. Если поля не определены, в них вставляется значение по умолчанию.

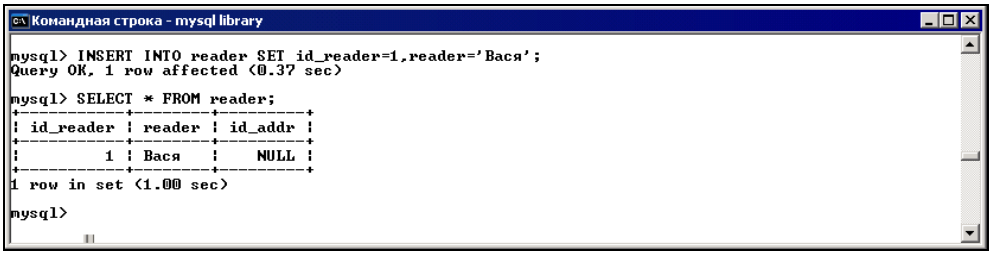


Рис. 9.2. Использование синтаксиса `INSERT ... SET`

Выполнив команду `SELECT`, вы сможете просмотреть результаты внесения данных в таблицу:

```
mysql> SELECT * FROM reader;
```

Команда `LOAD DATA` актуальна, если требуется внести в таблицу большое количество записей:

```
LOAD DATA [LOCAL] INFILE 'имя_файла.txt' [IGNORE | REPLACE]
  INTO TABLE имя_таблицы
```

Эта команда читает строки из текстового файла и вставляет их в таблицу с очень высокой скоростью. Файл должен находиться в каталоге данных сервера (в нашем случае это C:\MYSQL\DATA\LIBRARY). Без ключевого слова `LOCAL` файл считывается с того компьютера, где установлен сервер MySQL. Обычно возможность использования ключевого слова `LOCAL` на стороне сервера отключена.

По умолчанию команда работает так:

1. Ищет конец строки в виде символов `'\n'`.
2. Разбивает строки на поля по символу табуляции.

Например, создадим в указанном каталоге файл 1.TXT, записав в него несколько строк со значениями, разделенными табуляцией:

```
1  Denis     1
2  Oleg      2
3  Nina      3
```

Можно сразу поместить их в таблицу с помощью команды `LOAD DATA` (рис. 9.3):

```
mysql> LOAD DATA INFILE '1.txt' INTO TABLE reader;
```

```
Командная строка - mysql library

mysql> LOAD DATA INFILE '1.txt' INTO TABLE reader;
Query OK, 3 rows affected (0.01 sec)
Records: 3  Deleted: 0  Skipped: 0  Warnings: 0

mysql> SELECT * FROM reader;
+-----------+---------+---------+
| id_reader | reader  | id_addr |
+-----------+---------+---------+
|         1 | Denis   |       1 |
|         2 | Oleg    |       2 |
|         3 | Nina    |       3 |
+-----------+---------+---------+
3 rows in set (0.00 sec)

mysql>
```

Рис. 9.3. Выполнение команды `LOAD DATA`

Если вам нужно пропустить несколько строк из файла, используйте предложение `IGNORE` *n* `LINES`. Например, это понадобится, если наш файл выглядит так:

```
id_reader  reader    id_addr
1    Denis       1
2    Oleg        2
3    Nina        3
```

После выполнения команды

```
mysql> LOAD DATA INFILE '1.txt' INTO TABLE reader IGNORE 1 LINES;
```

все данные из текстового файла будут помещены в таблицу. При указании ключевых слов `IGNORE` и `REPLACE` дублирующие значения для уникальных индексов либо игнорируются, либо заменяют уже существующие.

В режиме консоли также можно заносить в таблицу (например, `section`) данные следующим образом (рис. 9.4):

```
mysql> INSERT INTO section VALUES
-> (1,'Детектив'),
-> (2,'Фантастика'),
-> (3,'Сказка');
```

Теперь вы знаете, как можно заполнить таблицы, находящиеся в БД.

Необходимо заполнить нашу учебную БД, внеся хотя бы по несколько записей в каждую таблицу, и тогда мы сможем с ней работать. На сопроводительном компакт-диске книги есть файл DATA_FOR_LIBRARY.TXT с командами `INSERT` для каждой таблицы. Текст файла записан в кодировке Win-

dows-1251, поэтому при просмотре содержимого таблиц посредством командной строки кириллица может отображаться некорректно (в дальнейшем, при выводе данных в браузере все будет отображаться верно). Поэтому для тестирования операторов языка SQL (`DELETE`, `UPDATE`, `SELECT`) вводите данные непосредственно в командной строке. Напоминаю, что для выполнения команд из файла необходимо ввести в командной строке такую команду:

```
C:\mysql\bin> mysql -uroot library<data_for_library.txt
```

Параметр `-uroot` не нужен, если вы работаете с правами администратора, но если вы его укажете, хуже не будет. Я рекомендую, конечно же, заполнить таблицы самостоятельно, закрепив пройденный материал.

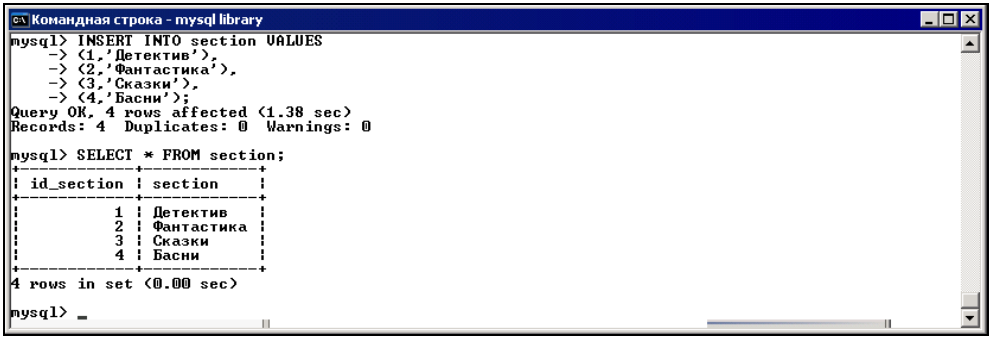


Рис. 9.4. Занесение данных в таблицу `section`

УРОК 10

Удаление данных

Если вы допустили какие-то ошибки при занесении данных или занесли в таблицу то, что в ней не должно находиться, то потребуется удалить эти неверные данные. В этом уроке мы рассмотрим удаление записей из таблицы. Также эти знания могут понадобиться в дальнейшем при написании приложений на языке PHP, обслуживающих БД.

Для удаления записей из таблицы предназначен оператор `DELETE`:

```
DELETE FROM имя_таблицы [WHERE условие];
```

Оператор `DELETE` удаляет из таблицы *имя_таблицы* строки, которые удовлетворяют заданным в параметре `WHERE` условиям. Условия `WHERE` не являются обязательными, но используются в большинстве запросов на удаление. Если, например, ввести команду

```
mysql> DELETE FROM reader;
```

то будут удалены *все* записи таблицы `reader` ("Читатель"), что вряд ли нам нужно. Какие же условия мы можем задавать при удалении записей? Давайте рассмотрим это подробнее.

Параметр *условие* задается как *имя_поля*='*значение*'. Например, запрос

```
mysql> DELETE FROM reader WHERE id_reader=1;
```

означает "удалить из таблицы `reader` ту запись, где значение поля `id_reader` равно `1`". Этим запросом мы удалим запись только для одного читателя с определенным номером (номер уникальный, т. к. `id_reader` является первичным ключом — `PRIMARY KEY`).

При описании условия кроме знака равенства (=) можно использовать следующие операции:

- > ("больше, чем"):

  mysql> **DELETE FROM book WHERE year_issue>2000;**

  После выполнения этого запроса будут удалены записи книг начиная с 2001 года выпуска (рис. 10.1);

```
Командная строка - mysql -uroot library

mysql> SELECT * FROM book;
+---------+-------------------+------------+----------+-------------+-----------+
| id_book | title             | year_issue | id_author | id_publisher | id_section |
+---------+-------------------+------------+----------+-------------+-----------+
|       1 | Нос               |       1990 |        1 |           1 |         2 |
|       2 | Незнайка на луне  |       1995 |        2 |           1 |         3 |
|       3 | Ночной дозор      |       2004 |        4 |           1 |         1 |
|       4 | Властелин колец   |       2001 |       21 |           3 |         5 |
+---------+-------------------+------------+----------+-------------+-----------+
4 rows in set (0.00 sec)

mysql> DELETE FROM book WHERE year_issue>2000;
Query OK, 2 rows affected (0.03 sec)

mysql> SELECT * FROM book;
+---------+-------------------+------------+----------+-------------+-----------+
| id_book | title             | year_issue | id_author | id_publisher | id_section |
+---------+-------------------+------------+----------+-------------+-----------+
|       1 | Нос               |       1990 |        1 |           1 |         2 |
|       2 | Незнайка на луне  |       1995 |        2 |           1 |         3 |
+---------+-------------------+------------+----------+-------------+-----------+
2 rows in set (0.00 sec)

mysql> _
```

Рис. 10.1. Удаление записей из таблицы book

- >= ("больше или равно"):

  mysql> **DELETE FROM reader WHERE id_reader>=25;**

  После выполнения этого запроса будут удалены записи читателей с номерами, больше или равными 25;

- < ("меньше, чем");
- <= ("меньше или равно");
- <> или != ("не равно):

  mysql> **DELETE FROM author WHERE id_author<>5;**

  В этом случае удаляются все записи авторов, номера которых не равны 5 (у нас осталась только одна запись, рис. 10.2);

- IS NULL (поле содержит значение NULL) (рис. 10.3):

  mysql> **DELETE FROM reader WHERE reader IS NULL;**

  Значение NULL можно установить в поле, если на момент внесения записи вам неизвестно, какие данные будут храниться в этом поле. По сути, NULL обозначает отсутствующее или неизвестное значение;

Примечание

Нельзя проверять значение на равенство NULL при помощи обычных операций сравнения (=, > или !=).

```
Командная строка - mysql -uroot library
mysql> SELECT * FROM author;
+-----------+-------------+
| id_author | author      |
+-----------+-------------+
|         1 | Пушкин      |
|         2 | Гоголь      |
|         3 | Лермонтов   |
|         4 | Достоевский |
|         5 | Толстой     |
|         6 | Чехов       |
+-----------+-------------+
6 rows in set (0.02 sec)

mysql> DELETE FROM author WHERE id_author<>5;
Query OK, 5 rows affected (0.03 sec)

mysql> SELECT * FROM author;
+-----------+---------+
| id_author | author  |
+-----------+---------+
|         5 | Толстой |
+-----------+---------+
1 row in set (0.00 sec)

mysql> _
```

Рис. 10.2. Удаление записей из таблицы `author`

```
Командная строка - mysql -uroot library
mysql> SELECT * FROM reader;
+-----------+-----------+---------+
| id_reader | reader    | id_addr |
+-----------+-----------+---------+
|         4 | Задоренков |       2 |
|         3 | Сидоров   |       1 |
|         2 | NULL      |       3 |
|         1 | Иванов    |       4 |
+-----------+-----------+---------+
4 rows in set (0.00 sec)

mysql> DELETE FROM reader WHERE reader IS NULL;
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM reader;
+-----------+-----------+---------+
| id_reader | reader    | id_addr |
+-----------+-----------+---------+
|         4 | Задоренков |       2 |
|         3 | Сидоров   |       1 |
|         1 | Иванов    |       4 |
+-----------+-----------+---------+
3 rows in set (0.00 sec)

mysql> _
```

Рис. 10.3. Удаление записей с полем, содержащим значение `NULL`

```
Командная строка - mysql -uroot library

mysql> SELECT * FROM reader;
+-----------+-----------+---------+
| id_reader | reader    | id_addr |
+-----------+-----------+---------+
|         4 | Задоренков |       2 |
|         3 | Сидоров   |       1 |
|         2 | NULL      |       3 |
|         1 | Иванов    |       4 |
+-----------+-----------+---------+
4 rows in set (0.00 sec)

mysql> DELETE FROM reader WHERE reader IS NOT NULL;
Query OK, 3 rows affected (0.00 sec)

mysql> SELECT * FROM reader;
+-----------+--------+---------+
| id_reader | reader | id_addr |
+-----------+--------+---------+
|         2 | NULL   |       3 |
+-----------+--------+---------+
1 row in set (0.00 sec)

mysql> _
```

Рис. 10.4. Удаление записей с полем `reader`, значение которого отлично от `NULL`

- `IS NOT NULL` (поле содержит значение, отличное от `NULL`) (рис. 10.4):

  `mysql>` **`DELETE FROM reader WHERE reader IS NOT NULL;`**

Сравнение по шаблону

В MySQL реализовано сравнение по шаблонам:

- `LIKE '`*шаблон*`'` — значение поля соответствует шаблону;
- `NOT LIKE '`*шаблон*`'` — значение поля не соответствует шаблону.

При описании можно использовать символы шаблона:

- `'_'` — соответствует любому одиночному символу;
- `'%'` — соответствует любому количеству символов.

В MySQL шаблоны по умолчанию не чувствительны к регистру символов.

Например, если вы хотите удалить записи всех авторов, фамилии которых начинаются с буквы `'Л'`, то введите такой запрос (рис. 10.5):

`mysql>` **`DELETE FROM author WHERE author LIKE 'Л%';`**

Примечание

Постарайтесь по возможности избегать шаблонов, начинающихся с символа `'%'`. СУБД тратит довольно много времени на поиск значений, соответствующих такому шаблону. Примеры запросов с таким шаблоном представлены на рис. 10.6 и 10.7.

```
Командная строка - mysql -uroot library

mysql> SELECT * FROM author;
+-----------+--------------+
| id_author | author       |
+-----------+--------------+
|         6 | Чехов        |
|         4 | Достоевский  |
|         3 | Лермонтов    |
|         2 | Гоголь       |
|         5 | Толстой      |
|         1 | Пушкин       |
|         7 | Лукьяненко   |
+-----------+--------------+
7 rows in set (0.00 sec)

mysql> DELETE FROM author WHERE author LIKE 'Л%';
Query OK, 2 rows affected (0.00 sec)

mysql> SELECT * FROM author;
+-----------+--------------+
| id_author | author       |
+-----------+--------------+
|         6 | Чехов        |
|         4 | Достоевский  |
|         2 | Гоголь       |
|         5 | Толстой      |
|         1 | Пушкин       |
+-----------+--------------+
5 rows in set (0.00 sec)

mysql> _
```

Рис. 10.5. Удаление по шаблону `'Л%'`

Если нужно удалить записи авторов, фамилии которых заканчиваются на `'й'`, введите запрос (рис. 10.6):

```
mysql> DELETE FROM author WHERE author LIKE '%й';
```

```
Командная строка - mysql -uroot library
+------------+-------------+
| id_author  | author      |
+------------+-------------+
|          1 | Пушкин      |
|          2 | Гоголь      |
|          3 | Достоевский |
|          4 | Толстой     |
|          5 | Лермонтов   |
+------------+-------------+
5 rows in set (0.00 sec)

mysql> DELETE FROM author WHERE author LIKE '%й';
Query OK, 2 rows affected (0.03 sec)

mysql> SELECT * FROM author;
+------------+-----------+
| id_author  | author    |
+------------+-----------+
|          1 | Пушкин    |
|          2 | Гоголь    |
|          5 | Лермонтов |
+------------+-----------+
3 rows in set (0.00 sec)

mysql> _
```

Рис. 10.6. Удаление по шаблону `'%й'`

А если нужно удалить записи авторов, в фамилиях которых есть буква `'о'`, введите такой запрос (рис. 10.7):

```
mysql> DELETE FROM author WHERE author LIKE '%о%';
```

```
Командная строка - mysql -uroot library

mysql> SELECT * FROM author;
+------------+-----------+
| id_author  | author    |
+------------+-----------+
|          1 | Пушкин    |
|          2 | Гоголь    |
|          5 | Лермонтов |
+------------+-----------+
3 rows in set (0.00 sec)

mysql> DELETE FROM author WHERE author LIKE '%о%';
Query OK, 2 rows affected (0.16 sec)

mysql> SELECT * FROM author;
+------------+--------+
| id_author  | author |
+------------+--------+
|          1 | Пушкин |
+------------+--------+
1 row in set (0.02 sec)

mysql> _
```

Рис. 10.7. Удаление по шаблону `'%о%'`

Удалить записи авторов, фамилии которых состоят из шести букв, можно с помощью следующего запроса:

```
mysql> DELETE FROM author WHERE author LIKE '______';
```

Здесь шаблон состоит из шести символов `'_'`. Поскольку значения `'Пушкин'` и `'Гоголь'` совпадают с этим шаблоном, система удаляет их из таблицы (рис. 10.8).

```
Командная строка - mysql -uroot library
mysql> SELECT * FROM author;
+-----------+-------------+
| id_author | author      |
+-----------+-------------+
|         1 | Пушкин      |
|         4 | Толстой     |
|         3 | Гоголь      |
|         5 | Лермонтов   |
|         2 | Достоевский |
+-----------+-------------+
5 rows in set (0.00 sec)

mysql> DELETE FROM author WHERE author LIKE '______';
Query OK, 2 rows affected (0.01 sec)

mysql> SELECT * FROM author;
+-----------+-------------+
| id_author | author      |
+-----------+-------------+
|         4 | Толстой     |
|         5 | Лермонтов   |
|         2 | Достоевский |
+-----------+-------------+
3 rows in set (0.00 sec)

mysql> _
```

Рис. 10.8. Удаление по шаблону `'_____'`

Расширенные регулярные выражения

Кроме описанных ранее, вы можете применять шаблоны другого вида, основанные на *расширенных регулярных выражениях*. В этом случае для проверки значений на соответствие или несоответствие шаблону используются операторы `REGEXP` и `NOT REGEXP` соответственно. Также можно применять их синонимы `RLIKE` и `NOT RLIKE`.

При составлении шаблона допустимы следующие выражения:

- . (точка) — любой символ;
- `[...]` (квадратные скобки) — любой из символов, перечисленных в скобках (класс символов). Например, `'[ab2]'` обозначает букву `'a'` или `'b'` или цифру `'2'`;

Примечание

Класс символов — это набор символов, обозначающий один символ из перечисленного набора. В классе можно задавать диапазоны, например, класс `[a-z]`

обозначает любую букву в нижнем регистре, а класс `[0-9]` — любую цифру. Есть возможность комбинировать диапазоны в классе, например, класс `[a-z0-9]` обозначает любую букву или цифру.

- `+` — одно или более вхождений символа. То есть шаблон `'[0-9]+'` обозначает одно или более вхождений любой цифры, шаблон `'a+'` — одно или более вхождений буквы `'a'`, а шаблон `'.+'` — одно или более вхождений любого символа;
- `*` — ноль или больше вхождений символа. Отсюда следует, что шаблон `'[a-z]*'` — это любое количество букв;
- `?` — ноль или одно вхождение символа;
- `{n}` — определенное число вхождений символа. Например, шаблон `'[0-9]{4}'` обозначает четыре любых цифры;
- `{n,}` — `n` и больше вхождений символа;
- `{n,m}` — от `n` до `m` вхождений символа (т. е. задаются минимальное и максимальное количества вхождений символа);
- `^` — привязка к началу строки. Например, шаблон `'^[0-9]{4}'` будет искать наличие четырех цифр в начале строки;
- `$` — привязка к концу строки. Шаблон `'[a-zA-Z]{2}$'` будет искать наличие двух букв в верхнем или нижнем регистре, начиная с конца строки.

Примечание

В MySQL версий до 3.23.4 оператор `REGEXP` учитывает регистр при сравнении по шаблону. Более поздние версии не учитывают регистр символов. Но можно "заставить" `REGEXP` учитывать регистр, если добавить ключевое слово `BINARY`.

Давайте теперь рассмотрим на примерах, как все это работает. Составим несколько запросов, использующих при удалении сравнение значений на соответствие шаблону.

```
mysql> DELETE FROM book WHERE title REGEXP BINARY '^[а-я]';
```

Этот запрос удалил все записи всех книг, названия которых начинаются с буквы в нижнем регистре, — мы лишились `'буратино'` (рис. 10.9).

```
mysql> DELETE FROM book WHERE title RLIKE '^.{3}$';
```

Этот запрос удалил записи книг, названия которых состоят из трех символов (`'Нос'`) (рис. 10.10).

```
mysql> DELETE FROM book WHERE title RLIKE '[0-9]+';
```

Этот запрос позволил нам удалить записи книг, в названиях которых встречаются цифры (рис. 10.11).

```
Командная строка - mysql library
| title                        |
+------------------------------+
| Нос                          |
| Незнайка на луне             |
| Война и мир                  |
| буратино                     |
| Путешествие к центру земли   |
+------------------------------+
5 rows in set (0.00 sec)

mysql> DELETE FROM book WHERE title REGEXP BINARY '^[а-я]';
Query OK, 1 row affected (0.01 sec)

mysql> SELECT title FROM book;
+------------------------------+
| title                        |
+------------------------------+
| Нос                          |
| Незнайка на луне             |
| Война и мир                  |
| Путешествие к центру земли   |
+------------------------------+
4 rows in set (0.00 sec)

mysql> _
```

Рис. 10.9. Удаление по шаблону с учетом регистра символов

```
Командная строка - mysql library
| title                        |
+------------------------------+
| Нос                          |
| Путешествие к центру земли   |
| Буратино                     |
| Война и мир                  |
| Незнайка на луне             |
+------------------------------+
5 rows in set (0.00 sec)

mysql> DELETE FROM book WHERE title RLIKE '^.{3}$';
Query OK, 1 row affected (0.00 sec)

mysql> SELECT title FROM book;
+------------------------------+
| title                        |
+------------------------------+
| Путешествие к центру земли   |
| Буратино                     |
| Война и мир                  |
| Незнайка на луне             |
+------------------------------+
4 rows in set (0.00 sec)

mysql> _
```

Рис. 10.10. Удаление по шаблону `'^.{3}$'`

```
mysql> DELETE FROM book WHERE title RLIKE '[а-я]+( [а-я]+)+';
```

А здесь мы удалили записи книг, название которых состоит более чем из одного слова. Первая часть шаблона `'[а-я]+'` описывает слово (одну или несколько букв, идущих подряд), далее следуют пробел и еще одно слово `'( [а-я]+)+'`. Но последнюю часть шаблона мы поместили в круглые скобки и применили к этому выражению оператор `+`, обозначающий, что оно может повториться еще несколько раз. То есть после первого слова в названии кни-

ги должно обязательно следовать еще одно или более слов, разделенных символом пробела. Поэтому в результате запроса в таблице осталась лишь книга с названием `'Буратино'` (рис. 10.12).

```
Командная строка - mysql library
+-----------------------------+
| 1984                        |
| Путешествие к центру земли  |
| Буратино                    |
| Война и мир                 |
| Незнайка на луне            |
| 7 способов похудеть         |
+-----------------------------+
6 rows in set (0.00 sec)

mysql> DELETE FROM book WHERE title RLIKE '[0-9]+';
Query OK, 2 rows affected (0.00 sec)

mysql> SELECT title FROM book;
+-----------------------------+
| title                       |
+-----------------------------+
| Путешествие к центру земли  |
| Буратино                    |
| Война и мир                 |
| Незнайка на луне            |
+-----------------------------+
4 rows in set (0.02 sec)

mysql>
```

Рис. 10.11. Удаление по шаблону `'[0-9]+'`

```
Командная строка - mysql library
mysql> SELECT title FROM book;
+-----------------------------+
| title                       |
+-----------------------------+
| Путешествие к центру земли  |
| Буратино                    |
| Война и мир                 |
| Незнайка на луне            |
+-----------------------------+
4 rows in set (0.02 sec)

mysql> DELETE FROM book WHERE title RLIKE '[а-я]+( [а-я]+)+';
Query OK, 3 rows affected (0.00 sec)

mysql> SELECT title FROM book;
+-----------+
| title     |
+-----------+
| Буратино  |
+-----------+
1 row in set (0.00 sec)

mysql>
```

Рис. 10.12. Удаление по шаблону `'[а-я]+( [а-я]+)+'`

Полную очистку таблицы можно выполнить с помощью оператора `TRUNCATE`:

`TRUNCATE [TABLE]` *`имя_таблицы`*`;`

Он работает быстрее, нежели построчное удаление таблиц. Ключевое слово `TABLE` необходимо опускать в MySQL версии до 3.23.33.

