

Лекция 1 Введение в веб-технологии

Лекция 2 Веб-дизайн и юзабилити.

Лекция 3 HTML и CSS.

Лекция 4 Язык PHP.

Лекция 5 СУБД Mysql

Роль СУБД в веб-технологиях (на примере mysql). Особенности, ограничения, администрирование. Сфера применения.

Базы данных: основные понятия

В жизни мы часто сталкиваемся с необходимостью хранить какую-либо информацию, а потому часто имеем дело и с базами данных. Например, мы используем записную книжку для хранения номеров телефонов своих друзей и планирования своего времени. Телефонная книга содержит информацию о людях, живущих в одном городе. Все это своего рода базы данных. Ну а раз это базы данных, то посмотрим, как в них хранятся данные. Например, телефонная книга представляет собой таблицу ([табл. 10.1](#)).

В этой таблице данные – это собственно номера телефонов, адреса и ФИО., т.е. строки «Иванов Иван Иванович», «32-43-12» и т.п., а названия столбцов этой таблицы, т.е. строки «ФИО», «Номер телефона» и «Адрес» задают смысл этих данных, их семантику.

Таблица 10.1. Пример базы данных: телефонная книга

ФИО	Номер телефона	Адрес
Иванов Иван Иванович	32-43-12	ул. Ленина, 12, 43
Ильин Федор Иванович	32-32-34	пр. Маркса, 32, 45

Теперь представьте, что записей в этой таблице не две, а две тысячи, вы занимаетесь созданием этого справочника и где-то произошла ошибка (например, опечатка в адресе). Видимо, тяжело будет найти и исправить эту ошибку вручную. Нужно воспользоваться какими-то средствами автоматизации. Для управления большим количеством данных программисты (не без помощи математиков) придумали системы управления базами данных (СУБД). По сравнению с текстовыми базами данных электронные СУБД имеют огромное число преимуществ, от возможности быстрого поиска информации, взаимосвязи

данных между собой до использования этих данных в различных прикладных программах и одновременного доступа к данным нескольких пользователей.

Для точности дадим определение базы данных, предлагаемое Глоссарий.ру

База данных – это совокупность связанных данных, организованных по определенным правилам, предусматривающим общие принципы описания, хранения и манипулирования, независимая от прикладных программ. База данных является информационной моделью предметной области. Обращение к базам данных осуществляется с помощью системы управления базами данных (**СУБД**). СУБД обеспечивает поддержку создания баз данных, централизованного управления и организации доступа к ним различных пользователей.

Итак, мы пришли к выводу, что хранить данные независимо от программ, так, что они связаны между собой и организованы по определенным правилам, целесообразно. Но вопрос, как хранить данные, по каким правилам они должны быть организованы, остался открытым. Способов существует множество (кстати, называются они моделями представления или хранения данных). Наиболее популярные – объектная и реляционная модели данных.

Автором **реляционной модели** считается Э. Кодд, который первым предложил использовать для обработки данных аппарат теории множеств (объединение, пересечение, разность, декартово произведение) и показал, что любое представление данных сводится к совокупности двумерных таблиц особого вида, известного в математике как отношение.

Таким образом, реляционная база данных представляет собой набор таблиц (точно таких же, как приведенная выше), связанных между собой. Строка в таблице соответствует сущности реального мира (в приведенном выше примере это информация о человеке).

Примеры реляционных СУБД: MySql, PostgreSQL.

В основу **объектной модели** положена концепция объектно-ориентированного программирования, в которой данные представляются в виде набора объектов и классов, связанных между собой родственными отношениями, а работа с объектами осуществляется с помощью скрытых (инкапсулированных) в них методов.

Примеры объектных СУБД: Cache, GemStone (от Servio Corporation), ONTOS (ONTOS).

В последнее время производители СУБД стремятся соединить два этих подхода и проповедуют объектно-реляционную модель представления данных. Примеры таких СУБД – IBM DB2 for Common Servers, Oracle8.

Поскольку мы собираемся работать с Mysql, то будем обсуждать аспекты работы только с реляционными базами данных. Нам осталось рассмотреть еще два важных понятия из этой области: ключи и индексирование, после чего мы сможем приступить к изучению языка запросов SQL.

Ключи

Для начала давайте подумаем над таким вопросом: какую информацию нужно дать о человеке, чтобы собеседник точно сказал, что это именно тот человек, сомнений быть не может, второго такого нет? Сообщить фамилию, очевидно, недостаточно, поскольку существуют однофамильцы. Если собеседник человек, то мы можем приблизительно объяснить, о ком речь, например вспомнить поступок, который совершил тот человек, или еще как-то. Компьютер же такого объяснения не поймет, ему нужны четкие правила, как определить, о ком идет речь. В системах управления базами данных для решения такой задачи ввели понятие первичного ключа.

Первичный ключ (primary key, PK) – минимальный набор полей, уникально идентифицирующий запись в таблице. Значит, первичный ключ – это в первую очередь набор полей таблицы, во-вторых, каждый набор значений этих полей должен определять единственную запись (строку) в таблице и, в-третьих, этот набор полей должен быть минимальным из всех обладающих