Логические операторы

Если вы хотите указать несколько условий при удалении записей, то необходимо перечислить их с помощью *логического ИЛИ (OR)* или объединить с помощью *логического И (AND)*. Например:

```
mysql> DELETE FROM address WHERE phone IS NULL OR phone NOT LIKE '+7%';
mysql> DELETE FROM book WHERE title='Война и мир' AND year_issue=1993;
```

Первый запрос удаляет из таблицы с адресами те записи, где поле `phone` содержит значение `NULL` или номер, который не начинается с `'+7'`.

Второй запрос удаляет записи книг с названием `'Война и мир'` и годом выпуска `1993`.

УРОК 11

Изменение данных

Изменение (обновление) содержимого таблиц выполняется при помощи оператора `UPDATE`:

```
UPDATE имя_таблицы SET имя_поля1=значение1 [, имя_поля2=значение2, ...]
   [WHERE условие]
```

Параметры: *имя_таблицы* — имя таблицы, поля которой будут обновляться; *имя_поля* — обновляемое поле; *значение* — новое значение поля; *условие* — условие, описывающее записи, которые требуется обновить.

Таким образом, оператор `UPDATE` обновляет в таблице *имя_таблицы* поле *имя_поля*, устанавливая его в *значение*, для тех записей, которые удовлетворяют условию `условие`.

Можно обновить сразу несколько полей, перечислив пары *имя_поля=значение* через запятую. Например, обновим какую-нибудь запись в таблице `author` (рис. 11.1):

```
mysql> UPDATE author SET author='Тургенев' WHERE id_author=3;
```

Примечание

Если поле устанавливается в его текущее значение, то MySQL не станет обновлять запись.

А теперь давайте обновим несколько полей в одной таблице (рис. 11.2):

```
mysql> UPDATE address SET address='ул. Мира, 32/15', phone='320-54-54'
  WHERE id_addr=2;
```

Данный запрос изменяет значения адреса и телефона в записи с номером адреса, равным 2. Это, например, может понадобиться, если читатель переехал на другое место жительства.

```
Командная строка - mysql -uroot library
| id_author | author      |
+-----------+-------------+
|         1 | Тютчев      |
|         2 | Маяковский  |
|         3 | Блок        |
|         4 | Есенин      |
+-----------+-------------+
4 rows in set (0.02 sec)

mysql> UPDATE author SET author='Тургенев' WHERE id_author=3;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM author;
+-----------+-------------+
| id_author | author      |
+-----------+-------------+
|         1 | Тютчев      |
|         2 | Маяковский  |
|         3 | Тургенев    |
|         4 | Есенин      |
+-----------+-------------+
4 rows in set (0.00 sec)

mysql>
```

Рис. 11.1. Выполнение запроса `UPDATE`

```
Командная строка - mysql -uroot library
+---------+-----------------------------+------------+
| id_addr | address                     | phone      |
+---------+-----------------------------+------------+
|       1 | ул. Ленина, 21/12           | 145-98-65  |
|       2 | ул. Маршала Жукова, 134/11  | 345-58-34  |
|       3 | ул. Зеленая, 34/22          | 239-48-67  |
+---------+-----------------------------+------------+
3 rows in set (0.00 sec)

mysql> UPDATE address SET address='ул. Мира, 32/15', phone='320-54-54'
    -> WHERE id_addr=2;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM address;
+---------+--------------------+------------+
| id_addr | address            | phone      |
+---------+--------------------+------------+
|       1 | ул. Ленина, 21/12  | 145-98-65  |
|       2 | ул. Мира, 32/15    | 320-54-54  |
|       3 | ул. Зеленая, 34/22 | 239-48-67  |
+---------+--------------------+------------+
3 rows in set (0.00 sec)

mysql>
```

Рис. 11.2. Обновление нескольких полей

Можно использовать и такой синтаксис:

```
UPDATE имя_таблицы SET имя_поля=имя_поля+1 WHERE условие;
```

Здесь в поле помещается его текущее значение, увеличенное на единицу.

```
UPDATE имя_таблицы SET имя_поля=имя_поля*2, имя_поля=имя_поля+1
  WHERE условие;
```

Такой запрос увеличивает значение поля *имя_поля* в два раза, а затем к получившемуся значению прибавляется единица.

Если предложение `WHERE` не определено, то изменяются все записи таблицы. Данный оператор возвращает количество обновленных строк (см. рис. 11.2).

```
Rows matched: 1 Changed: 1 Warnings: 0
```

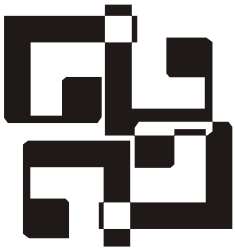
Если в результате обновления появится дублирующее значение, оператор `UPDATE` завершится ошибкой. Использование ключевого слова `IGNORE` приводит к тому, что ошибка не возникнет, но и записи обновлены не будут.

Можно ограничивать количество обновляемых записей:

```
UPDATE имя_таблицы SET имя_поля=значение WHERE условие LIMIT n;
```

Значение *n* в предложении `LIMIT` указывает, сколько записей необходимо обновить.

УРОК 12



Выборка данных (оператор *SELECT*)

Для получения записей из одной и более таблиц предназначен оператор `SELECT`. Его можно использовать для составления отчетов или просмотра результатов работы других операторов. В примерах предыдущих уроков этот оператор уже встречался, а теперь настало время ознакомиться с ним подробнее.

Выборка всех данных

Общий синтаксис оператора `SELECT` (мы его уже применяли):

```
SELECT * FROM имя_таблицы;
```

Символ `'*'` говорит о том, что выбираются все поля таблицы *имя_таблицы*. Данный синтаксис очень удобен, если нужно просмотреть всю таблицу. Так можно отследить ошибки, которые вы, возможно, допустили при вводе данных.

Выборка из определенных полей

Для выборки данных из определенных полей таблицы достаточно перечислить имена этих полей через запятую. Например, чтобы узнать название книги и год ее выпуска, введите запрос:

```
mysql> SELECT title, year_issue FROM book;
```

Значения полей возвращаются в том же порядке, в котором они хранятся в таблице (рис. 12.1). Поля можно перечислять в любом порядке и указывать одно поле любое количество раз.

```
Командная строка - mysql -uroot library
mysql> SELECT title, year_issue FROM book;
+-------------------------+-------------+
| title                   | year_issue  |
+-------------------------+-------------+
| Нос                     |        2003 |
| Сказка о золотой рыбке  |        2003 |
| Руслан и Людмила        |        2003 |
| Мартышка и очки         |        2003 |
| Евгений Онегин          |        2001 |
| Шинель                  |        2001 |
| Идиот                   |        1991 |
+-------------------------+-------------+
7 rows in set (0.00 sec)

mysql> _
```

Рис. 12.1. Выборка данных из определенных полей

Исключение дубликатов

Обычный запрос просто выводит все данные из указанных полей. Но многие данные могут повторяться при выводе. Например, может быть довольно много книг с одинаковым названием, но разных годов выпуска. Значит, при выборке по названию нам придется наблюдать множество повторяющихся значений. Ограничить вывод можно при помощи ключевого слова `DISTINCT`.

```
mysql> SELECT DISTINCT title FROM book;
```

Этот запрос будет выводить только уникальные записи (рис. 12.2).

```
Командная строка - mysql -uroot library
mysql> SELECT title FROM book;
+-------------------------+
| title                   |
+-------------------------+
| Нос                     |
| Сказка о золотой рыбке  |
| Руслан и Людмила        |
| Мартышка и очки         |
| Евгений Онегин          |
| Шинель                  |
| Идиот                   |
| Идиот                   |
| Евгений Онегин          |
+-------------------------+
9 rows in set (0.00 sec)

mysql> SELECT DISTINCT title FROM book;
+-------------------------+
| title                   |
+-------------------------+
| Нос                     |
| Сказка о золотой рыбке  |
| Руслан и Людмила        |
| Мартышка и очки         |
| Евгений Онегин          |
| Шинель                  |
| Идиот                   |
+-------------------------+
7 rows in set (0.00 sec)

mysql>
```

Рис. 12.2. Удаление дубликатов

Ограничение вывода

Для ограничения количества записей, выводимых запросом SELECT, используется предложение LIMIT, принимающее один или два аргумента. Эти аргументы должны быть целочисленными константами. Синтаксис:

SELECT *имя_поля* FROM *имя_таблицы* LIMIT *число*;

Здесь предложение LIMIT указано с одним аргументом: *число* — число выводимых записей. Например, чтобы выбрать названия только пяти первых книг, нужно ввести такой запрос (рис. 12.3):

```
mysql> SELECT title FROM book LIMIT 5;
```

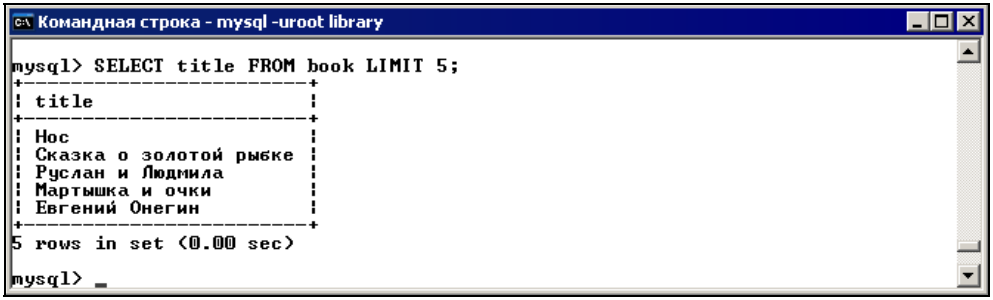


Рис. 12.3. Ограничение вывода

Также можно указать в запросе номер записи, с которой нужно начать выборку. Для этого следует в предложении LIMIT установить два аргумента:

SELECT *имя_поля* FROM *имя_таблицы* LIMIT *смещение*, *число*;

Параметры: смещение — номер записи, с которой начинается выборка, причем эта запись в выборку не включается; *число* — число выбираемых записей.

Допусти́м также формат записи LIMIT *число*,-1, позволяющий пропустить первые *число* записей, а затем произвести выборку всех оставшихся независимо от их количества. В принципе формат записи LIMIT *n* эквивалентен формату записи LIMIT 0,*n*.

Напишем запрос с двумя аргументами в предложении LIMIT. Следующий запрос выбирает после второй записи три названия книги:

```
mysql> SELECT title FROM book LIMIT 2,3;
```

Результат выборки представлен на рис. 12.4.

Если предложение LIMIT *n* используется с ключевыми словами ORDER BY, то MySQL закончит сортировку значений, как только найдет первые *n* строк, вместо того, чтобы сортировать всю таблицу.

```
Командная строка - mysql -uroot library

mysql> SELECT title FROM book;
+-------------------------+
| title                   |
+-------------------------+
| Нос                     |
| Сказка о золотой рыбке  |
| Руслан и Людмила        |
| Мартышка и очки         |
| Евгений Онегин          |
| Шинель                  |
| Идиот                   |
| Идиот                   |
| Евгений Онегин          |
+-------------------------+
9 rows in set (0.00 sec)

mysql> SELECT title FROM book LIMIT 2,3;
+-------------------+
| title             |
+-------------------+
| Руслан и Людмила  |
| Мартышка и очки   |
| Евгений Онегин    |
+-------------------+
3 rows in set (0.00 sec)

mysql> _
```

Рис. 12.4. Использование опции `LIMIT offset, count`

Выборка определенных записей

Имеется возможность выбирать из таблицы определенные строки, удовлетворяющие каким-либо условиям. Синтаксис:

```
SELECT ... WHERE
```

Например, если понадобится выбрать книги только определенного автора, то можно выполнить следующий запрос (рис. 12.5):

```
mysql> SELECT * FROM book WHERE id_author=1;
```

```
Командная строка - mysql library

mysql> SELECT * FROM book WHERE id_author=1;
+---------+-------------------------+------------+-----------+---------------+----
--------+
| id_book | title                   | year_issue | id_author | id_publisher | id_
section |
+---------+-------------------------+------------+-----------+---------------+----
--------+
|       2 | Сказка о золотой рыбке  |       2003 |         1 |             2 |
      3 |
|       3 | Руслан и Людмила        |       2003 |         1 |             2 |
      3 |
|       6 | Шинель                  |       2001 |         1 |             3 |
      1 |
+---------+-------------------------+------------+-----------+---------------+----
--------+
3 rows in set (0.00 sec)

mysql> _
```

Рис. 12.5. Выборка книг определенного автора

Примечание

При создании условия для выборки можно использовать все, что было описано в *уроке 10*. В предложении `WHERE` можно применять арифметические операции, операции сравнения и логические операции. Выражения можно группировать при помощи скобок.

При использовании логических операций нужно хорошо понимать разницу между логическим И и логическим ИЛИ. Например, если нужно выбрать книги, вышедшие в 2003 и 1995 году, то можно попробовать выполнить такой запрос:

```
mysql> SELECT title, year_issue FROM book WHERE year_issue=2003
  AND year_issue=1995;
```

Но этот запрос будет *неверным*. Здесь мы на самом деле пытаемся выбрать книгу, которая вышла и в 2003, и в 1995 году, но у одной книги не может быть два года выпуска. Правильный запрос должен выглядеть так (рис. 12.6):

```
mysql> SELECT title, year_issue FROM book WHERE year_issue=2003
  OR year_issue=1995;
```

Данный запрос выбирает названия и года издания книг, вышедших в 2003 или 1995 году.

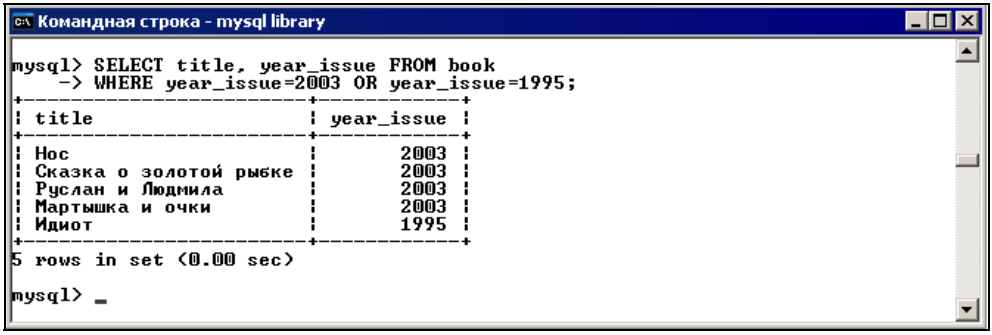


Рис. 12.6. Выборка с использованием оператора `OR`

Поэтому будьте очень внимательны при использовании в запросах логических операций.

Оператор *IN*

Последний запрос из предыдущего раздела можно записать и так (рис. 12.7):

```
mysql> SELECT title, year_issue FROM book WHERE year_issue IN(2003,1995);
```

```
Командная строка - mysql library

mysql> SELECT title, year_issue FROM book
    -> WHERE year_issue IN(2003,1995);
+-------------------------+------------+
| title                   | year_issue |
+-------------------------+------------+
| Нос                     |       2003 |
| Сказка о золотой рыбке  |       2003 |
| Руслан и Людмила        |       2003 |
| Мартышка и очки         |       2003 |
| Идиот                   |       1995 |
+-------------------------+------------+
5 rows in set (0.02 sec)

mysql>
```

Рис. 12.7. Использование оператора `IN`

Данная форма записи с оператором `IN` ("принадлежит") является просто краткой записью последовательности условий, перечисленных с помощью оператора `OR` (как в предыдущем примере). То есть два последних запроса эквивалентны.

Кроме этого, можно использовать оператор `NOT IN` ("не принадлежит"):

```
mysql> SELECT title, year_issue FROM book WHERE year_issue
  NOT IN(2003,1995);
```

В этом случае мы получим книги, которые выпущены в года, отличные от перечисленных в списке.

Оператор *BETWEEN* ... *AND* ...

С помощью оператора `BETWEEN ... AND ...` ("входит в диапазон от ... до ...") можно выбрать записи, в которых значение какого-либо поля находится в определенном диапазоне.

```
Командная строка - mysql library

mysql> SELECT title, year_issue FROM book
    -> WHERE year_issue BETWEEN 1990 and 2000;
+-------------------------+------------+
| title                   | year_issue |
+-------------------------+------------+
| Идиот                   |       1991 |
| Идиот                   |       1995 |
| Flash MX                |       1997 |
| Программирование на Perl |       1996 |
+-------------------------+------------+
4 rows in set (0.00 sec)

mysql> _
```

Рис. 12.8. Использование оператора `BETWEEN ... AND ..`

Например, следующий запрос позволяет выбрать книги, вышедшие между 1990 и 2000 годами (рис. 12.8):

```
mysql> SELECT title, year_issue FROM book WHERE year_issue
  BETWEEN 1990 AND 2000;
```

Также можно применять оператор NOT BETWEEN ("не входит в диапазон") (рис. 12.9).

```
Командная строка - mysql library
mysql> SELECT title, year_issue FROM book
    -> WHERE year_issue NOT BETWEEN 1990 and 2000 AND id_author=1;
+-------------------------+------------+
| title                   | year_issue |
+-------------------------+------------+
| Сказка о золотой рыбке  |       2003 |
| Руслан и Людмила        |       2003 |
| Шинель                  |       2001 |
+-------------------------+------------+
3 rows in set (0.00 sec)

mysql> _
```

Рис. 12.9. Использование оператора NOT BETWEEN

Выборка с упорядочением

В процессе работы с БД записи в таблицах модифицируются, удаляются и добавляются. Поэтому довольно часто необходим *упорядоченный вывод*. Если требуется отсортировать выводимые данные по какому-либо полю, используйте синтаксис:

SELECT *имя_поля1*, *имя_поля2*, ... FROM *имя_таблицы* ORDER BY *имя_поля*;

В отсортированных данных гораздо легче разобраться. В предложении ORDER BY вы можете указывать не только имя поля, по которому производится сортировка, но и порядковый номер данного поля (позицию в таблице).

Примечание

Нумерация позиций полей начинается с 1.

Что будет, если поля содержат значения NULL? Результаты сортировки в этом случае зависят от версии MySQL. В MySQL версии 4.0.2 значения NULL всегда располагаются в начале списка значений. В более ранних версиях они выводятся в начале при сортировке по возрастанию и в конце — при сортировке по убыванию.

По умолчанию записи будут отсортированы по возрастанию величин в указанном поле. Например, сделаем выборку книг, отсортировав записи по году выпуска:

```
mysql> SELECT * FROM book ORDER BY year_issue;
```

или

```
mysql> SELECT * FROM book ORDER BY 3;
```

```
Командная строка - mysql library
mysql> SELECT * FROM book ORDER BY year_issue;
+---------+-------------------------+------------+-----------+--------------+----
--------+
| id_book | title                   | year_issue | id_author | id_publisher | id_
section |
+---------+-------------------------+------------+-----------+--------------+----
--------+
|       9 | Евгений Онегин          |       1985 |         4 |            2 |
      1 |
|       7 | Идиот                   |       1991 |         5 |            3 |
      1 |
|       8 | Идиот                   |       1995 |         5 |            3 |
      1 |
|       5 | Евгений Онегин          |       2001 |         4 |            1 |
      1 |
|       6 | Шинель                  |       2001 |         1 |            3 |
      1 |
|       1 | Нос                     |       2003 |         3 |            2 |
      1 |
|       2 | Сказка о золотой рыбке  |       2003 |         1 |            2 |
      3 |
|       3 | Руслан и Людмила        |       2003 |         1 |            2 |
      3 |
|       4 | Мартышка и очки         |       2003 |         2 |            1 |
      2 |
+---------+-------------------------+------------+-----------+--------------+----
--------+
9 rows in set (0.01 sec)

mysql>
```

Рис. 12.10. Выборка книг с упорядочением по году выпуска

Данный запрос выводит все поля таблицы `book`, упорядочив (отсортировав) их по году выпуска (рис. 12.10).

Конечно же, сортировку можно вести и по текстовым полям (по алфавиту). Например (рис. 12.11):

```
mysql> SELECT title, year_issue FROM book ORDER BY title;
```

```
Командная строка - mysql library
mysql> SELECT title,year_issue FROM book ORDER BY title;
+-------------------------+------------+
| title                   | year_issue |
+-------------------------+------------+
| Евгений Онегин          |       2001 |
| Евгений Онегин          |       1985 |
| Идиот                   |       1991 |
| Идиот                   |       1995 |
| Мартышка и очки         |       2003 |
| Нос                     |       2003 |
| Руслан и Людмила        |       2003 |
| Сказка о золотой рыбке  |       2003 |
| Шинель                  |       2001 |
+-------------------------+------------+
9 rows in set (0.01 sec)

mysql> _
```

Рис. 12.11. Сортировка по названию книги

Можно указать в операторе `ORDER BY` несколько полей:

```
mysql> SELECT title, year_issue FROM book ORDER BY title, year_issue;
```

Здесь мы получили сортировку названий книг по возрастанию и сортировку по возрастанию года выпуска для каждого названия (например, для книги "Евгений Онегин" выведен сначала 1985 год, а затем 2001 год) (рис. 12.12).

```
Командная строка - mysql library

mysql> SELECT title,year_issue FROM book ORDER BY title, year_issue;
+-------------------------+------------+
| title                   | year_issue |
+-------------------------+------------+
| Евгений Онегин          |       1985 |
| Евгений Онегин          |       2001 |
| Идиот                   |       1991 |
| Идиот                   |       1995 |
| Мартышка и очки         |       2003 |
| Нос                     |       2003 |
| Руслан и Людмила        |       2003 |
| Сказка о золотой рыбке  |       2003 |
| Шинель                  |       2001 |
+-------------------------+------------+
9 rows in set (0.00 sec)

mysql>
```

Рис. 12.12. Сортировка по нескольким полям

При сортировке полей с символьными значениями не учитывается регистр. Это значит, что порядок расположения значений, совпадающих во всем, кроме регистра букв, будет неопределенным. Проводить сортировку с учетом регистра можно при помощи ключевого слова `BINARY`:

```
SELECT * FROM имя_таблицы ORDER BY BINARY имя_поля;
```

Для сортировки полей в обратном порядке (по убыванию) используйте ключевое слово `DESC`. Например:

```
mysql> SELECT title, year_issue FROM book ORDER BY title DESC;
```

Результат выполнения этого запроса представлен на рис. 12.13.

```
Командная строка - mysql library

mysql> SELECT title,year_issue FROM book ORDER BY title DESC;
+-------------------------+------------+
| title                   | year_issue |
+-------------------------+------------+
| Шинель                  |       2001 |
| Сказка о золотой рыбке  |       2003 |
| Руслан и Людмила        |       2003 |
| Нос                     |       2003 |
| Мартышка и очки         |       2003 |
| Идиот                   |       1991 |
| Идиот                   |       1995 |
| Евгений Онегин          |       2001 |
| Евгений Онегин          |       1985 |
+-------------------------+------------+
9 rows in set (0.00 sec)

mysql> _
```

Рис. 12.13. Сортировка по убыванию значений

Можно сортировать несколько столбцов, при этом каждый из них будет отсортирован по возрастанию или убыванию независимо от других. Например:

```
mysql> SELECT title, year_issue FROM book ORDER BY title
  DESC, year_issue ASC;
```

Здесь к полю year_issue добавлено ключевое слово ASC (сортировка поля по возрастанию значений, по умолчанию).

Можно выбрать самую новую из имеющихся в библиотеке книг:

```
mysql> SELECT title, year_issue FROM book ORDER BY year_issue
  DESC LIMIT 1;
```

В случае, если книг окажется несколько, будет выбрана запись только одной из них (первая встреченная).

Действие ключевого слова DESC распространяется только на поле, расположенное непосредственно перед ним. Значения остальных полей сортируются по возрастанию.

Группировка

Предложение GROUP BY служит для перекомпоновки записей таблицы по группам, в каждой из которых все записи имеют одинаковые значения в поле, указанном в предложении GROUP BY.

Пример использования предложения GROUP BY для вывода только уникальных названий из поля title таблицы book:

```
mysql> SELECT title FROM book GROUP BY title;
```

Пример использования предложения GROUP BY для вывода только уникальных названий и соответствующих им годов из полей title и year_issue таблицы book:

```
mysql> SELECT title, year_issue FROM book GROUP BY title;
```

Пример использования предложения GROUP BY для вывода только уникальных годов и названий книг, вышедших в эти года, из полей title и year_issue таблицы book:

```
mysql> SELECT title, year_issue FROM book GROUP BY year_issue;
```

Если есть несколько книг, вышедших в каком-либо году, то выводится первое найденное название, соответствующее этому году.

Результаты выполнения этих запросов показаны на рис. 12.14.

```
Командная строка - mysql library
mysql> SELECT title FROM book GROUP BY title;
+-------------------------+
| title                   |
+-------------------------+
| Евгений Онегин          |
| Идиот                   |
| Мартышка и очки         |
| Нос                     |
| Руслан и Людмила        |
| Сказка о золотой рыбке  |
| Шинель                  |
+-------------------------+
7 rows in set (0.00 sec)

mysql> SELECT title, year_issue FROM book GROUP BY title;
+-------------------------+------------+
| title                   | year_issue |
+-------------------------+------------+
| Евгений Онегин          |       2001 |
| Идиот                   |       1991 |
| Мартышка и очки         |       2003 |
| Нос                     |       2003 |
| Руслан и Людмила        |       2003 |
| Сказка о золотой рыбке  |       2003 |
| Шинель                  |       2001 |
+-------------------------+------------+
7 rows in set (0.00 sec)

mysql> SELECT title, year_issue FROM book GROUP BY year_issue;
+----------------+------------+
| title          | year_issue |
+----------------+------------+
| Евгений Онегин |       1985 |
| Идиот          |       1991 |
| Идиот          |       1995 |
| Евгений Онегин |       2001 |
| Нос            |       2003 |
+----------------+------------+
5 rows in set (0.01 sec)

mysql> _
```

Рис. 12.14. Использование предложения `GROUP BY`

Для исключения некоторых групп из вывода используйте выражение `HAVING` *условие*. Предложение `HAVING` играет такую же роль для групп, как выражение `WHERE` для строк. Это предложение включается в запрос лишь при наличии оператора `GROUP BY`. Оно будет обрабатываться последним, непосредственно перед выводом.

Использование функций и операций при выборке данных

В языке SQL операторы `SELECT` и `WHERE` могут содержать различные функции и операции.

Примечание

Между именем функции и круглыми скобками, в которые заключены ее параметры, не должно быть пробелов.

В табл. 12.1 описаны не все функции и операции, присутствующие в языке SQL, а только наиболее полезные (по моему мнению) из них.

Таблица 12.1. *Основные функции и операции*

Функция/операция	**Описание**
`+, -, *, /`	Арифметические операции сложения, вычитания, умножения и деления
`%`	Остаток от деления
`()`	Определяют порядок вычислений
`BETWEEN(A,B,C)`	Аналогична выражению `(A>=B) AND (A<=C)`
`BIT_COUNT()`	Определяет количество бит
`ELT(N,a,b,c)`	Возвращает значение `a`, если `N==1`, значение `b`, если `N==2`, и т. д. (`a`, `b`, `c` — строки). Например, функция `ELT(2,'яблоко','груша','апельсин')` возвратит значение `'груша'`
`FIELD(S,a,b,c)`	Возвращает значение `a`, если `S==a`, значение `b`, если `S==b`, и т. д. (`a`, `b`, `c` — строки). Например, функция `FIELD('яблоко', 'яблоко', 'груша', 'апельсин')` возвратит значение `'яблоко'`
`IF(A,B,C)`	Если выражение `A` истинно (не равно 0 или `NULL`), то возвращается значение `B`, иначе возвращается значение `C`
`IFNULL(A,B)`	Если выражение `A` не равно `NULL`, возвращается значение `A`, иначе возвращается значение `B`
`ISNULL(A)`	Возвращает значение 1, если `A==NULL`
`NOT, AND, OR`	Возвращают значение `TRUE` (1) или `FALSE` (0)
`SIGN()`	Определяет знак аргумента, возвращает значение –1, 0 или 1
`=, <>, <=, <, >=, >`	Возвращают значение `TRUE` (1) или `FALSE` (0)
`LIKE` *выражение*	Возвращает значение `TRUE` (1) или `FALSE` (0)
`NOT LIKE` *выражение*	Возвращает значение `TRUE` (1) или `FALSE` (0)
`REGEXP` *выражение*	Возвращает значение `TRUE` (1) или `FALSE` (0)
`NOT REGEXP` *выражение*	Возвращает значение `TRUE` (1) или `FALSE` (0)

Некоторые из представленных в табл. 12.1 выражений упоминались в этой книге ранее.

Кроме того, в языке SQL есть ряд математических функций (табл. 12.2).

Таблица 12.2. *Математические функции*

Функция	Описание
ABS(X)	Возвращает абсолютное значение (модуль) числа X
CEILING(X)	Возвращает ближайшее целое число больше X
FLOOR(X)	Возвращает ближайшее целое число меньше X
ROUND(X)	Округляет значение X до ближайшего целого
EXP(X)	Возвращает значение e^X (e — основание натурального логарифма)
LOG()	Возвращает значение натурального логарифма
LOG10()	Возвращает значение логарифма по основанию 10
MOD()	Возвращает остаток от деления
POW(X,Y)	Возвращает значение X^Y
SQRT(X)	Возвращает квадратный корень из X
RAND()	Возвращает случайную величину в диапазоне от 0 до 1
PI()	Возвращает значение π
SIN(X)	Возвращает синус X (значение X задается в радианах)
COS(X)	Возвращает косинус X
TAN(X)	Возвращает тангенс X
COT(X)	Возвращает котангенс X
RADIANS(X)	Возвращает значение X, преобразованное из градусов в радианы
DEGREES(X)	Возвращает значение X, преобразованное из радианов в градусы

При выборке данных можно использовать ряд *строковых* функций (табл. 12.3). Если строковая функция содержит в качестве аргумента строку с двоичными данными, то и результирующая строка также будет строкой с двоичными данными. При этом число, конвертированное в строку, воспринимается как строка с двоичными данными (это имеет значение только при выполнении операций сравнения).

В табл. 12.4 описаны прочие полезные функции.

Таблица 12.3. *Строковые функции*

Функция	Описание
BIN(N)	Возвращает строковое представление двоичного значения числа N, где N — целое число большого размера (BIGINT). Если N равно NULL, возвращается значение NULL

Таблица 12.3 *(окончание)*

Функция	Описание
OCT(N)	Возвращает строковое представление восьмеричного значения числа N, где N — целое число большого размера. Если N равно NULL, возвращается значение NULL
HEX(N)	Если N — число, то возвращается строковое представление шестнадцатеричного числа N, где N — целое число большого размера (BIGINT). Если N — строка, то возвращается шестнадцатеричная строка N, где каждый символ в N конвертируется в 2 шестнадцатеричных числа
CONCAT()	Выполняет объединение строк
INTERVAL(A,a,b,c,d)	Возвращает значение 1, если A==a, значение 2, если A==b, и т. д. (A, a, b, c, d — строки). Если совпадений нет, возвращает значение 0
INSERT(str,pos,len,newstr)	Возвращает строку str, в которой подстрока длиной len, начинающаяся с позиции pos, замещена подстрокой newstr
LCASE(S)	Приводит буквы строки S к нижнему регистру
UCASE(S)	Приводит буквы строки S к верхнему регистру
LEFT(str,len)	Возвращает крайние слева len символов из строки str
RIGHT(str,len)	Возвращает крайние справа len символов из строки str
MID(str,pos,len)	Возвращает подстроку длиной len символов из строки str, начиная от позиции pos
LENGTH(str)	Возвращает длину строки str
TRIM(str)	Возвращает строку str без конечных и начальных пробелов
LTRIM(str)	Возвращает строку str без начальных пробелов
RTRIM(str)	Возвращает строку str без конечных пробелов
LOCATE(A,B)	Возвращает позицию подстроки B в строке A
LOCATE(A,B,C)	Возвращает позицию подстроки B в строке A, начиная с позиции C
SUBSTRING(str,pos,len)	Возвращает подстроку длиной len символов из строки str, начиная с позиции pos

Таблица 12.4. *Разные функции*

Функция	Описание
CURTIME()	Возвращает текущее время в формате *HH:MM:SS* или *HHMMSS*. Формат зависит от того, в каком контексте используется функция: в числовом — например, при выполнении запроса SELECT CURTIME()+0; получим 165651 (число); в строковом — например, при выполнении запроса SELECT CURTIME(); получим '16:56:51' (строка)
CURDATE()	Возвращает текущую дату в формате *YYYY-MM-DD* или *YYYYMMDD*
DATABASE()	Возвращает имя текущей базы данных
VERSION()	Возвращает строку с номером версии MySQL
USER()	Возвращает регистрационное имя текущего пользователя, под которым пользователь подключился к БД (login)
PASSWORD(str)	Создает (шифрует) строку "пароля" из простого текста, заданного в аргументе str

Групповые функции

Функции, представленные в табл. 12.5, можно использовать в предложении GROUP.

Таблица 12.5. *Групповые функции*

Функция	Описание
AVG()	Возвращает среднее значение для группы
SUM()	Возвращает сумму элементов группы
COUNT()	Возвращает число элементов группы
MIN()	Возвращает минимальный элемент группы
MAX()	Возвращает максимальный элемент группы

Вызов групповых функций для SQL-запросов, не содержащих предложение GROUP BY, эквивалентен выполнению этих функций над всем набором возвращаемых данных. Эти функции нельзя использовать в выражениях, но их аргументы могут быть выражениями, например: SUM(value/10).

Примеры использования некоторых функций

Функция COUNT(*выражение*) возвращает количество значений (отличных от NULL) в строках, полученных при использовании оператора SELECT (рис. 12.15).

```
mysql> SELECT COUNT(id_book) FROM book;
```

```
C:\WINDOWS\system32\cmd.exe - mysql library

mysql> SELECT * FROM book;
+---------+-----------------------+------------+-----------+-------------+------------+
| id_book | title                 | year_issue | id_author | id_publisher | id_section |
+---------+-----------------------+------------+-----------+-------------+------------+
|       1 | Нос                   |       2003 |         3 |           2 |          1 |
|       2 | Сказка о золотой рыбке |       2003 |         1 |           2 |          3 |
|       3 | Руслан и Людмила      |       2003 |         1 |           2 |          3 |
|       4 | Мартышка и очки       |       2003 |         2 |           1 |          2 |
|       5 | Евгений Онегин        |       2001 |         4 |           1 |          1 |
|       6 | Шинель                |       2001 |         1 |           3 |          1 |
|       7 | Идиот                 |       1991 |         5 |           3 |          1 |
|       8 | Идиот                 |       1995 |         5 |           3 |          1 |
|       9 | Евгений Онегин        |       1985 |         4 |           2 |          1 |
+---------+-----------------------+------------+-----------+-------------+------------+
9 rows in set (0.00 sec)

mysql> SELECT COUNT(id_book) FROM book;
+----------------+
| COUNT(id_book) |
+----------------+
|              9 |
+----------------+
1 row in set (0.00 sec)

mysql>
```

Рис. 12.15. Использование функции `COUNT()`

Функция `COUNT(*)` возвращает количество извлеченных строк, содержащих значения `NULL`. Она оптимизирована для быстрого возвращения результата, при этом запрос выборки должен относиться к одной таблице и не содержать выражение `WHERE`.

```
mysql> SELECT COUNT(*) FROM book;
```

При подсчете количества записей можно исключить повторяющиеся значения в поле, для которого применяется функция `COUNT()`.

Например, узнаем количество записей в таблице `book`, исключив записи книг с одинаковыми названиями (рис. 12.16):

```
mysql> SELECT COUNT(DISTINCT title) FROM book;
```

```
C:\WINDOWS\system32\cmd.exe - mysql library

mysql> SELECT * FROM book;
+---------+-----------------------+------------+-----------+-------------+------------+
| id_book | title                 | year_issue | id_author | id_publisher | id_section |
+---------+-----------------------+------------+-----------+-------------+------------+
|       1 | Нос                   |       2003 |         3 |           2 |          1 |
|       2 | Сказка о золотой рыбке |       2003 |         1 |           2 |          3 |
|       3 | Руслан и Людмила      |       2003 |         1 |           2 |          3 |
|       4 | Мартышка и очки       |       2003 |         2 |           1 |          2 |
|       5 | Евгений Онегин        |       2001 |         4 |           1 |          1 |
|       6 | Шинель                |       2001 |         1 |           3 |          1 |
|       7 | Идиот                 |       1991 |         5 |           3 |          1 |
|       8 | Идиот                 |       1995 |         5 |           3 |          1 |
|       9 | Евгений Онегин        |       1985 |         4 |           2 |          1 |
+---------+-----------------------+------------+-----------+-------------+------------+
9 rows in set (0.02 sec)

mysql> SELECT COUNT(DISTINCT title) FROM book;
+-------------------------+
| COUNT(DISTINCT title) |
+-------------------------+
|                       7 |
+-------------------------+
1 row in set (0.00 sec)

mysql>
```

Рис. 12.16. Использование синтаксиса `COUNT(DISTINCT` *выражение*`)`

А теперь подсчитаем количество книг, сгруппировав их по названиям (рис. 12.17):

```
mysql> SELECT title, COUNT(id_book) FROM book GROUP BY title;
```

```
C:\WINDOWS\system32\cmd.exe - mysql library

mysql> SELECT title,COUNT(id_book) FROM book GROUP BY title;
+-----------------------+-----------------+
| title                 | COUNT(id_book)  |
+-----------------------+-----------------+
| Евгений Онегин        |               2 |
| Идиот                 |               2 |
| Мартышка и очки       |               1 |
| Нос                   |               1 |
| Руслан и Людмила      |               1 |
| Сказка о золотой рыбке |              1 |
| Шинель                |               1 |
+-----------------------+-----------------+
7 rows in set (0.00 sec)

mysql> _
```

Рис. 12.17. Использование функции `COUNT()` с предложением `GROUP BY`

После выполнения данного запроса мы увидим количество книг с одинаковыми названиями, т. к. произвели группировку по полю `title`.

Функции `MIN()` и `MAX()` возвращают соответственно минимальное и максимальное значения в поле (рис. 12.18). Эти функции могут принимать в качестве аргумента как числовые, так и строковые величины.

```
C:\WINDOWS\system32\cmd.exe - mysql library

mysql> SELECT MAX(title) FROM book;
+-------------+
| MAX(title)  |
+-------------+
| Шинель      |
+-------------+
1 row in set (0.00 sec)

mysql> SELECT MIN(title) FROM book;
+----------------+
| MIN(title)     |
+----------------+
| Евгений Онегин |
+----------------+
1 row in set (0.00 sec)

mysql> SELECT MIN(year_issue) FROM book;
+------------------+
| MIN(year_issue)  |
+------------------+
|             1985 |
+------------------+
1 row in set (0.00 sec)

mysql> SELECT MAX(year_issue) FROM book;
+------------------+
| MAX(year_issue)  |
+------------------+
|             2003 |
+------------------+
1 row in set (0.00 sec)

mysql> _
```

Рис. 12.18. Использование функций `MIN()` и `MAX()`

Функция `CONCAT(`*`строка1`*`,`*`строка2`*`,...)` выполняет конкатенацию ("склеивание" или сложение) строк и возвращает результат (рис. 12.19). Если одним из

аргументов функции будет число, то оно автоматически конвертируется в эквивалентную строковую форму.

```
C:\WINDOWS\system32\cmd.exe - mysql library
mysql> select CONCAT("light","green");
+---------------------------+
| CONCAT("light","green") |
+---------------------------+
| lightgreen                |
+---------------------------+
1 row in set (0.03 sec)

mysql> select CONCAT("light","green",1000);
+-------------------------------+
| CONCAT("light","green",1000) |
+-------------------------------+
| lightgreen1000                |
+-------------------------------+
1 row in set (0.00 sec)

mysql> select CONCAT("light","green",NULL);
+-------------------------------+
| CONCAT("light","green",NULL) |
+-------------------------------+
| NULL                          |
+-------------------------------+
1 row in set (0.00 sec)

mysql> _
```

Рис. 12.19. Работа функции `CONCAT()`

В случае, когда один из аргументов имеет значение `NULL`, результатом выполнения функции будет значение `NULL`.

Функция `CONCAT_WS(`*разделитель*`,`*строка1*`,`*строка2*`,...)` аналогична функции `CONCAT()`, но не "склеивает" строки, а объединяет их посредством разделителя, указанного в качестве первого аргумента.

Если требуется получить случайное значение, то можно воспользоваться функцией `RAND()`. Эта функция применяется в двух вариантах — с аргументом и без. Если аргумента нет, то функция возвращает значение, находящееся в промежутке между нулем и единицей, а если аргумент указан, то он используется как начальное значение возвращаемой величины. Например, выполнив запрос

```
mysql> SELECT RAND();
```

получим значение 0,031077887891864.

Выполнив запрос

```
mysql> SELECT RAND();
```

получим значение 0,004335573971584.

Выполнив запрос

```
mysql> SELECT RAND(4);
```

получим значение 0,40613597483014.

Выполнив запрос

```
mysql> SELECT RAND(4);
```

получим значение 0,40613597483014.

Эту функцию также можно применять для отображения случайной выборки при использовании оператора `SELECT` с предложением `ORDER BY`. Например, для получения случайного значения из поля `title` таблицы `book` можно выполнить запрос (рис. 12.20):

```
mysql> SELECT title FROM book ORDER BY RAND() LIMIT 1;
```

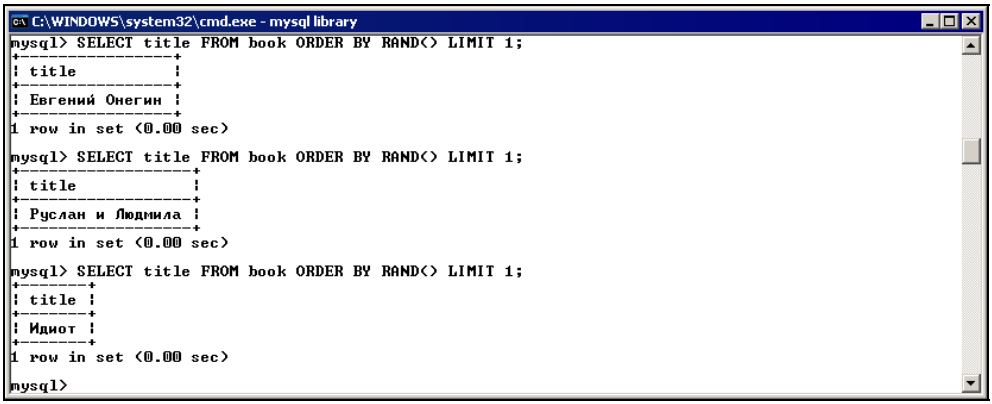


Рис. 12.20. Использование функции `RAND`

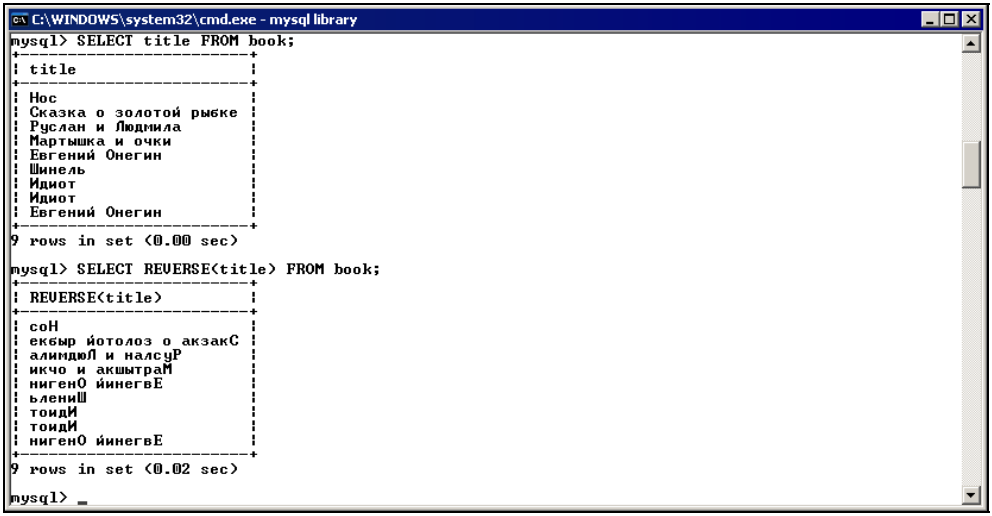


Рис. 12.21. Пример работы функции `REVERSE()`

Если вы используете функцию `RAND()` в предложении `WHERE`, то она будет вычисляться каждый раз заново при выполнении выражения `WHERE`.

Функция `REVERSE(`*строка*`)` возвращает "перевернутую" строку (рис. 12.21).

Функция `REPEAT(`*строка*`,`*число*`)` возвращает символьную строку *строка*, повторенную столько раз, сколько задано параметром *число*. Например, запрос

```
mysql> SELECT REPEAT('Привет', 3);
```

вернет строку `'ПриветПриветПривет'`.

Объединение данных из нескольких таблиц

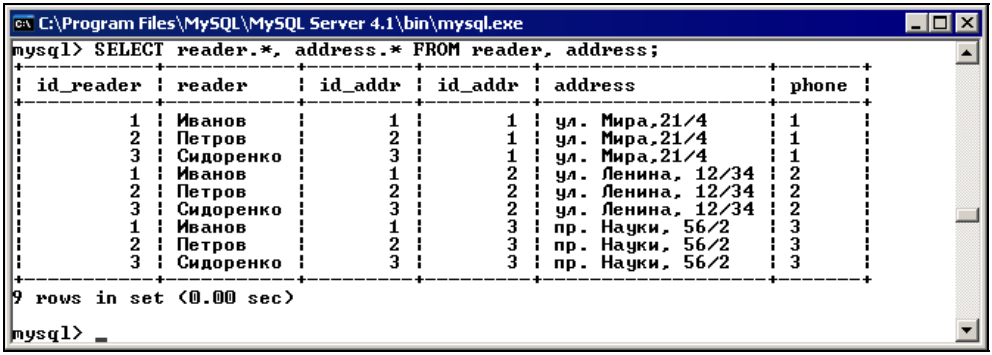
Очень часто бывает необходимо сделать выборку данных из нескольких таблиц сразу. Это более сложный запрос, в котором нужно четко определить, как следует связать (объединить) таблицы, чтобы получить желаемый результат. Например, если мы хотим получить в результате выборки название книги и ее автора, то необходимо задействовать таблицы `book` и `author`. Для указания выбираемого поля нужно использовать синтаксис *имя_таблицы*.*имя_поля* во избежание неоднозначности, которая может возникнуть, если в объединяемых таблицах есть поля с одинаковыми именами (когда мы выбирали данные из одной таблицы, неоднозначность исключалась).

Для полного объединения таблиц `table1` и `table2` можно выполнить запрос:

```
mysql> SELECT table1.*, table2.* FROM table1, table2;
```

Например, произведем выборку всех полей из таблиц `reader` и `address` (рис. 12.22):

```
mysql> SELECT reader.*, address.* FROM reader, address;
```



```
C:\Program Files\MySQL\MySQL Server 4.1\bin\mysql.exe
mysql> SELECT reader.*, address.* FROM reader, address;
+-----------+-----------+---------+---------+-------------------+-------+
| id_reader | reader    | id_addr | id_addr | address           | phone |
+-----------+-----------+---------+---------+-------------------+-------+
|         1 | Иванов    |       1 |       1 | ул. Мира,21/4     | 1     |
|         2 | Петров    |       2 |       1 | ул. Мира,21/4     | 1     |
|         3 | Сидоренко |       3 |       1 | ул. Мира,21/4     | 1     |
|         1 | Иванов    |       1 |       2 | ул. Ленина, 12/34 | 2     |
|         2 | Петров    |       2 |       2 | ул. Ленина, 12/34 | 2     |
|         3 | Сидоренко |       3 |       2 | ул. Ленина, 12/34 | 2     |
|         1 | Иванов    |       1 |       3 | пр. Науки, 56/2   | 3     |
|         2 | Петров    |       2 |       3 | пр. Науки, 56/2   | 3     |
|         3 | Сидоренко |       3 |       3 | пр. Науки, 56/2   | 3     |
+-----------+-----------+---------+---------+-------------------+-------+
9 rows in set (0.00 sec)

mysql> _
```

Рис. 12.22. Полное объединение

Каждая строка первой таблицы объединяется с каждой строкой второй таблицы. Такое объединение называют также *перекрестным*. На рис. 12.22 вы видите все возможные комбинации строк двух таблиц. Мы получаем довольно большое количество записей при выводе, т. к. итог это произведение количеств строк всех таблиц. В наших таблицах было всего по 3 записи. Если каждая таблица будет содержать по 100 записей, то в результате такой выборки получится $100 \times 100 = 10\,000$ строк. В этом случае, чтобы уменьшить итоговую выборку, нужно указать с помощью предложения `WHERE`, *как* объединяемые таблицы связаны друг с другом (рис. 12.23). В итоге мы получим следующий запрос:

```
mysql> SELECT reader.*, address.* FROM reader, address WHERE
  reader.id_addr=address.id_addr;
```

```
C:\Program Files\MySQL\MySQL Server 4.1\bin\mysql.exe

mysql> SELECT reader.*, address.* FROM reader, address
    -> WHERE reader.id_addr=address.id_addr;
+-----------+-----------+---------+---------+-------------------+-------+
| id_reader | reader    | id_addr | id_addr | address           | phone |
+-----------+-----------+---------+---------+-------------------+-------+
|         1 | Иванов    |       1 |       1 | ул. Мира,21/4     | 1     |
|         2 | Петров    |       2 |       2 | ул. Ленина, 12/34 | 2     |
|         3 | Сидоренко |       3 |       3 | пр. Науки, 56/2   | 3     |
+-----------+-----------+---------+---------+-------------------+-------+
3 rows in set (0.00 sec)

mysql>
```

Рис. 12.23. Использование предложения `WHERE`

При задании условий отбора можно указать только определенные поля. Например, вывести названия книг и фамилии их авторов можно с помощью следующего запроса:

```
mysql> SELECT book.title, author.author FROM book, author WHERE
  book.id_author=author.id_author;
```

После выполнения данного запроса мы увидим названия книг и фамилии их авторов (рис. 12.24).

Примечание

Если, например, в таблице `book` имеется запись о книге с номером автора (`id_author`), который отсутствует в таблице `author` (т. е. в таблицах не соблюдена ссылочная целостность), то данная книга не отобразится в результате выборки.

Примечание

В общем случае для данного запроса не обязательно указывать перед именем выбираемого поля имя таблицы, которой оно принадлежит (рис. 12.25). Делай-

те это, только если может возникнуть неоднозначность, т. е. если в таблицах присутствуют поля с одинаковыми именами.

```
C:\WINDOWS\system32\cmd.exe - mysql library
mysql> SELECT book.title,author.author FROM book,author WHERE book.id_author=author.id_author;
+-------------------------+--------------+
| title                   | author       |
+-------------------------+--------------+
| Сказка о золотой рыбке  | Пушкин       |
| Руслан и Людмила        | Пушкин       |
| Евгений Онегин          | Пушкин       |
| Евгений Онегин          | Пушкин       |
| Мартышка и очки         | Крылов       |
| Нос                     | Гоголь       |
| Шинель                  | Гоголь       |
| Идиот                   | Достоевский  |
| Идиот                   | Достоевский  |
+-------------------------+--------------+
9 rows in set (0.00 sec)

mysql> _
```

Рис. 12.24. Выборка из таблиц `book` и `author`

```
C:\WINDOWS\system32\cmd.exe - mysql library
mysql> SELECT title,author FROM book,author WHERE book.id_author=author.id_author;
+-------------------------+--------------+
| title                   | author       |
+-------------------------+--------------+
| Сказка о золотой рыбке  | Пушкин       |
| Руслан и Людмила        | Пушкин       |
| Евгений Онегин          | Пушкин       |
| Евгений Онегин          | Пушкин       |
| Мартышка и очки         | Крылов       |
| Нос                     | Гоголь       |
| Шинель                  | Гоголь       |
| Идиот                   | Достоевский  |
| Идиот                   | Достоевский  |
+-------------------------+--------------+
9 rows in set (0.00 sec)

mysql> _
```

Рис. 12.25. Выборка из таблиц `book` и `author` без указания имен таблиц

```
C:\WINDOWS\system32\cmd.exe - mysql library

mysql> SELECT title,reader,address,get_date,exp_date
    -> FROM book,reader,abonement,address,unit
    -> WHERE book.id_book=unit.id_book AND unit.id_unit=abonement.id_unit AND
    -> reader.id_reader=abonement.id_reader AND reader.id_addr=address.id_addr;
+-------------------------+----------+-----------------+-----------+-----------+
| title                   | reader   | address         | get_date  | exp_date  |
+-------------------------+----------+-----------------+-----------+-----------+
| Сказка о золотой рыбке  | Иванов   | ул. Ленина, 21/12 | 2005-01-22 | 2005-02-15 |
| Нос                     | Иващенко | ул. Мира, 32/15 | 2005-03-12 | 2005-03-27 |
| Евгений Онегин          | Зиновьева | пр. Мира, 23/93 | 2005-04-05 | 2005-04-29 |
+-------------------------+----------+-----------------+-----------+-----------+
3 rows in set (0.00 sec)

mysql>
```

Рис. 12.26. Выборка из пяти таблиц

Попробуем сделать выборку с большим количеством объединяемых таблиц (рис. 12.26). Допустим, нам нужна информация о книгах, которые находятся на руках у читателей. Выберем в БД читателя его адрес, название книги, которую он взял, а также дату выдачи и дату возврата:

```
mysql> SELECT title, reader, address, get_date, exp_date FROM book,
  reader, abonement, address, unit WHERE book.id_book=unit.id_book
```

```
  AND unit.id_unit=abonement.id_unit
  AND reader.id_reader=abonement.id_reader
  AND reader.id_addr=address.id_addr;
```

Как вы помните, условия, описанные в предложении WHERE, разделяются логическими операторами OR или AND. В данном случае используется оператор AND, т. к. нам необходимо выполнение *всех* описанных условий сразу.

В предложении FROM указана таблица unit, хотя из нее непосредственно никакие поля не выбираются. Вы должны перечислить все таблицы, которые каким-то образом участвуют в запросе. В данном случае таблица unit использовалась как промежуточная таблица для связки таблицы abonement и таблицы book, благодаря этому мы смогли по номеру экземпляра, хранящегося в таблице abonement, определить название книги, взяв его из таблицы book.

Использование других объединений (*JOIN*)

До настоящего времени для связки нескольких таблиц мы использовали *объединение по равенству* (equi-join). Нужные таблицы мы указывали в предложении FROM, объединив их с помощью запятой (,).

Существуют и другие типы объединений. Объединения JOIN и CROSS JOIN эквивалентны оператору объединения ','.

```
mysql> SELECT book.title, author.author FROM book JOIN author WHERE
  book.id_author=author.id_author;
mysql> SELECT book.title, author.author FROM book CROSS JOIN author WHERE
  book.id_author=author.id_author;
```

После выполнения приведенных запросов мы получим такой же результат, как на рис. 12.25.

Объединение STRAIGHT_JOIN работает так же, как уже описанные объединения, но с одним исключением. Здесь таблицы объединяются именно в том порядке, в каком они перечислены в предложении FROM. Ключевое слово STRAIGHT_JOIN можно указать в двух местах оператора SELECT:

```
SELECT STRAIGHT_JOIN ... FROM табл1, табл2,...
SELECT ... FROM табл1 STARIGHT_JOIN табл2 STRAIGHT_JOIN табл3 ...
```

Итак, рассмотренные объединения позволяют выбрать строки, значения которых полностью совпадают. Но есть и другие объединения. Правое и левое объединения позволяют также включить в результат выборки строки одной из таблиц, не имеющие соответствий в другой таблице. То есть при левом объединении двух таблиц будут выбраны строки, значения которых совпада-

ют в обеих таблицах и, кроме этого, будут выбраны строки левой таблицы, значения в которых не совпали со значениями в правой таблице. Таким образом, левое объединение выбирает все строки из левой таблицы вне зависимости от того, совпадают значения в таблицах или нет. Правое объединение работает аналогично, только таблицы меняются ролями.

Рассмотрим работу объединений на примере двух произвольных таблиц с именами `table1` и `table2` (рис. 12.27).

```
C:\WINDOWS\system32\cmd.exe - mysql test
mysql> SELECT * FROM table1;
+----+------+
| id | txt  |
+----+------+
|  1 | One  |
|  2 | Two  |
+----+------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM table2;
+----+------+
| id | txt  |
+----+------+
|  1 | a    |
|  2 | b    |
|  3 | c    |
+----+------+
3 rows in set (0.01 sec)
```

Рис. 12.27. Таблицы `table1` и `table2`

Если сделать полное объединение (рис. 12.28), то мы получим только две записи, т. к. в поле `id` совпадают два значения:

```
mysql> SELECT table1.*, table2.* FROM table1, table2 WHERE
  table1.id=table2.id;
```

```
C:\WINDOWS\system32\cmd.exe - mysql test
mysql> SELECT table1.*, table2.* FROM table1, table2
    -> WHERE table1.id=table2.id;
+----+------+----+------+
| id | txt  | id | txt  |
+----+------+----+------+
|  1 | One  |  1 | a    |
|  2 | Two  |  2 | b    |
+----+------+----+------+
2 rows in set (0.06 sec)

mysql> _
```

Рис. 12.28. Полное объединение

Теперь попробуем сделать левое объединение этих таблиц. Мы используем объединение `LEFT JOIN`, а условие на соответствие значений зададим при помощи предложения `ON` (вместо `WHERE`).

Примечание

Рекомендуется использовать объединение `LEFT JOIN` вместо `RIGHT JOIN` для сохранения переносимости кода между различными базами данных.

Здесь мы поменяем местами таблицы для удобства выборки (чтобы не использовать объединение `RIGHT JOIN`):

```
mysql> SELECT table2.*, table1.* FROM table2 LEFT JOIN table1
  ON table2.id=table1.id;
```

```
C:\WINDOWS\system32\cmd.exe - mysql test
mysql> SELECT table2.*, table1.* FROM table2 LEFT JOIN table1
    -> ON table2.id=table1.id;
+----+------+------+------+
| id | txt  | id   | txt  |
+----+------+------+------+
|  1 | a    |    1 | One  |
|  2 | b    |    2 | Two  |
|  3 | c    | NULL | NULL |
+----+------+------+------+
3 rows in set (0.00 sec)

mysql>
```

Рис. 12.29. Использование объединения `LEFT JOIN`

В результате (рис. 12.29) видим, что появилась третья строка из таблицы `table2`, которая не имеет совпадений в таблице `table1`. Поля для правой таблицы устанавливаются в значение `NULL`.

С помощью левого объединения можно отслеживать те записи в левой таблице, которые не находят соответствия в правой. Для этого в запрос нужно добавить предложение `WHERE`:

```
mysql> SELECT table2.* FROM table2 LEFT JOIN table1
  ON table2.id=table1.id WHERE table1.id IS NULL;
```

В результате мы видим одну запись из таблицы `table2` (рис. 12.30).

```
C:\WINDOWS\system32\cmd.exe - mysql test
mysql> SELECT table2.* FROM table2 LEFT JOIN table1
    -> ON table2.id=table1.id
    -> WHERE table1.id IS NULL;
+----+------+
| id | txt  |
+----+------+
|  3 | c    |
+----+------+
1 row in set (1.40 sec)

mysql>
```

Рис. 12.30. Вывод записей с несовпадающими значениями

Предложение `ON` позволяет задавать соответствие между полями, имеющими как одинаковые, так и разные имена. Но есть еще один способ задания соответствия между полями таблиц — с помощью предложения `USING()`. Оно аналогично предложению `ON`, но имена полей, по которым производится объединение, здесь должны полностью совпадать. Поскольку в нашем примере связываемые поля таблиц имеют одинаковые имена, запрос с левым объединением можно написать так (рис. 12.31):

```
mysql> SELECT table2.*, table1.* FROM table2 LEFT JOIN table1 USING(id);
```

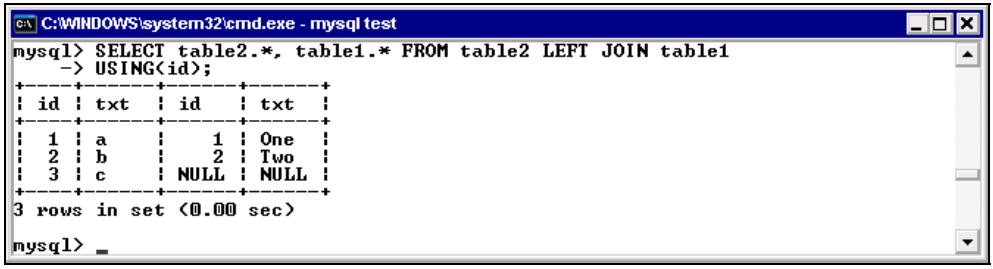


Рис. 12.31. Использование предложения `USING()`

Итак, левое объединение очень удобно применять в том случае, когда требуется узнать, какие записи в таблицах не имеют соответствия.

Обратимся к нашей учебной БД `library`. При помощи объединения `LEFT JOIN`, например, можно узнать, книги с какими инвентарными номерами есть в наличии:

```
mysql> SELECT unit.id_unit FROM unit LEFT JOIN abonement USING(id_unit)
  WHERE abonement.id_unit IS NULL;
```

Объединение `INNER JOIN` делает полное объединение таблиц, связывание задается при помощи предложения `ON`. Следующие два запроса эквивалентны.

```
mysql> SELECT table1.*, table2.* FROM table1, table2 WHERE
  table1.id=table2.id;
mysql> SELECT table1.*, table2.* FROM table1 INNER JOIN table2
  ON table1.id=table2.id;
```

Использование вложенных запросов

Вложенный запрос (подзапрос) — это запрос, заключенный в круглые скобки и вложенный в предложение `WHERE` (`HAVING`) оператора `SELECT` или других операторов, использующих предложение `WHERE`. Вложенный запрос также может содержать в своем предложении `WHERE` (`HAVING`) другой вложенный запрос

и т. д. Нетрудно догадаться, что вложенный запрос служит для того, чтобы при выборке записей таблицы, сформированной основным запросом, можно было использовать данные из других таблиц. Эта возможность появилась в MySQL версии 4.1. В предыдущей версии СУБД (4.0) некоторые вложенные выборки можно было записать как объединения. Об этом будет рассказано чуть позже.

Есть несколько способов создания вложенных запросов. Вложенные запросы включаются в предложение `WHERE` (`HAVING`) с помощью условий `IN`, `EXISTS` или одного из условий сравнения (=, <>, <, <=, > или >=).

Простые вложенные запросы обрабатываются по принципу "снизу вверх". Первым обрабатывается вложенный запрос самого нижнего уровня. Значение, полученное в результате его выполнения, используется при реализации подзапроса более высокого уровня и т. д.

Запросы с коррелированными вложенными запросами обрабатываются системой в обратном порядке. Сначала выбирается первая строка рабочей таблицы, сформированной основным запросом, и из нее выбираются значения тех столбцов, которые используются во вложенном запросе (вложенных запросах). Если эти значения удовлетворяют условиям вложенного запроса, то выбранная строка включается в результат. Затем выбирается вторая строка и т. д., пока в результат не будут включены все строки, удовлетворяющие вложенному запросу (последовательности вложенных запросов).

Следует отметить, что язык SQL обладает большой избыточностью в том смысле, что он часто предоставляет несколько различных способов формулировки одного и того же запроса. Многие из запросов, приведенных в этом уроке, можно было написать и по-другому. Но здесь все же будут приведены их варианты с использованием вложенных запросов. Это связано с необходимостью детального знакомства с созданием и принципом выполнения вложенных запросов, т. к. есть немало задач (особенно на удаление и изменение данных), которые не могут быть реализованы другим способом. Кроме того, разные формулировки одного и того же запроса требуют для своего выполнения различных ресурсов памяти и могут значительно отличаться по времени реализации в разных СУБД.

Простые вложенные запросы

Здесь внутренний оператор `SELECT` используется для получения значения, которое будет использовано в операциях сравнения внешнего оператора `SELECT`. Например:

```
mysql> SELECT title FROM book WHERE id_book=(SELECT id_book FROM book
WHERE id_author=1 and year_issue=2001);
```

```
C:\WINDOWS\system32\cmd.exe - mysql library
mysql> SELECT title FROM book WHERE id_book=
    -> (SELECT id_book FROM book WHERE id_author=1 and year_issue=2001);
+-----------------+
| title           |
+-----------------+
| Евгений Онегин  |
+-----------------+
1 row in set (0.01 sec)

mysql> _
```

Рис. 12.32. Использование вложенного запроса

Здесь перед внутренним запросом стоит оператор сравнения, поэтому необходимо, чтобы в результате вложенного запроса было получено не больше одного значения (рис. 12.32). Если получено больше значений, то выполнение оператора вызовет ошибку (рис. 12.33).

```
Пометить C:\WINDOWS\system32\cmd.exe - mysql library
mysql> SELECT title FROM book WHERE id_book=
    -> (SELECT id_book FROM book WHERE id_author=1 and year_issue=2003);
ERROR 1242 (21000): Subquery returns more than 1 row
mysql>
```

Рис. 12.33. Ошибка при получении более одного значения во вложенном запросе

В некоторых ситуациях можно добавить в оператор `SELECT` предложение `LIMIT 1`. Вложенный запрос такого вида можно использовать для вставки в предложение `WHERE` какой-нибудь функции. Например, чтобы узнать, какая книга самая старая в библиотеке, надо написать:

```
mysql> SELECT * FROM book WHERE year_issue=MIN(year_issue);
```

```
C:\WINDOWS\system32\cmd.exe - mysql library
mysql> SELECT * FROM book WHERE year_issue=MIN(year_issue);
ERROR 1111 (HY000): Invalid use of group function
mysql> SELECT * FROM book WHERE year_issue=
    -> (SELECT MIN(year_issue) FROM book);
+---------+----------------+------------+-----------+--------------+------------
+
| id_book | title          | year_issue | id_author | id_publisher | id_section
|
+---------+----------------+------------+-----------+--------------+------------
+
|      10 | Евгений Онегин |       1985 |         1 |            2 |          1
|
+---------+----------------+------------+-----------+--------------+------------
+
1 row in set (0.00 sec)

mysql>
```

Рис. 12.34. Выборка самой старой книги в нашей библиотеке

Но данный запрос неверен и вызовет ошибку, т. к. в предложении `WHERE` нельзя использовать групповые функции. Выход можно найти с помощью вложенного запроса. Предыдущий запрос можно переписать так:

```
mysql> SELECT * FROM book WHERE year_issue=(SELECT MIN(year_issue)
  FROM book);
```

Результаты обоих запросов представлены на рис. 12.34.

Вложенные запросы в предложениях *EXISTS* и *NOT EXISTS*

Этот способ выборки работает с помощью передачи значений, полученных из внешнего оператора `SELECT`, во внутренний для проверки того, соответствуют ли они условиям, описанным во внутреннем запросе. Данную форму запроса можно использовать при поиске в таблице записей, соответствующих или не соответствующих записям другой таблицы.

Для примера возьмем таблицы `table1` и `table2`, которые мы уже использовали ранее (см. рис. 12.27).

Напишем запрос, определяющий соответствие наших таблиц:

```
mysql> SELECT id FROM table1 WHERE EXISTS (SELECT * FROM table2 WHERE
  table1.id=table2.id);
```

Данный запрос произвел выборку по значениям, присутствующим в обеих таблицах.

Примечание

Здесь для обращения к полю таблицы использовался синтаксис *имя_таблицы*.*имя_поля*, иначе возникла бы неоднозначность, т. к. имена полей в таблицах совпадают.

Результат вложенного запроса в данном случае оценивается как "истина" или "ложь", т. е. возвращает или не возвращает он хоть какие-то строки (не конкретное значение, как в запросе первого вида). Соответственно, в подзапросе можно указывать различные имена полей, главное — понимать, что он должен вернуть значение "истина". Поэтому предыдущий запрос можно записать так:

```
mysql> SELECT id FROM table1 WHERE EXISTS (SELECT 1 FROM table2 WHERE
  table1.id=table2.id);
```

Выражение `NOT EXISTS` определяет несоответствия, т. е. значения, которые присутствуют в одной таблице, но отсутствуют в другой (рис. 12.36):

```
mysql> SELECT txt FROM table2 WHERE NOT EXISTS (SELECT * FROM table1
  WHERE table1.txt=table2.txt);
mysql> SELECT id FROM table2 WHERE NOT EXISTS (SELECT * FROM table1
  WHERE table1.id=table2.id);
```

```
C:\WINDOWS\system32\cmd.exe - mysql test
mysql> SELECT id FROM table1
    -> WHERE EXISTS (SELECT * FROM table2 WHERE table1.id=table2.id);
+----+
| id |
+----+
|  1 |
|  2 |
+----+
2 rows in set (0.00 sec)

mysql> SELECT id FROM table1
    -> WHERE EXISTS (SELECT 1 FROM table2 WHERE table1.id=table2.id);
+----+
| id |
+----+
|  1 |
|  2 |
+----+
2 rows in set (0.00 sec)

mysql> _
```

Рис. 12.35. Использование выражения EXISTS

```
C:\WINDOWS\system32\cmd.exe - mysql test
mysql> SELECT txt FROM table2
    -> WHERE NOT EXISTS (SELECT * FROM table1 WHERE table1.txt=table2.txt);
+------+
| txt  |
+------+
| a    |
| b    |
| c    |
+------+
3 rows in set (0.00 sec)

mysql> SELECT id FROM table2
    -> WHERE NOT EXISTS (SELECT * FROM table1 WHERE table1.id=table2.id);
+----+
| id |
+----+
|  3 |
+----+
1 row in set (0.00 sec)

mysql> _
```

Рис. 12.36. Использование выражения NOT EXISTS

Вложенные запросы в предложениях *IN* и *NOT IN*

Этот вид запросов работает так: вложенный оператор SELECT возвращает значения из одного поля, которые будут оцениваться во внешнем операторе SELECT. Например (рис. 12.37 и 12.38):

```
mysql> SELECT txt FROM table2 WHERE txt NOT IN (SELECT txt FROM table1);
mysql> SELECT id FROM table2 WHERE id NOT IN (SELECT id FROM table1);
mysql> SELECT id FROM table1 WHERE id IN (SELECT id FROM table2);
```

```
C:\WINDOWS\system32\cmd.exe - mysql test
mysql> SELECT txt FROM table2
    -> WHERE txt NOT IN (SELECT txt FROM table1);
+------+
| txt  |
+------+
| a    |
| b    |
| c    |
+------+
3 rows in set (0.00 sec)

mysql> SELECT id FROM table2
    -> WHERE id NOT IN (SELECT id FROM table1);
+----+
| id |
+----+
|  3 |
+----+
1 row in set (0.00 sec)

mysql> _
```

Рис. 12.37. Использование выражения `NOT IN`

```
C:\WINDOWS\system32\cmd.exe - mysql test
mysql> SELECT id FROM table1
    -> WHERE id IN (SELECT id FROM table2);
+----+
| id |
+----+
|  1 |
|  2 |
+----+
2 rows in set (0.00 sec)

mysql>
```

Рис. 12.38. Использование выражения `IN`

Последние примеры — это переделка предыдущих запросов выборки, где применялись предложения `EXISTS` и `NOT EXISTS`. Запросы, использующие вложенную выборку, можно переписать с использованием объединений (как и поступали при работе с MySQL версий до 4.1). Например:

```
mysql> SELECT title FROM book WHERE id_book=(SELECT id_book FROM unit
  WHERE id_unit=20);
```

Подобный запрос можно написать, используя простое объединение:

```
mysql> SELECT title FROM book,unit WHERE book.id_book=unit.id_book and
  unit.id_unit=20;
```

Если вы работаете с MySQL 4.1 и выше, то это не значит, что все запросы выборки теперь можно писать с использованием вложенных запросов, —

стоит проверить варианты запросов, использующие объединение, т. к. довольно часто они работают эффективнее.

Объединение *UNION*

Оператор UNION, реализованный в MySQL 4.0.0, предназначен для объединения результатов выполнения нескольких операторов SELECT в один набор результатов. Эти операторы SELECT являются обычными запросами выборки данных.

Оператор UNION имеет следующие особенности.

- Имена и типы данных полей для результирующего набора устанавливаются по имени и типу первого оператора SELECT. Остальные операторы SELECT в объединении UNION должны иметь одинаковое количество полей, но им совсем необязательно иметь одинаковые тип и имя.
- По умолчанию результирующий набор запроса с оператором UNION не содержит повторяющихся строк (как при использовании ключевого слова DISTINCT).
- Для упорядочения результата необходимо добавить предложение ORDER BY в последний оператор SELECT. Оно будет относиться ко всему результирующему набору.
- Можно осуществлять выборку из одной таблицы различных наборов, удовлетворяющих разным условиям.

Для того чтобы создать оператор UNION, нужно написать несколько операторов SELECT, вставив между ними ключевое слово UNION:

```
mysql> SELECT title FROM book UNION SELECT author FROM author;
```

В результирующей выборке после выполнения данного запроса мы видим (рис. 12.39) названия всех книг, имеющихся в библиотеке, а также фамилии всех авторов.

Во время выполнения запроса поля подбираются по порядку следования, а не по имени. Именно поэтому следующие запросы (рис. 12.40) дают разный результат:

```
mysql> SELECT id_author, year_issue FROM book UNION SELECT id_author,
  author FROM author;
mysql> SELECT id_author, year_issue FROM book UNION SELECT author,
  id_author FROM author;
```

И в первом и во втором случае поля, выбранные из таблицы book, задают типы, полученные в результате работы оператора UNION.

```
C:\WINDOWS\system32\cmd.exe - mysql library
mysql> SELECT title FROM book UNION SELECT author FROM author;
+-------------------------+
| title                   |
+-------------------------+
| Нос                     |
| Сказка о золотой рыбке  |
| Руслан и Людмила        |
| Мартышка и очки         |
| Евгений Онегин          |
| Шинель                  |
| Идиот                   |
| Сборник стихов          |
| Пушкин                  |
| Крылов                  |
| Гоголь                  |
| Есенин                  |
| Достоевский             |
+-------------------------+
13 rows in set (0.00 sec)

mysql> _
```

Рис. 12.39. Использование оператора UNION

Как уже говорилось, при использовании оператора UNION в результирующей выборке исключены повторения. Для того чтобы появились повторения, необходимо добавить ключевое слово ALL после первого ключевого слова UNION:

```
mysql> SELECT id_author, year_issue FROM book UNION SELECT author,
  id_author FROM author;
```

Результат работы запроса с ключевым словом ALL можно увидеть на рис. 12.40. У нас появилось одно повторение — дважды встречается 1 в поле id_author и дважды 2003 год в поле year_issue.

При сортировке предложение ORDER BY может задаваться для отдельного оператора SELECT, для этого оператор SELECT вместе с ORDER BY необходимо поместить в скобки.

В запросах с использованием оператора UNION можно указывать предложение LIMIT *n*. Например:

```
mysql> SELECT * FROM table1 UNION SELECT * FROM table2 LIMIT 1;
```

В этом случае он относится ко всему результирующему набору. Если заключить его в скобки вместе с оператором SELECT, то он будет действовать именно на этот оператор:

```
mysql> (SELECT * FROM table1 LIMIT 1) UNION (SELECT * FROM table2
  LIMIT 1);
```

Как уже говорилось, вы можете производить выборку различных результирующих наборов, удовлетворяющих разным условиям, в том числе и из одной таблицы. Оператор UNION отсутствовал в MySQL версий до 4.0, поэтому приходилось пользоваться другими методами — сделать выборки во временную таблицу, а затем сделать выборку из временной таблицы. Это выглядело примерно так:

```
SELECT TEMPORARY TABLE tmp_table TYPE=HEAP SELECT ...
  FROM table1 WHERE...;
INSERT INTO tmp_table SELECT ... FROM table2 WHERE...;
...
SELECT * FROM tmp_table;
```

```
C:\WINDOWS\system32\cmd.exe - mysql library

mysql> SELECT id_author,year_issue FROM book UNION SELECT id_author,author FROM
author;
+-----------+-------------+
| id_author | year_issue  |
+-----------+-------------+
|         3 | 2003        |
|         1 | 2003        |
|         2 | 2003        |
|         1 | 2001        |
|         3 | 2001        |
|         5 | 1991        |
|         5 | 1995        |
|         4 | 1995        |
|         1 | 1985        |
|         1 | Пушкин      |
|         2 | Крылов      |
|         3 | Гоголь      |
|         4 | Есенин      |
|         5 | Достоевский |
+-----------+-------------+
14 rows in set (0.00 sec)

mysql> SELECT id_author,year_issue FROM book UNION SELECT author,id_author FROM
author;
+-------------+------------+
| id_author   | year_issue |
+-------------+------------+
| 3           |       2003 |
| 1           |       2003 |
| 2           |       2003 |
| 1           |       2001 |
| 3           |       2001 |
| 5           |       1991 |
| 5           |       1995 |
| 4           |       1995 |
| 1           |       1985 |
| Пушкин      |          1 |
| Крылов      |          2 |
| Гоголь      |          3 |
| Есенин      |          4 |
| Достоевский |          5 |
+-------------+------------+
14 rows in set (0.00 sec)

mysql> SELECT id_author,year_issue FROM book UNION ALL SELECT author,id_author F
ROM author;
+-------------+------------+
| id_author   | year_issue |
+-------------+------------+
| 3           |       2003 |
| 1           |       2003 |
| 1           |       2003 |
| 2           |       2003 |
| 1           |       2001 |
| 3           |       2001 |
| 5           |       1991 |
| 5           |       1995 |
| 4           |       1995 |
| 1           |       1985 |
| Пушкин      |          1 |
| Крылов      |          2 |
| Гоголь      |          3 |
| Есенин      |          4 |
| Достоевский |          5 |
+-------------+------------+
15 rows in set (0.00 sec)

mysql> _
```

Рис. 12.40. Примеры работы оператора `UNION`

MySQL автоматически удаляет временные (TEMPORARY) таблицы после завершения текущего сеанса. Здесь использовалась таблица типа HEAP, она хранится в памяти, что позволит увеличить быстродействие. Хотя вы можете удалить таблицу самостоятельно, используя оператор DROP TABLE — это позволит освободить ресурсы, особенно если запросы продолжают выполняться. Это всегда приходилось делать в версиях до 3.23, поскольку не существовало таблиц типа TEMPORARY и отсутствовали таблицы типа HEAP.

Удаление и обновление нескольких таблиц

В MySQL, начиная с версии 4, появилась возможность удаления и обновления нескольких таблиц.

```
mysql> DELETE table1 FROM table1, table2 WHERE table1.id=table2.id;
```

Данный запрос удалит записи из таблицы table1, имеющие соответствующие значения в таблице table2. В предложении FROM нужно перечислить таблицы, участвующие в запросе, а в предложении WHERE — задать условия соответствия записей.

Можно удалить записи одновременно из нескольких таблиц (там, где обнаружено соответствие):

```
mysql> DELETE table1, table2 FROM table1, table2 WHERE
  table1.id=table2.id;
```

Для удаления несоответствующих записей можно написать:

```
mysql> DELETE table1 FROM table1 LEFT JOIN table2 ON table1.id=table2.id
  WHERE table2.id IS NULL;
```

или

```
mysql> DELETE FROM table1 USING table1, table2 WHERE table1.id=table2.id;
mysql> DELETE FROM table1, table2 USING table1, table2 WHERE
  table1.id=table2.id;
mysql> DELETE FROM table1 USING table1 LEFT JOIN table2 ON
  table1.id=table2.id WHERE table2.id IS NULL;
```

Примечание

Три последние варианта запроса поддерживаются, начиная с версии 4.0.2.

Оператор UPDATE имеет те же принципы работы на нескольких таблицах, что и оператор DELETE. Например:

```
mysql> UPDATE table1, table2 SET table1.txt=table2.txt WHERE
  table1.id=table2,id;
```

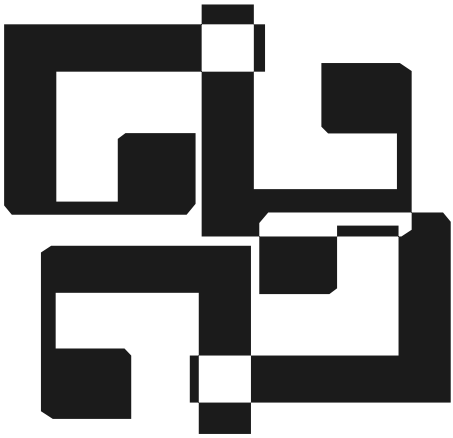
Несколько слов о транзакциях

Транзакция — это набор операторов SQL, которые будут выполнены как одна операция, не прерываемая другими клиентами. Транзакции обеспечивают неприкосновенность данных во время вашей работы с ними. Это означает, что MySQL блокирует некоторые операторы SQL, чтобы клиенты не смогли повлиять на работу друг друга.

Взаимное влияние могут оказывать и операции. Транзакция группирует операторы в единую исполняемую структуру, что позволяет избежать проблем при параллельной работе большого количества клиентов.

Транзакции поддерживают операции `commit` (выполнение) и `rollback` (откат). То есть если какая-то часть транзакции дала сбой, есть возможность отменить результаты выполнения предыдущих операций. Соответственно БД останется в том виде, который она имела до начала выполнения транзакции.

В MySQL транзакции поддерживают таблицы типов `BDB` и `InnoDB`.



ЧАСТЬ IV

PHP и MySQL

PHP — это серверный язык создания сценариев, предназначенный специально для генерации web-приложений. Разработка языка была начата в 1995 году Расмусом Лердорфом (Rasmus Lerdorf). Изначально название PHP являлось сокращением от Personal Home Page (персональная начальная страница), но в дальнейшем эта аббревиатура стала означать PHP Hypertext Preprocessor (препроцессор языка PHP). Сейчас язык PHP стал очень популярным благодаря легкости изучения и простоте встраивания в HTML-код и используется на нескольких миллионах доменах. Язык PHP, как и СУБД MySQL, является продуктом с открытым исходным кодом (open-source).

Совместное использование PHP и MySQL позволит вам создавать динамические сайты с информацией, изменяемой в реальном режиме времени, и предоставит большие возможности по настройке структуры проекта. Для использования PHP вам нужно установить на своем компьютере интерпретатор языка PHP и web-сервер (например, Apache).

УРОК 13

PHP в HTML

Основная задача языка PHP — интерпретация сценариев (программ) для генерации web-страниц, отсылаемых программе-клиенту (браузеру). Сценарий (скрипт — от англ. script) может содержать как PHP-, так и HTML-код. HTML-код пересылается в литеральном представлении, а PHP-код выполняется, и клиенту отсылается результат его работы. Таким образом, пользователь никогда не видит PHP-кода. Вы можете использовать PHP для обработки данных, которые пользователь ввел в форму.

Для встраивания PHP-кода в HTML должны присутствовать открывающие и закрывающие теги, идентифицирующие PHP-код.

Есть несколько способов вставки PHP кода:

- `<?php` *`PHP_код`* `?>` — стандартный способ;
- `<?` *`PHP_код`* `?>` — использование коротких тегов (этот способ работает только при соответствующей настройке сервера);
- `<script language="php">` *`PHP_код`* `</script>` — аналогично вставке кода на JavaScript.

Файл, содержащий PHP-код, должен иметь расширение PHP. Все созданные вами PHP-сценарии будут размещаться в корневом каталоге сервера. В ОС Linux этот каталог называется PUBLIC_HTML. В Windows, если вы используете Apache как web-сервер, таким каталогом является HTDOCS. В случае использования IIS как web-сервера это будет каталог WWWROOT. Доступ к корневому каталогу вы можете получить, введя команду **`localhost`** в адресной строке браузера.

Примеры данной книги были разработаны с использованием сервера Apache. Вы должны настроить конфигурационный файл HTTPD.CONF для того, чтобы сервер Apache распознавал файлы с расширением PHP. Необходимо добавить в него следующие строки:

```
AddType application/x-httpd-php .php
ScriptAlias /__php__/ "c:/php/"
Action aplication/x-httpd-php /__php__/php.exe
```

Примечание

Здесь предполагается, что интерпретатор PHP установлен в корневой каталог диска C:.

Для проверки работы PHP откройте любой текстовый редактор и наберите следующий код:

```
<?php
phpinfo();
?>
```

Сохраните файл под именем TEST.PHP в корневом каталоге сервера и запустите его в браузере. На рис. 13.1 показан результат работы сценария.

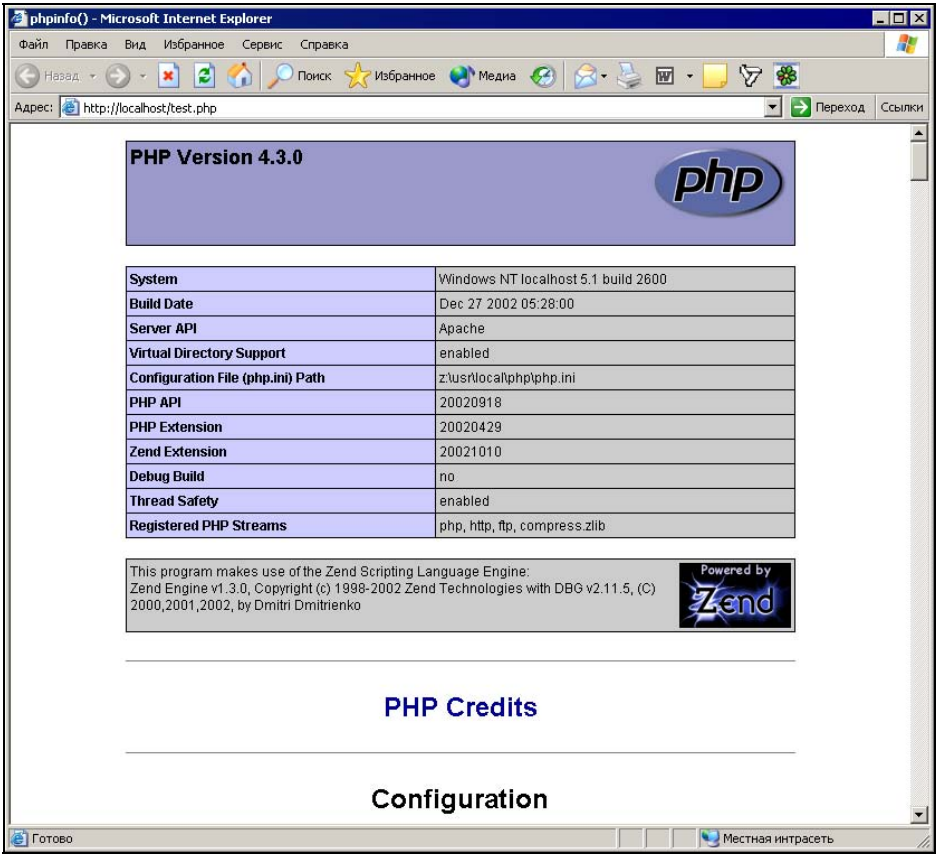
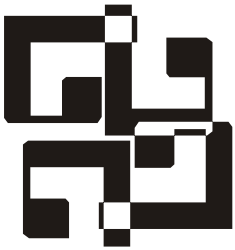


Рис. 13.1. Если интерпретатор PHP установлен, вы увидите такую страницу

УРОК 14



Основы языка PHP

Все команды (утверждения) в коде должны заканчиваться точкой с запятой (`;`). Вы можете размещать несколько команд на одной строке, заканчивая их точкой с запятой. Но для ясности кода программы лучше этого не делать. В PHP также присутствуют управляющие элементы (условные операторы, циклы и т. д.), после которых точка с запятой не требуется.

В код программы можно вставлять комментарии, которые будут игнорироваться как PHP-интерпретатором, так и HTML-браузером. Использование комментариев помогает программисту вспомнить, о чем он думал в тот момент. Однострочный комментарий предваряется двумя прямыми слэшами (`//`) или символом решетки (`#`), а многострочный заключается между открывающей и закрывающей "скобками" (`/*` и `*/`).

Переменные

Переменные в PHP начинаются со знака доллара (`$`). Имя переменной может включать латинские буквы, цифры и символ подчеркивания (_) и должно начинаться с буквы или символа подчеркивания (_). Имя переменной чувствительно к регистру букв, т. е. `$a` и `$A` — это разные переменные. При объявлении переменной обычно не задается определенный тип, как во многих других языках. Тип переменной (т. е. тип данных, который она будет хранить) определяется по контексту использования.

В PHP есть следующие типы переменных:

- `integer`
- `floating point`
- `string`

- object
- array

"Пустая" переменная объявляется при помощи ключевого слова VAR:

```
VAR $some_var;
```

Переменная также может быть объявлена, когда она впервые используется:

```
$some_var="value";
```

Тип *integer*

Переменные типа integer могут содержать целое число в пределах от –2 биллионов до +2 биллионов в десятичном, восьмеричном и шестнадцатеричном исчислении. Например:

```
$var1=100;   /* Десятичное значение */
$var2=0144;  /* Восьмеричное значение */
$var3=0x64;  /* Шестнадцатеричное значение */
```

Тип *floating point*

Переменная типа floating point — это число с плавающей точкой (дробь). Например:

```
$var1=2.34;
$var2=234e1;
```

Тип *string*

Строка (переменная типа string) — это любая комбинация из букв, цифр и специальных символов. При задании строка оформляется в одиночных (апострофы) или двойных кавычках. Если строка задана в двойных кавычках, она будет просмотрена на наличие переменных. Если они присутствуют, то вместо имен переменных будут подставлены их значения.

Примеры строковых переменных:

```
$var="23";
$str1='Переменная содержит значение $var';
$str2="Переменная содержит значение $var";
```

При выводе наших переменных мы получим соответственно:

```
Переменная содержит значение $var
Переменная содержит значение 23
```

Возможно, вам понадобится включить в строку специальные символы.

Например, вы можете использовать обратный слэш (\) для вывода символа двойной кавычки ("). Если написать

```
$str1="фирма "Графика"";
```

это вызовет ошибку, но если использовать обратный слэш (\)

```
$str1="фирма \"Графика\"";
```

то ошибки не будет.

Примечание

При наличии в строке символа обратного слэша (\) следующий за ним символ воспринимается как литерал.

В табл. 14.1 описаны Escape-последовательности (обратный слэш и специальный символ), применяемые в строковых значениях.

Таблица 14.1. *Escape-последовательности, применяемые в языке PHP*

Escape-последовательность	Значение
\n	Начало новой строки
\r	Перевод каретки
\t	Символ табуляции
\\	Символ (\)
\"	Символ (")
\$	Символ ($)
\0	Восьмеричное значение
\x	Шестнадцатеричное значение

Тип *object*

Объекты (переменные типа `object`) являются экземплярами *класса*. Для создания объекта необходимо прежде создать класс. Класс может содержать набор переменных и функций для работы с ними.

Примечание

Более подробную информацию вы найдете в документации по PHP (**http://php.net**) или в учебниках по языку PHP.

Тип *array*

Переменные типа `array` используются при объявлении *массивов*. В PHP есть два вида массивов:

- массив с целочисленными индексами;
- массив с индексированный строками (хэш).

Например, для создания массива, содержащего четыре значения, нужно написать:

```
$myarr=array("value1", "value2" , "value3", "value4");
```

Каждому элементу, помещенному в массив, присваивается индекс (начиная с 0). Таким образом, `$myarr[0]` даст нам `"value1"`, а `$myarr[3]` — `"value4"`. Конечно, можно добавить в массив элемент: `$myarr[]="value5";`

Создать хэш можно так:

```
$myhash=array('size'=>'large', 'style'=>'italic', 'family'=>'Arial');
```

Операции

PHP поддерживает несколько видов операций:

- присваивание;
- арифметические операции;
- логические операции;
- конкатенация;
- сравнение.

Арифметические операции

В табл. 14.2 кратко описаны арифметические операции.

Таблица 14.2. *Арифметические операции*

Оператор	Пример операции	Описание
+	`$a+$b`	Вычисление суммы значений переменных `$a` и `$b`
-	`$a-$b`	Вычисление разности значений переменных `$a` и `$b`
*	`$a*$b`	Вычисление произведения значений переменных `$a` и `$b`
/	`$a/$b`	Вычисление частного значений переменных `$a` и `$b`
%	`$a%$b`	Вычисление остатка от деления по модулю значения переменной `$a` на значение переменной `$b`

Логические операции

Логическая операция (табл. 14.3) определяет в соответствии с определенными критериями, является ли возвращаемое значение истинным (`true`) или ложным (`false`).

Таблица 14.3. *Логические операции*

Оператор	Пример операции	Описание
`and` или `&&`	`$a and $b` `$a && $b`	Возвращает значение `true`, если истинны значения обеих переменных
`or` или `||`	`$a or $b` `$a || $b`	Возвращает значение `true`, если истинно значение хотя бы одной из переменных
`not` или `!`	`not $a` `!$a`	Логическое отрицание, инвертирует значение переменной

Конкатенация

Оператор конкатенации (`.`) осуществляет сложение двух строк (две строки объединяются). Например:

```
<?php
$a='Оля';
echo 'Привет '.$a;
?>
```

или

```
<?php
$h=12;
$m=35;
$time=$h.':'.$m;
echo $time;
//получим 12:35
?>
```

Сравнение

Операции сравнения определяют отношение между двумя переменными или выражениями и возвращают значение `true` или `false`. Операторы сравнения приведены в табл. 14.4.

Таблица 14.4. *Операторы сравнения*

Оператор	Описание
==	Равно
!=	Не равно
>	Больше, чем
<	Меньше, чем
>=	Больше или равно
<=	Меньше или равно

Структуры управления

PHP поддерживает несколько структур управления ходом программы. К ним относятся `if`, `for`, `while`, `switch` и др.

if / elseif

Данная структура позволяет реализовать программную логику. Вы можете проверять различные выражения и в зависимости от результата проверки выполнять то или иное действие. Базовый синтаксис можно описать так:

```
<?php
if(выражение1)
  {
  действия1;
  }
elseif(выражение2)
  {
  действия2;
  }
  else
    {
    действия по умолчанию;
    }
?>
```

действия1 и *действия2* выполняются лишь в том случае, если *выражение1* в условии `if` или *выражение2* в условии `elseif` оказалось истинным. Если все описанные выражения оказались ложными, то выполняются *действия по умолчанию*, прописанные в разделе `else` ("иначе").

for и foreach

Оператор цикла `for` выполняет заданный блок кода определенное количество раз (итераций).

```
<?php
for(выражение1; выражение2; выражение3)
{
действия
}
?>
```

Параметры: *выражение1* — начальное значение счетчика повторений; *выражение2* — описание критерия (условия), в соответствии с которым выполнение цикла `FOR` будет прекращено; *выражение3* — изменение значения счетчика (увеличение или уменьшение).

Пример (`echo` — это конструкция языка, предназначенная для вывода одной или более строк на экран):

```
<?php
echo "<select name='num'>\n";
for($i=0;$i<10;$i++)
{
echo "<option>$i</option>\n";
}
echo "</select>";
?>
```

Результат представлен на рис. 14.1.

Структура `FOREACH` предназначена для перебора элементов числового массива или хэша.

```
<?php
foreach($array as $value)
{
PHP-код
}
?>
```

В случае хэша синтаксис следующий:

```
<?php
foreach($hash as $key=>$value)
{
PHP-код
}
?>
```

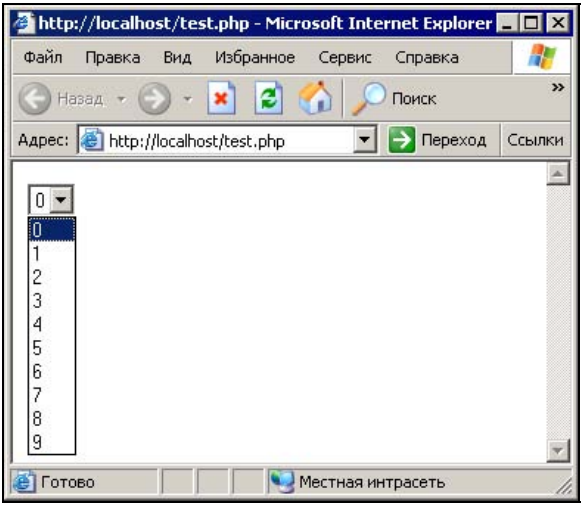


Рис. 14.1. Результат работы примера программы на PHP

while

Оператор цикла `while` повторяет блок кода до тех пор, пока выполняется некоторое условие. Как только условие станет ложным, выполнение цикла заканчивается.

```
<?php
while(условие)
{
действия
}
?>
```

switch

Оператор `switch` очень похож на выполнение серии операторов `if` с одним выражением. Довольно часто приходится сравнивать значение переменной (или выражения) с разными значениями и выполнять различные блоки кода в зависимости от результата этого сравнения.

Синтаксис оператора `switch`:

```
<?php
switch(выражение)
{
```

```
case значение1:
  действия1;
  break;
case значение2:
  действия2;
  break;
...
default:
действия_по_умолчанию;
}
?>
```

Данный оператор выполняется построчно. Блок кода *действия* будет выполнен только тогда, когда будет найдено *значение*, совпадающее со значением *выражение* в операторе `switch`. PHP будет продолжать выполнять команды до конца блока `switch` или до первого оператора `break`. Если не поставить `break` в конце оператора `case`, то выполнятся и все оставшиеся участки кода. Так что не забывайте об этом (хотя, в некоторых ситуациях ставить `break` и не требуется).

действия_по_умолчанию — это код, который выполнится в случае, если ни одно указанное *значение* не совпало со значением выражения. Оператор `default` должен находиться в самом конце списка операторов `case`.

Пример:

```
<?php
switch($a)
{
case 1: echo "i равно 1"; break;
case 2: echo "i равно 2"; break;
case 3: echo "i равно 3"; break;
}
?>
```

Функции

Функции являются традиционными конструкциями языка. Они выполняют набор определенных действий, используя различные параметры, и возвращают результат своей работы в точку вызова функции. Язык PHP поддерживает два вида функций:

- функции, объявленные программистом (пользовательские);
- стандартные (встроенные) функции языка.

Пользовательские функции

Если вам требуется периодически выполнять серию команд, то вы можете поместить эти команды в функцию. Это сделает ваш код более гибким и удобочитаемым.

Вместо того чтобы постоянно прописывать серию одних и тех же команд, вы можете сделать вызов функции (предварительно объявленной). Это будет намного короче. Кроме этого, если появится необходимость внесения каких-либо изменений, их нужно будет сделать один раз, исправив объявленную вами функцию.

Для создания функции в PHP-коде используйте следующий синтаксис:

```
function имя_функции(параметры_функции);
{
  действия;
}
```

Параметры: *имя_функции* — имя функции, задается программистом; *параметры_функции* — параметры, используемые функцией (в зависимости от значения этих параметров может изменяться результат работы функции); *действия* — тело функции (набор команд, которые она должна выполнить).

Встроенные функции

Это функции, описанные разработчиками языка при его создании, которые выполняют часто применяемые операции. Для того чтобы использовать встроенную функцию, вам необходимо знать имя этой функции и список ее параметров. Обращение к функции (вызов) происходит по ее имени.

Примечание

Более подробную информацию вы найдете в документации по PHP (на сайте **http://php.net**).

УРОК 15

Отображение и вставка данных

Настало время посмотреть, как же PHP осуществляет доступ к СУБД MySQL и отображает результаты запросов на web-странице.

Примечание

В листингах PHP-функции, предназначенные для работы с MySQl, выделены полужирным начертанием.

В листинге 15.1 приведен пример простого PHP-сценария.

Листинг 15.1. Простой PHP-сценарий

```
<html>
<head>
<title>Список читателей</title>
</head>
<body bgcolor="#ffffff">
<h3>Список читателей библиотеки</h3>
<?php
#соединяемся с сервером
$id_con=mysql_connect("localhost", "root", "")
or die("Невозможно соединиться с сервером");
#выбираем БД
mysql_select_db("library") or die("Невозможно выбрать БД");
#посылаем запрос
$res_id=mysql_query("select * from reader") or die("Неверный запрос");
#обрабатываем результат
while($mr=mysql_fetch_row($res_id))
{
echo "$mr[1]<br>";
}
```

```
mysql_close($id_con);
?>
</body>
</html>
```

Текст этой программы выполняет следующие действия:

1. Осуществляет соединение с сервером MySQL. Здесь используется функция `mysql_connect()`, ее параметрами являются имя сервера, имя пользователя и пароль. В переменной `$id_con` возвращается идентификатор соединения. В случае неудачи на экран выводится сообщение "Невозможно соединиться с сервером".

Примечание

Более подробная информация об этой и других функциях приведена в *приложении 2*.

2. При помощи функции `mysql_select_db()` делаем БД `library` активной.

3. После этого можно посылать запросы к БД, что мы и сделали посредством функции `mysql_query()`. Аргументом функции является SQL-запрос, который выбирает все поля из таблицы `reader`. В переменную `$res_id` будет возвращен идентификатор результата, а сам результат выборки хранится в памяти.

4. Для получения данных по идентификатору результата используем функцию `mysql_fetch_row()`. Она возвращает результирующий набор данных в массиве (мы его назвали `$mr`). Количество элементов массива соответствует количеству полей, выбираемых SQL-запросом. В нашем случае их будет три: в первом элементе будет находиться номер читателя (поле `id_reader`), во втором — его фамилия (поле `reader`), а в третьем — номер адреса (поле `id_addr`). Один вызов функции вернет нам только первую запись из таблицы `reader`. Чтобы получить данные обо всех читателях, мы организуем цикл `while`, который будет вызывать функцию `mysql_fetch_row()`, пока мы не получим все строки из результирующего набора (все записи из таблицы). Таким образом, на каждой итерации цикла мы будем иметь массив, содержащий очередную запись из таблицы `reader`. Остается только организовать вывод — ну, здесь уж все совсем просто. Печатаем второй элемент массива (фамилию читателя) и делаем перевод строки.

5. Закрываем соединение с помощью функции `mysql_close()`, хотя это необязательно (соединение будет закрыто автоматически после окончания работы сценария) (рис. 15.1).

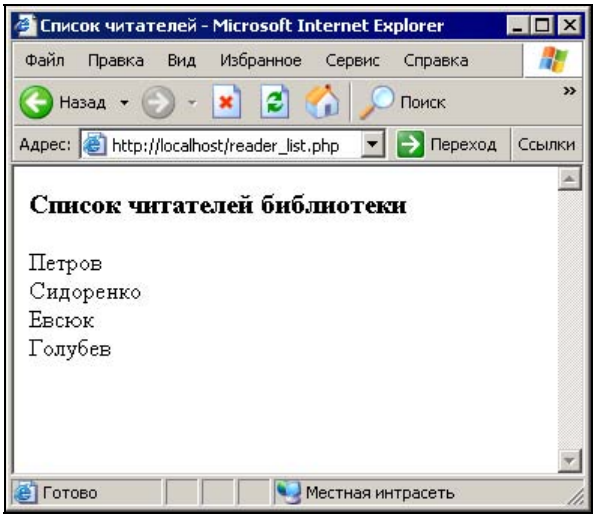


Рис. 15.1. Результат работы сценария

Итак, выводить данные таблиц мы научились. Но этого не достаточно.

Давайте создадим несложный web-интерфейс и на примере таблицы `author` посмотрим, как можно работать с данными БД.

Для начала откройте текстовый редактор, наберите следующий код и сохраните его в файле с именем DBCONNECT.PHP в корневом каталоге сервера.

```
<?php
$id_con=mysql_connect("localhost", "root", "")
or die("Невозможно соединиться с сервером");
mysql_select_db("library") or die("Невозможно выбрать БД");
?>
```

Данный сценарий осуществляет соединение с сервером и выбирает БД или выводит сообщение об ошибке в случае неудачи. Этот файл будет подключаться к остальным сценариям.

Для начала создадим сценарий AUTHOR_LIST.PHP (листинг 15.2), который выберет фамилии всех авторов и сформирует на странице раскрывающийся список (`<select>`). Данный скрипт схож с предыдущим, разница лишь в форме вывода.

Примечание

В самом начале программы мы подключим файл DBCONNECT.PHP.

Листинг 15.2. Файл AUTHOR_LIST.PHP

```
<?php
require "dbconnect.php";
?>

<html>
<head>
<title>Список авторов</title>
</head>
<body>

<?php
  $res_id=mysql_query("select * from author")
  or die(mysql_error());
  #узнаем общее количество выбранных записей
  $num=mysql_num_rows($res_id);
  echo "Всего в базе: <b>$num</b>";
?>
<h3>Список авторов</h3>
<form method="post">
<select name="auth_list">

<?php
while($am=mysql_fetch_row($res_id))
{
    echo "<option value='$am[0]'>$am[1]</option>\n";
}
?>
</select>
</form>
</body>
</html>
```

Откроем файл в браузере и просмотрим результат (рис. 15.2).

Итак, в результате работы сценария мы получили сгенерированную страницу со списком всех имеющихся в БД авторов (пока их только пять).

Нужно также предусмотреть возможность добавления в БД записей новых авторов. Для этого мы немного модифицируем предыдущий скрипт, добавив в форму текстовое поле и кнопку отправки данных (рис. 15.3).

В листинге 15.3 я привел измененную часть кода.

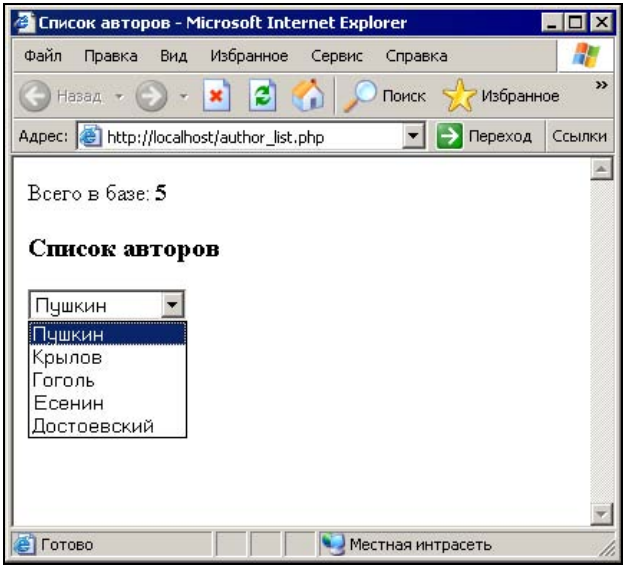


Рис. 15.2. Список авторов

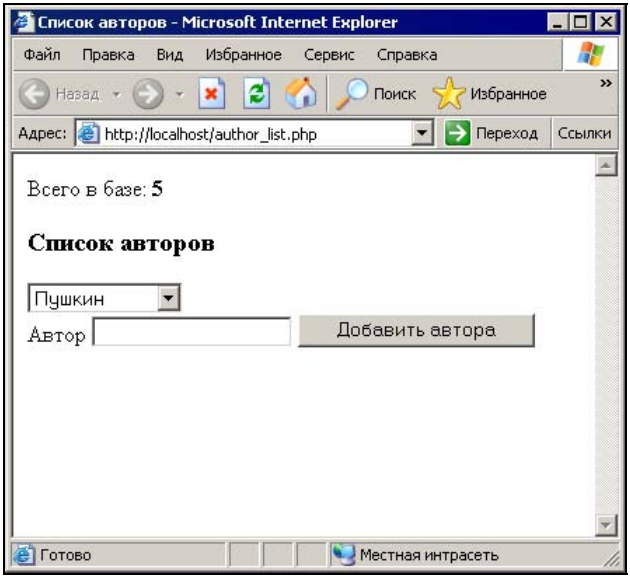


Рис. 15.3. Результат изменений

Листинг 15.3. Изменение кода предыдущего скрипта

```
<form method="post" action="add_author.php">
<select name="auth_list">
<?php
while($am=mysql_fetch_row($res_id))
{
  echo "<option value='$am[0]'>$am[1]</option>\n";
}
?>
</select>
<br>
Автор <input type="text" name="new_author">
<input type="submit" name="send" value="Добавить автора">
</form>
```

Теперь нужно реализовать обработку формы. В форму, как вы заметили, был добавлен параметр `action="add_author.php"`. Это сценарий, который будет запускаться при отправке формы и обрабатывать пришедшие данные, а именно — добавлять нового автора в таблицу. Наберите следующий код (листинг 15.4) и сохраните его в файле в корневом каталоге.

Листинг 15.4. Обработка формы

```
<?php
require "dbconnect.php";

if(isset($_POST['new_author']))
{
  $query="insert into author values('','".$_POST['new_author']."')";
  if(mysql_query($query))
    {
      echo "Запись добавлена";
    }
  else
    {
      echo mysql_error();
    }
}
?>
```

После добавления автора вы увидите сообщение "Запись добавлена". Вернитесь в браузере на страницу AUTOR_LIST.PHP и нажмите кнопку **Обновить**. В раскрывающемся списке вы увидите добавленного автора (рис. 15.4).

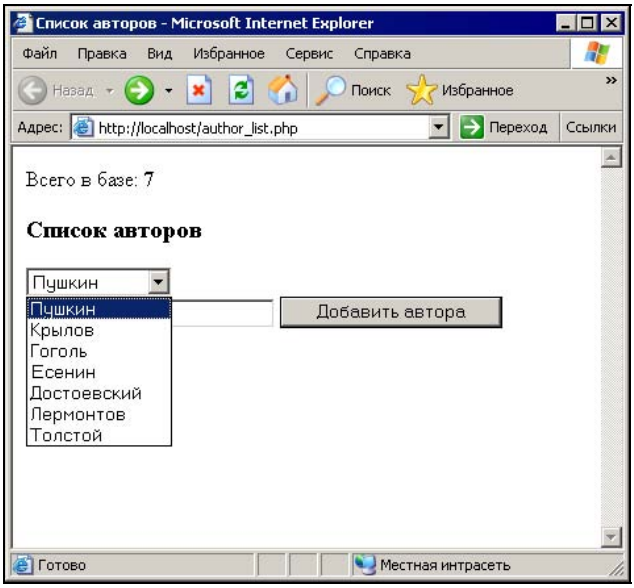


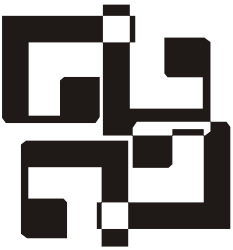
Рис. 15.4. Просмотр результатов

Как можно заметить, в обоих наших сценариях мы использовали функцию `require()` для подключения файла DBCONNECT.PHP (этот сценарий выполняет соединение с сервером и выбор БД). Таким образом, использование подключаемых файлов позволяет избежать многократного описания процесса подключения в каждом из создаваемых сценариев. Кроме того, это позволяет вносить глобальные изменения сразу во все сценарии, использующие данный файл (например, при изменении имени сервера).

Кроме функции `require()` можно использовать функцию `include()`. Эти функции идентичны почти во всем, за исключением того, как они обрабатывают неудачное выполнение. Функция `include()` выводит на экран предупреждение (`Warning!`), а функция `require()` генерирует ошибку (`Fatal Error`). Другими словами, используйте функцию `require()`, если вам нужно, чтобы отсутствующий файл останавливал обработку страницы. В случае использования функции `include()` скрипт все равно продолжит работу.

Для вывода сообщений об ошибке мы использовали конструкцию `die()`. Она печатает сообщение об ошибке и завершает работу сценария. Также для завершения работы сценария можно использовать функцию `exit()`.

УРОК 16



Обработка результатов запроса

Давайте рассмотрим в деталях процесс выполнения запроса к MySQL и обработку результатов. Для отправки запроса мы использовали функцию `mysql_query()` и указывали в качестве аргумента SQL-запрос или переменную, содержащую запрос:

```
$result_id=mysql_query("запрос");
```

или

```
$query="запрос";
$result_id=mysql_query($query);
```

Эта функция возвращает идентификатор результирующего набора (при запросе `SELECT`) или значение `FALSE` в случае неудачи. Если использовались такие запросы, как `DELETE`, `INSERT`, `UPDATE`, `REPLACE`, функция возвращает значение `TRUE`, если все прошло успешно, или значение `FALSE` при ошибке. Получение значения `TRUE` после запроса на изменение данных еще не означает, что все в порядке. Допустим, мы написали следующий код:

```
$result_id=mysql_query("delete from reader where id_reader=35");
if($result_id)
  {
    echo "Читатель с номером 35 удален";
  }
else
  {
    echo "Ошибка запроса";
  }
```

В данном случае оператор `DELETE` вернет значение `TRUE`, даже если ничего удалено не будет (читатель с таким номером не присутствует в таблице). Поэтому данный код неверен. Перепишем его немного по-другому:

```
$result_id=mysql_query("delete from reader where id_reader=35");
if(!$result_id)
  {
    echo "Ошибка запроса";
  }
else if(mysql_affected_rows()<1)
       {
         echo "Запись с номером 35 не найдена";
       }
     else
       {
         echo "Читатель с номером 35 удален";
       }
```

Здесь использована функция `mysql_affected_rows()`, которая возвращает количество обработанных записей.

Иногда начинающие программисты при работе с `mysql_query()` могут полагать, что возвращаемое значение является непосредственно данными или что это счетчик выбранных строк. И то и другое неверно.

Получение данных после выполнения запроса `SELECT` производится при помощи функции `mysql_fetch_row()` по идентификатору результата.

Есть и другие функции, получающие результат выборки *(см. приложение 2)*. В случае если возникла ошибка, и функции был передан неверный идентификатор, вы увидите на экране примерно следующее:

```
Warning: mysql_num_rows(): supplied argument is not a valid MySQL result
resource in path/script_name.php on line n.
```

Можно проверять идентификатор, прежде чем пытаться получить данные:

```
$result_id=mysql_query("select * from reader");
if(!$result_id)
  {
    echo "Ошибка выполнения запроса";
  }
else
  {
    $num=mysql_num_rows($result_id);
    echo "Количество читателей: $num";
  }
```

Ошибка выполнения запроса может возникнуть, если:

- запрос содержит синтаксическую ошибку;
- выполняется бессмысленный запрос (например, попытка выборки данных из несуществующего поля);

- вы не имеете достаточных прав на выполнение данного запроса;
- по какой-то причине недоступен сервер MySQL.

Для подавления ошибок в различных функциях вы можете использовать символ `'@'`. Например:

```
$connect_id=@mysql_connect("host", "login", "password");
```

Кроме этого есть возможность включения/выключения сообщений об ошибках с помощью функции `error_reporting()`, позволяющей настроить обработку ошибок. Существует несколько уровней обработки ошибок:

- `E_ALL` — все предупреждения и ошибки;
- `E_ERROR` — ошибки обычных функций;
- `E_WARNING` — предупреждения;
- `E_PARSE` — ошибки синтаксиса;
- `E_NOTICE` — замечания;
- `E_CORE_ERROR` — ошибки обработчика;
- `E_CORE_WARNING` — предупреждения обработчика;
- `E_COMPILE_ERROR` — ошибки времени трансляции;
- `E_COMPILE_WARNING` — предупреждения времени трансляции;
- `E_USER_ERROR` — ошибки, сгенерированные пользователем;
- `E_USER_WARNING` — предупреждения, сгенерированные пользователем;
- `E_USER_NOTICE` — замечания, сгенерированные пользователем.

Для подавления вывода ошибок можно вызвать функцию `error_reporting()` так:

```
error_reporting(E_PARSE | E_NOTICE);
```

Отключать ошибки синтаксического анализатора не стоит, т. к. это усложнит исправление ваших SQL-запросов. А предупреждения `E_NOTICE` в большинстве случаев можно игнорировать.

Для просмотра ошибки предназначена функция `mysql_error()`.

По окончании работы сценария вы можете очистить результирующий набор, вызвав функцию `mysql_free_result()`:

```
mysql_free_result($result_id);
```

На самом деле, PHP самостоятельно очищает память после завершения работы сценария. То есть использовать эту функцию целесообразно только при работе с большим количеством запросов или при запросах, возвращающих большое количество данных.

Давайте разберем еще пару сценариев.

Создадим интерфейс для добавления и просмотра читателей, зарегистрированных в библиотеке. Откройте текстовый редактор, наберите текст листинга 16.1 и сохраните его в файле с именем READER_LIST.PHP в корневом каталоге (все скрипты, описанные в этой книге, вы сможете найти на сопроводительном компакт-диске книги).

Листинг 16.1

```
<html>
<head>
<title>Список читателей</title>
</head>
<body>
<?php
#Подключаем файл для соединения с сервером и выбора БД
require "dbconnect.php";
#Проверяем отправку формы и начинаем обработку данных
if(isset($_POST['send']))
{
#Формируем запрос на добавление записи в таблицу "address"
$query1="insert into address
  values('', '".$_POST['addr']."','".$_POST['phone']."')";
$res_id1=mysql_query($query1);
/* Выбираем номер адреса, который мы добавили. Он понадобится нам
   для формирования следующего запроса */
$query2="select max(id_addr) from address";
$res_id2=mysql_query($query2);
#получаем номер адреса в масиив $addr_mas
$addr_mas=mysql_fetch_row($res_id2);
/* Теперь номер адреса известен, нам остается лишь добавить запись
   в таблицу reader */
$query3="insert into reader
  values('', '".$_POST['name']."','".$addr_mas[0]."')";
$res_id3=mysql_query($query3);
}
#Выводим данные по читателям
echo "<table border='1'>\n";
echo "<tr><th>Фамилия</th><th>Адрес</th><th>Телефон</th></tr>\n";
#Посылаем запрос на выборку данных
$res_id=mysql_query("select reader, address, phone from reader, address
  where reader.id_addr=address.id_addr ");
```

```
#Формируем строки таблицы
while($reader_mas=mysql_fetch_row($res_id))
{
echo "<tr>
<td>$reader_mas[0]</td>
<td>$reader_mas[1]</td>
<td>$reader_mas[2]</td>
</tr>\n";
}
?>
</table>
<form method="post">
Имя читателя<br>
<input type="text" name="name"><br>
Адрес читателя<br>
<input type="text" name="addr"><br>
Телефон<br>
<input type="text" name="phone"><br>
<input type="submit" name="send" value="Добавить">
</form>
</body>
</html>
```

Откройте браузер и запустите сценарий, введя в адресной строке URL **`http://localhost/reader_list.php`**.

После выполнения сценария вы увидите таблицу с данными по читателям и форму для добавления новых читателей (рис. 16.1).

Введите в поля формы соответствующие данные (например, **Фамилия** = **`Иванов`**, **Адрес** = **`ул. Ткачей, 21/7`**, **Телефон** = **`540-40-80`**) и нажмите кнопку **Добавить**. Вы увидите, что после выполнения сценария в таблице появилась новая запись (рис. 16.2).

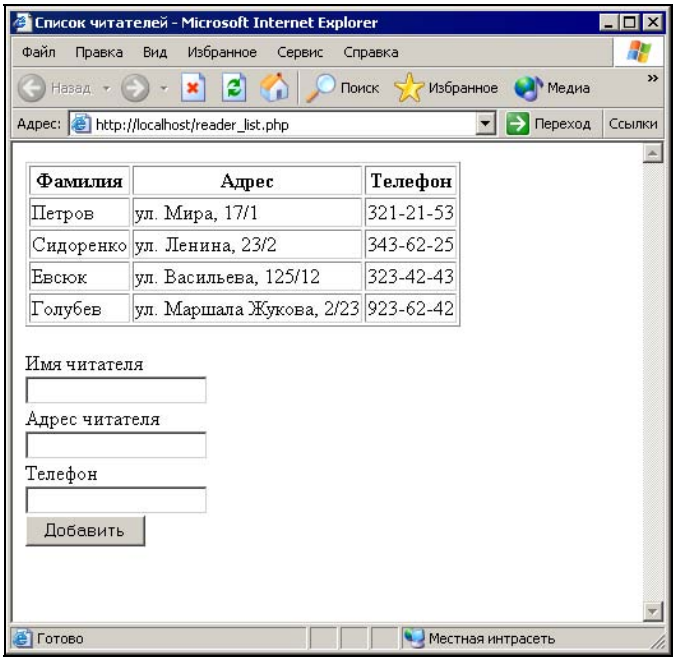


Рис. 16.1. Список читателей

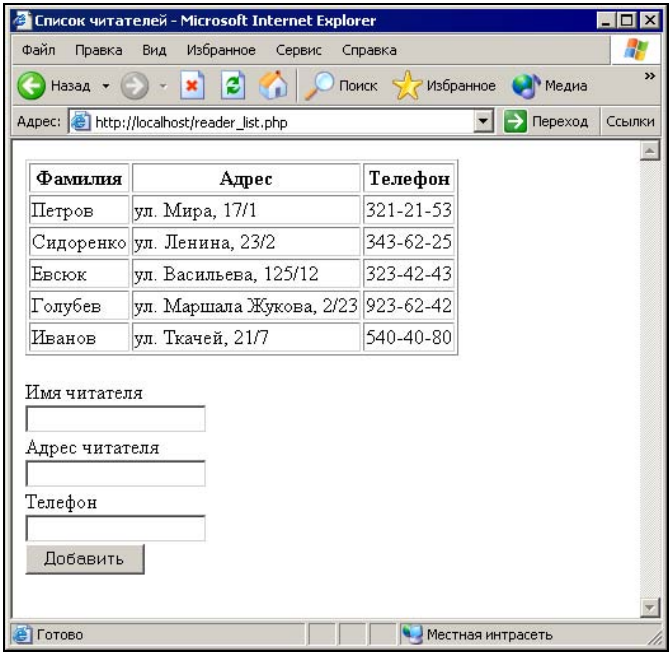


Рис. 16.2. Результат выполнения сценария

УРОК 17

Получение данных из формы

Данные из формы мы получали, используя конструкцию `$_POST['имя_поля']`.

`$_POST` — это глобальный ассоциативный массив (хэш), в который помещаются все значения, переданные из формы с помощью метода `POST`. При использовании метода `GET` нам доступен массив `$_GET`.

Можно использовать и альтернативный способ приема данных формы. Для этого в файле настройки PHP (PHP.INI) нужно установить `register_globals=On`. Это позволит регистрировать входные параметры непосредственно в переменных. Например, если какое-либо поле формы имело имя `address`, то будет зарегистрирована переменная `$address`, содержащая значение этого поля. Это значительно удобнее первого варианта. Но данная настройка несет определенный *риск для системы безопасности*, т. к. получается, что клиент может устанавливать переменные в вашем сценарии. Поэтому разработчики PHP рекомендуют отключать эту настройку: `register_globals=Off`.

Соответственно, вся обработка данных начинается при условии, что установлен параметр `'send'`:

```
if(isset($_POST['send']))
```

Это означает, что пользователь нажал кнопку **Добавить** и отправил данные. На самом деле проверка по имени кнопки отправки (`submit`) не очень надежна. Иногда, заполнив какую-либо форму, пользователь нажимает клавишу <Enter> вместо того, чтобы щелкнуть мышью на кнопке. В этом случае имя кнопки передано не будет и, соответственно, сценарий не станет обрабатывать данные, т. к. описанное нами условие ложно. Для избежания этой ситуации вы можете использовать для проверки передачи данных скрытое поле:

```
<input type="hidden" name="имя_поля" value="значение_поля">
```

То есть форму можно описать примерно так:

```
<form method="POST">
<input type="text" name="name">
<input type="hidden" name="send" value="yes">
<input type="submit" value="Ok">
</form>
```

Это поле не выводится браузером, т. е. пользователь не подозревает о его существовании, но данные из этого поля (*имя_поля = значение_поля*) передаются вне зависимости от того, каким образом была отправлена форма. Имя и значение этого поля вы устанавливаете произвольно.

Вы также можете передавать параметры сценарию через URL. То есть можно сформировать гипертекстовые ссылки на странице и, добавив в них параметры, указывать сценарию, что он должен выполнить. PHP преобразует параметры в переменные с именами параметров. Например, можно передать параметры так:

http://www.mywebsite.ru/script.php?parameter=value

Если нужно передать несколько параметров, разделите их символом '&'.

http://www.mywebsite.ru/script.php?parameter1=value&parameter2=value

Давайте рассмотрим простой пример сценария, который будет выполнять различные действия в зависимости от того, какую ссылку нажмет пользователь. Откройте текстовый редактор, наберите текст листинга 17.1 и сохраните его в файле с именем TEST_LINK.PHP в корневом каталоге сервера.

Листинг 17.1

```
<html>
<head>
<title>Передача параметров через URL</title>
</head>
<body>
<!-- Выводим гиперссылки и передаем параметру различные значения -->
<a href="test_link.php?action=1">Ссылка 1</a>
<br>
<a href="test_link.php?action=2">Ссылка 2</a>
<br>
<?php
if($action==1)
  {
    echo "Вы нажали ссылку 1";
  }
```

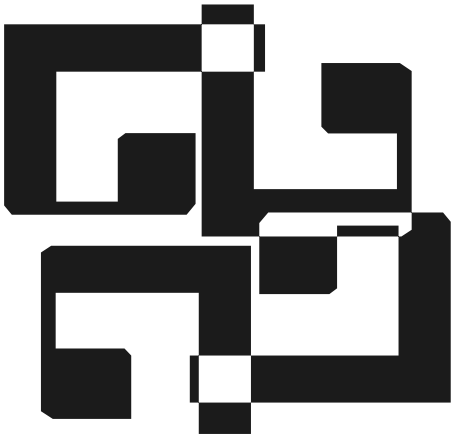
```
if($action==2)
  {
    echo "Вы нажали ссылку 2";
  }
?>
</body>
</html>
```

В итоге сценарий будет выводить либо первое, либо второе сообщение в зависимости от нажатой ссылки.

В скрипте, добавляющем нового читателя, нет никаких дополнительных проверок данных, приходящих из формы (предполагается, что человек, работающий с этим web-интерфейсом, не будет делать ничего лишнего). Но если требуется, вы всегда можете проверить данные, приходящие из формы, и не допустить добавление неверных записей в таблицы.

ЗАДАНИЯ ДЛЯ САМОКОНТРОЛЯ

1. Напишите сценарий для просмотра информации о книгах (названия и количество экземпляров).
2. Создайте сценарий для добавления новых разделов (жанров) в библиотеку.



ПРИЛОЖЕНИЯ

Приложение 1. Список зарезервированных слов MySQL

Приложение 2. Интерфейс PHP API для MySQL

Приложение 3. Ответы на вопросы и задания для самоконтроля

Приложение 4. Описание компакт-диска

ПРИЛОЖЕНИЕ 1

Список зарезервированных слов MySQL

Здесь приведены зарезервированные слова, которые нежелательно использовать при именовании таблиц MySQL и их полей.

ADD
ALL
ALTER
ANALYZE
AND
AS
ASC
ASENSITIVE
BEFORE
BETWEEN
BIGINT
BINARY
BLOB
BOTH
BY
CALL
CASCADE
CASE
CHANGE
CHAR
CHARACTER
CHECK
COLLATE
COLUMN
CONDITION
CONNECTION
CONSTRAINT
CONTINUE
CONVERT
CREATE
CROSS
CURRENT_DATE
CURRENT_TIME
CURRENT_TIMESTAMP
CURRENT_USER
CURSOR
DEFAULT
DATABASE
DATABASES
DAY_HOUR
DAY_MICROSECOND
DAY_MINUTE
DAY_SECOND
DEC
DECIMAL
DECLARE
DELAYED
DELETE
DESC
DESCRIBE
DETERMINISTIC
DISTINCT
DISTINCTROW
DIV
DOUBLE
DROP
DUAL
EACH
ELSE
ELSEIF
ENCLOSED
ESCAPED
EXISTS
EXIT
EXPLANE
FALSE
FETCH
FLOAT
FOR
FOR
FORCE
FOREIGN
FULLTEXT
GOTO
GRANT
GROUP
HAVING
HIGH_PRIORITY
HOUR_MICROSECOND
HOUR_MINUTE
HOUR_SECOND

IF
IGNORE
IN
INDEX
INFILE
INNER
INOUT
INSENSITIVE
INSERT
INT
INTEGER
INTERVAL
INTO
IS
ITERATE
JOIN
KEY
KEYS
KILL
LEADING
LEAVE
LEFT
LIKE
LIMIT
LINES
LOAD
LOCALTIME
LOCALTIMESTAMP
LOCK
LONG
LONGBLOB
LONGTEXT
LOOP
LOW_PRIORITY
MATCH
MEDIUMBLOB
MEDIUMINT
MEDIUMTEXT
MIDDLEINT
MINUTE_MICROSECOND
MINUTE_SECOND
MOD
MODIFIES
NATURAL
NO_WRITE_TO_BINLOG
NOT
NULL
NUMERIC
ON
OPTIMIZE
OPTION
OPTIONALLY
OR
ORDER
OUT
OUTER
OUTFILE
PRECISION
PRIMARY
PROCEDURE
PURGE
READ
READS
REAL
REFERENCES
REGEXP
RENAME
REPEAT
REPLACE
REQUIRE
RESTRICT
RETURN
REVOKE
RIGHT
RLIKE
SCHEMA
SCHEMAS
SECOND_MICROSECOND
SELECT
SENSITIVE
SEPARATOR
SET
SHOW
SMALLINT
SONAME
SPATIAL
SPECIFIC
SQL
SQL_BIG_RESULT
SQL_CALC_FOUND_ROWS
SQL_SMALL_RESULT
SQLEXCEPTION
SQLSTATE
SQLWARNING
SSL
STARTING
STRAIGHT_JOIN
TABLE
TERMINATED
THEN
TINYBLOB
TINYINT
TINYTEXT
TO
TRAILING
TRIGGER
TRUE
UNDO
UNION
UNIQUE
UNLOCK
UNSIGNED
UPDATE
USAGE
USE
USING
UTC_DATE
UTC_TIME
UTC_TIMESTAMP
VALUES
VARBINARY
VARCHAR
VARCHARACTER
VARYING
WHEN
WHERE
WHILE
WITH
WRITE
XOR
YEAR_MONTH
ZEROFILL

ПРИЛОЖЕНИЕ 2

Интерфейс PHP API для MySQL

В этом приложении приведены функции PHP API. При описании синтаксиса вызова многих функций используется идентификатор соединения (параметр *connect_id*), определяющий соединение с сервером MySQL, а также идентификатор результата (параметр *result_id*), обычно возвращаемый функцией `mysql_query()`. Если идентификатор соединения не определен, используется последнее открытое соединение. Все необязательные аргументы функций заключены в квадратные скобки (`[]`).

mysql_affected_rows

Функция `mysql_affected_rows()` возвращает количество строк, использованных предыдущими MySQL-операциями. Синтаксис вызова:

```
int mysql_affected_rows([resource connect_id]);
```

Функция возвращает количество строк, использованных запросами `INSERT`, `UPDATE`, `REPLACE` или `DELETE` на сервере, связываемом с определенным идентификатором соединения. Если последним запросом был запрос `DELETE` без условия `WHERE`, все записи будут удалены из таблицы, но эта функция возвратит нуль.

Эта функция не действует при запросе `SELECT` (она действует только при запросах, изменяющих записи). Чтобы получить количество строк возвращенных при выполнении запроса `SELECT`, используйте функцию `mysql_num_rows()`.

mysql_close

Функция `mysql_close()` закрывает MySQL-соединение. Синтаксис вызова:

```
bool mysql_close([resource connect_id]);
```

Возвращает значение: `true` — при успешном завершении, `false` — при ошибке.

Функция `mysql_close()` закрывает связь с базой данных MySQL, ассоциированной с определенным идентификатором соединения. Помните, что это не всегда необходимо, т. к. непостоянные открытые соединения автоматически закрываются в конце выполнения сценария.

Функция `mysql_close()` не закроет устойчивые соединения, сгенерированные функцией `mysql_pconnect()`.

mysql_connect

Функция `mysql_connect` открывает соединение с MySQL-сервером. Синтаксис вызова:

```
resource mysql_connect([string hostname [, string username
  [, string password [, bool new_connect]]]]);
```

Функция возвращает правильный идентификатор соединения MySQL при успешном выполнении или значение `false` при ошибке.

Функция `mysql_connect()` устанавливает соединение с MySQL-сервером. Все аргументы функции не являются обязательными, и если они пропущены, то устанавливаются по умолчанию — `'localhost'`, имя пользователя, который владеет процессом, пустой пароль. Аргумент *`hostname`* также может содержать номер порта (подобно `"localhost:3306"`).

В случае повторного вызова функции `mysql_connect()` с теми же аргументами никакое новое соединение не установится — вместо этого будет возвращен идентификатор уже открытого соединения. Параметр *`new_connect`*, установленный в значение `true`, меняет это поведение и заставляет функцию всегда открывать новое соединение.

Как только выполнение сценария закончится, связь с сервером закроется, если оно не было явно закрыто более ранним вызовом `mysql_close()`.

Пример:

```
<?php
    $id_con = mysql_connect("localhost", "mysql_user", "mysql_password")
        or die("Невозможно установить соединение");
    print ("Соединение установлено");
    mysql_close($id_con);
?>
```

mysql_create_db

Функция `mysql_create_db()` создает базу данных MySQL. Синтаксис вызова:

```
bool mysql_create_db(string database_name [, resource connect_id]);
```

Функция `mysql_create_db()` пытается создавать новую базу данных на сервере, связанном с определенным идентификатором соединения.

Возвращает значение `true` при успехе и `false` при неудаче.

mysql_data_seek

Функция `mysql_data_seek()` перемещает внутренний указатель результата. Синтаксис вызова:

```
int mysql_data_seek(resource result_id, int row_number);
```

Возвращает значение: `true` — при успешном выполнении, `false` — при ошибке.

Функция `mysql_data_seek()` перемещает внутренний указатель для заданного результирующего набора на заданную строку (каждый возвращаемый запросом `SELECT` результирующий набор содержит специальный курсор, указывающий на строку, которая будет возвращена при следующем вызове функций выборки строк).

mysql_db_query

Функция `mysql_db_query` посылает MySQL-запрос. Синтаксис вызова:

```
resource mysql_db_query(string database_name, string query
  [, resource connect_id]);
```

Возвращает правильный идентификатор результата MySQL в результате запроса или значение `false` при ошибке.

Функция `mysql_db_query()` выбирает базу данных и выполняет запрос к ней.

Примечание

Выбирать БД при каждом запросе неэффективно, вместо этого рекомендую использовать функцию `mysql_query()`.

mysql_drop_db

Функция `mysql_drop_db()` удаляет базу данных MySQL. Синтаксис вызова:

```
bool mysql_drop_db(string database_name [, resource connect_id]);
```

Возвращает значение: `true` — при успешном выполнении, `false` — при неудачном завершении.

mysql_errno

Функция `mysql_errno()` возвращает номер сообщения об ошибке предыдущей операции MySQL. Синтаксис вызова:

```
int mysql_errno([resource connect_id]);
```

mysql_error

Функция `mysql_error` возвращает текст сообщения об ошибке предыдущей операции MySQL. Синтаксис вызова:

```
string mysql_error([resource connect_id]);
```

mysql_escape_string

Функция `mysql_escape_string` мнемонизирует строку для использования в функции `mysql_query()`. Синтаксис вызова:

```
string mysql_escape_string(string str);
```

Выделяет любые кавычки и другие специальные символы в аргументе *str*, предваряя их обратным слэшем (\).

mysql_fetch_array

Функция `mysql_fetch_array()` возвращает результирующий набор как ассоциативный массив или числовой массив. Синтаксис вызова:

```
array mysql_fetch_array(resource result_id [, resource result_type]);
```

Возвращает результат строки запроса как массив или значение `false` при отсутствии результата.

Функция `mysql_fetch_array()` является расширенной версией функции `mysql_fetch_row()`. Кроме того, что она сохраняет данные в пронумерованных элементах результатного массива, а также сохраняет ассоциативную связь, используя имена полей как ключи. Если два или более столбцов имеют одинаковые имена, то связку ключ/значение получает последний из них. Чтобы получить доступ к другим полям с таким же именем, вы должны использовать индексы или задать псевдонимы (`alias`) для этих столбцов.

Параметр *result_type* может принимать следующие значения:

- MYSQL_ASSOC — возвращаются значения только по индексам имен;
- MYSQL_NUM — возвращаются значения только по числовым индексам;
- MYSQL_BOTH — возвращаются значения по индексам обоих типов.

mysql_fetch_assoc

Функция mysql_fetch_assoc() возвращает результирующий набор как ассоциативный массив. Синтаксис вызова:

```
array mysql_fetch_assoc(resource result_id);
```

Функция возвращает следующую строку заданного результирующего набора в виде ассоциативного массива или значение false, если строк больше нет.

mysql_fetch_field

Функция mysql_fetch_field() получает информацию о столбце из результата и возвращает ее как объект. Синтаксис вызова:

```
object mysql_fetch_field(resource result_id [, int field_number]);
```

Функцию mysql_fetch_field() можно использовать для получения информации о поле в определенном результате запроса. Если параметры поля не указаны, то будет запрошено поле, которое еще не было запрошено функцией mysql_fetch_field().

Свойства объекта:

- name — имя поля;
- table — имя таблицы, содержащей поле;
- max_length — длина наибольшего значения поля;
- not_null — 1, если поле не может принимать нулевые значения;
- primary_key — 1, если поле является первичным ключом;
- unique_key — 1, если поле является полем с уникальными значениями (имеет атрибут UNIQUE);
- multiple_key — 1, если поле не является полем с уникальными значениями (не имеет атрибута UNIQUE);
- numeric — 1, если поле числовое;
- blob — 1, если поле имее тип BLOB;
- type — тип поля;

- `unsigned` — 1, если поле беззнаковое;
- `zerofill` — 1, если поле имеет атрибут `ZEROFILL`.

mysql_fetch_lengths

Функция `mysql_fetch_lengths` получает максимальный размер данных для каждого выходного значения. Синтаксис вызова:

```
array mysql_fetch_lengths(resource result_id);
```

Возвращает массив (нумерация элементов — с нуля), содержащий размеры полей последней строки, выбранной функцией `mysql_fetch_row()`, или значение `false` при ошибке.

mysql_fetch_object

Функция `mysql_fetch_object()` получает строку результата как объект. Синтаксис вызова:

```
object mysql_fetch_object(resource result_id [, resource result_type]);
```

Возвращает объект со свойствами, который соответствует полученной строке, или значение `false`, если нет результата.

Функция `mysql_fetch_object()` подобна функции `mysql_fetch_array()` с одним отличием — вместо массива возвращается объект. Это означает, что вы можете иметь доступ к данным только по именам полей, а не по их параметрам (обращение по индексным числам в данном случае является неверным).

Пример:

```
<?php
$id_con=mysql_connect($host,$user,$password);
$result_id = mysql_db_query("database_name","select * from table");
while($row = mysql_fetch_object($result)) {
    echo $row->last_name;
    echo $row->first_name;
}
mysql_free_result($result_id);
?>
```

mysql_fetch_row

Функция `mysql_fetch_row()` получает строку результата как пронумерованный массив. Синтаксис вызова:

```
array mysql_fetch_row(resource result_id);
```

Возвращает массив значений выбираемых полей или значение `false`, если нет результата.

Функция `mysql_fetch_row()` выбирает поле данных из результата и связывает его с определенным идентификатором результата (индексом элемента массива). Запрос возвращается как массив. Доступ к значениям можно получать через элементы массива.

Последующий вызов функции `mysql_fetch_row()` должен возвратить следующую строку или значение `false`, если полей больше нет.

mysql_field_flags

Функция `mysql_field_flags()` получает флаги, ассоциированные с полем. Синтаксис вызова:

```
string mysql_fields_flags(resource result_id, int field_number);
```

Возвращает флаги специфицированного поля, разделенные пробелами (табл. П2.1).

Таблица П2.1. *Значения функции* `mysql_field_flags()`

Свойство	Значение
`auto_increment`	Поле имеет атрибут `auto_increment`
`binary`	Поле имеет атрибут `binary`
`blob`	Поле имеет тип `Blob` или `Text`
`enum`	Поле имеет тип `Enum`
`multiple_key`	Поле является частью неуникального индекса
`not_null`	Поле имеет атрибут `not_null`
`primary_key`	Поле является частью первичного ключа
`timestamp`	Поле имеет тип `Timestamp`
`unique_key`	Поле является частью индекса `unique`
`unsigned`	Поле имеет атрибут `unsigned`
`zerofill`	Поле имеет атрибут `zerofill`

Для разделения флагов можно воспользоваться функцией `explode()`. Аргумент *`field_number`* задает индекс (номер) поля, начиная с 0.

mysql_field_len

Функция `mysql_field_len()` возвращает длину поля. Синтаксис вызова:

```
int mysql_field_len(resource result_id, int field_number);
```

Функция возвращает максимальную возможную длину заданного поля. Аргумент *`field_number`* задает индекс (номер) поля, начиная с 0.

mysql_field_name

Функция `mysql_field_name()` получает имя определенного поля в результате. Синтаксис вызова:

```
string mysql_field_name(resource result_id, int field_number);
```

Функция `mysql_field_name()` возвращает имя указанного поля. Аргументами функции являются идентификатор результата и индекс поля, т. е. функция `mysql_field_name($result,2)` возвратит имя третьей области в результате, связанном с идентификатором результата (`$result`).

mysql_field_seek

Функция `mysql_field_seek()` устанавливает указатель запроса в определенное поле. Синтаксис вызова:

```
int mysql_field_seek(resource result_id, int field_number);
```

Функция осуществляет поиск в определенном поле. Если при следующем вызове функции `mysql_fetch_field()` поле не указано (отсутствует значение второго аргумента), то должно быть возвращено именно это поле. Возвращает значение `true`, если поиск заданного поля завершился удачно, и значение `false` при неудаче. Аргумент *`field_number`* задает индекс (номер) поля, начиная с 0.

mysql_field_table

Функция `mysql_field_table()` получает имя таблицы, в которой находится указанное поле. Синтаксис вызова:

```
string mysql_field_table(resource result_id, int field_number);
```

Получает имя таблицы поля. Аргумент *`field_number`* задает индекс (номер) поля, начиная с 0.

mysql_field_type

Функция `mysql_field_type()` получает тип указанного поля в результате. Синтаксис вызова:

```
string mysql_field_type(resource result_id, int field_number);
```

Функция `mysql_field_type()` подобна функции `mysql_field_name()`. Аргументы идентичны, но возвращается тип поля (`'int'`, `'real'`, `'string'`, `'blob'` и т. п.).

mysql_free_result

Функция `mysql_free_result()` освобождает память результата. Синтаксис вызова:

```
int mysql_free_result(resource result_id);
```

Функцию `mysql_free_result()` нужно использовать, только если вы беспокоитесь о слишком большом объеме памяти, занимаемой во время работы вашего скрипта. Вся используемая результатом память для определенного идентификатора результата будет освобождена автоматически.

mysql_list_dbs

Функция `mysql_list_dbs()` возвращает идентификатор результата, содержащего список имен баз данных, которые сервер относит к заданному соединению. Синтаксис вызова:

```
resource mysql_list_dbs([resource connect_id]);
```

mysql_list_fields

Функция `mysql_list_fields()` возвращает идентификатор результата со списком полей таблицы. Синтаксис вызова:

```
resource mysql_list_fields(string db_name, string table_name
  [, resource connect_id]);
```

Обязательные аргументы — имя базы данных и имя таблицы. Идентификатор результата является положительным целым. Функция возвращает –1, если происходит ошибка. Строка описания ошибки будет помещена в переменную `$phperrmsg`, и если функция не была вызвана как `@mysql()`, то затем также будет выведено это описание ошибки.

mysql_list_processes

Функция `mysql_list_processes()` возвращает идентификатор результата, содержащий информацию о текущих процессах, работающих на сервере. Синтаксис вызова:

```
resource mysql_list_processes([, resource connect_id]);
```

mysql_list_tables

Функция `mysql_list_tables()` возвращает идентификатор результата со списком имен таблиц заданной БД. Синтаксис вызова:

```
resource mysql_list_tables(string db_name, [, resource connect_id]);
```

Функция `mysql_list_tables()` принимает имя базы данных и указатель результата, подобно функции `mysql_db_query()`, и извлекает фактические имена таблиц.

mysql_num_fields

Функция `mysql_num_fields()` получает количество полей в указанном результате. Синтаксис вызова:

```
int mysql_num_fields(resource result_id);
```

mysql_num_rows

Функция `mysql_num_rows()` получает количество строк в указанном результате. Синтаксис вызова:

```
int mysql_num_rows(resource result_id);
```

mysql_pconnect

Функция `mysql_pconnect()` открывает устойчивое соединение с MySQL-сервером. Синтаксис вызова:

```
int mysql_pconnect([string hostname [, string username
  [, string password]]]);
```

Функция `mysql_pconnect()` возвращает правильный MySQL-идентификатор устойчивого соединения при успешном выполнении или значение `false` при ошибке.

Функция `mysql_pconnect()` очень похожа на функцию `mysql_connect()` с двумя важными отличиями:

- при соединении функция должна сначала попытаться найти уже открытую устойчивую связь с тем же хостом, именем пользователя и паролем. Если она обнаруживается, возвратится ее идентификатор (вместо открытия нового соединения);
- соединение с сервером SQL не закроется, когда закончится выполнение сценария — соединение останется открытым для последующего использования (функция `mysql_close()` не закрывает соединения, установленные функцией `mysql_pconnect()`).

Использование этой функции считается более эффективным методом соединения.

mysql_ping

Функция `mysql_ping()` посылает сигнал серверу для соединения. Синтаксис вызова:

```
bool mysql_ping([resource connect_id]);
```

Функция `mysql_ping()` устанавливает повторное соединение с сервером, если время соединения вышло. Можно использовать ее в скриптах, которые долгое время не запускаются.

Функция возвращает значение `true`, если соединение активно или было вновь восстановлено, и значение `false` в противном случае.

mysql_query

Функция `mysql_query()` отправляет SQL-запрос. Синтаксис вызова:

```
int mysql_query(string query [, resource connect_id]);
```

Функция `mysql_query()` посылает запрос к базе данных для заданного соединения. Если аргумент *connect_id* не указан, используется последнее открытое соединение. Если соединение не открыто, функция пытается установить соединение (как если бы была вызвана функция `mysql_connect()`) и использует его.

Эта функция возвращает значение `true` или `false`, чтобы показать успешность выполнения запросов `UPDATE`, `INSERT` и `DELETE`. Успешным считается запрос, выполненный без единой ошибки. Для запроса `SELECT` она возвращает новый идентификатор результата.

mysql_result

Функция `mysql_result()` получает данные результата. Синтаксис вызова:

```
mixed mysql_result(resource result_id, int row, mixed field);
```

Функция `mysql_result()` возвращает содержимое одной ячейки из результата с указанным идентификатором. Аргумент *`field`* может быть задан порядковым номером, или именем поля, или параметром типа *`имя_таблицы.имя_поля`*.

Работая с большими результатами, вы должны предусматривать использование одной из функций, выбирающих целую строку. За счет того что эти функции возвращают содержимое большого числа ячеек за один вызов функции, они выполняются значительно быстрее, чем функция `mysql_result()`. Также имейте в виду, что если аргумент *`field`* задан как числовой параметр, функция `mysql_result()` выполняется значительно быстрее, чем если он задан как *`имя_поля`* или как аргумент типа *`имя_таблицы.имя_поля`*.

Вызов функции `mysql_result()` не следует смешивать с вызовами других функций, имеющих дело с установленным результатом.

mysql_select_db

Функция `mysql_select_db()` выбирает базу данных MySQL. Синтаксис вызова:

```
int mysql_select_db(string db_name [, resource connect_id]);
```

Возвращает значение `true` при успешном выполнении и значение `false` при ошибке.

Функция `mysql_select_db()` определяет текущую активную базу данных на сервере, связанную с указанным идентификатором соединения. Если идентификатор соединения не определен, то принимается последнее открытое соединение. Каждый последующий вызов функции `mysql_query()` относится к активной базе данных.

Информационные функции

В этот раздел вынесены функции, возвращающие различную информацию:

- `string mysql_get_client_info(void)` — возвращает версию клиентской библиотеки;
- `string mysql_get_host_info([resource connect_id])` — возвращает описание заданного соединения;

- `int mysql_get_proto_info([resource connect_id])` — возвращает версию протокола;
- `string mysql_get_server_info([resource connect_id])` — возвращает версию сервера;
- `string mysql_info()` — возвращает подробную информацию о последнем запросе;
- `string mysql_stat([resource connect_id])` — возвращает информацию о статусе сервера;
- `int mysql_thread_id([resource connect_id])` — возвращает идентификатор процесса, связанного с данным идентификатором соединения;
- `string mysql_character_set_name([resource connect_id])` — возвращает название кодировки для текущего соединения.

ПРИЛОЖЕНИЕ 3

Ответы на вопросы и задания для самоконтроля

В данном приложении даны ответы на все контрольные вопросы и задания, которые есть в книге.

Урок 3

ВОПРОС 1. Какие четыре базовых текстовых типа поддерживает MySQL?

Ответ: `CHAR`, `VARCHAR`, `TEXT`, `BLOB`.

ВОПРОС 2. Сколько памяти требуется для хранения значения поля типа `SMALLINT`?

Ответ: 2 байта.

ВОПРОС 3. Каков диапазон допустимых значений для типа `YEAR`?

Ответ: От 1901 до 2155.

ВОПРОС 4. Какие типы таблиц MySQL вы можете использовать?

Ответ: `ISAM`, `MyISAM`, `HEAP`, `BDB`, `InnoDB`, `MERGE`.

ВОПРОС 5. Какой тип определен по умолчанию?

Ответ: `MyISAM`.

ВОПРОС 6. Какой тип можно использовать для временного хранения данных?

Ответ: `HEAP`.

Урок 7

ВОПРОС 1. Какие операторы можно использовать для удаления индекса?

Ответ: Удалить индекс можно с помощью оператора `DROP INDEX` или `ALTER TABLE`.

ВОПРОС 2. Какие типы индексов можно создавать в MySQL?

Ответ: В MySQL можно создавать следующие типы индексов:

- обычный индекс;
- уникальный индекс;
- индекс типа `FULLTEXT`.

Урок 8

ВОПРОС 1. Какая опция оператора `ALTER TABLE` позволяет удалить поле таблицы?

Ответ: Опция `DROP`.

ВОПРОС 2. Какие ключевые слова используются для указания позиции добавляемого поля?

Ответ: Ключевые слова `FIRST` и `AFTER`.

Урок 17

ЗАДАНИЕ 1. Напишите сценарий для просмотра информации о книгах (названия и количество экземпляров).

Возможный сценарий приведен в листинге П3.1.

Листинг П3.1. Пример решения задания 1

```
<?php
require "dbconnect.php";
?>
<html>
<head>
<title>Книги</title>
</head>
<body>
<table border width="100%">
<tr><td><b>Название книги</b></td><td><b>Количество экземпляров</b>
</td></tr>
<?php
$res_id=mysql_query("select title, count(id_unit) from book, unit where
  book.id_book=unit.id_book group by unit.id_book");
while($arr_book=mysql_fetch_row($res_id))
  {
```

```
    echo "<tr><td>$arr_book[0]</td><td>$arr_book[1]</td></tr>";
  }
?>
</table>
</body>
</html>
```

ЗАДАНИЕ 2. Создайте сценарий для добавления новых секций (жанров) в библиотеку.

Возможный сценарий приведен в листинге П3.2.

Листинг П3.2. Пример решения задания 2

```
<?php
require "dbconnect.php";

if(isset($_POST['new_section']))
  {
    $query="insert into section values('','".$_POST['new_section']."')";
      if(mysql_query($query))
        {
          echo "Запись добавлена";
        }
      else
        {
          echo mysql_error();
        }
}
?>
<html>
<head>
<title>Добавление секций</title>
</head>
<body>
<form method="POST">
<select>
<?php
$res_id=mysql_query("select section from section");
while($arr_auth=mysql_fetch_row($res_id))
  {
    echo "<option>$arr_auth[0]</option>";
  }
?>
```

```
</select>
<input type="text" name="new_section">
<input type="submit" name="add" value="Добавить">
</form>
</body>
</html>
```

ПРИЛОЖЕНИЕ 4

Описание компакт-диска

Папка	Содержание
БД Library	Содержит текстовые файлы со структурой учебной БД и данными для таблиц: • data_for_library.txt — данные для заполнения таблиц; • library.txt — набор операторов CREATE TABLE для создания всех таблиц учебной БД; • db_librzry.txt — пример дампа (от dump — сохранение) учебной БД
SOFT	Apache — web-сервер MySQL — дистрибутив для установки СУБД MySQL PHP — интерпретатор языка PHP
Скрипты к БД	Скрипты, описанные в книге

Предметный указатель

S

T

W

А

Б

В, Г

Д

З

И

К

Л

М

Н

О

П

Р

С

Т

У

Ф

Х

Ц, Ч

Ш, Э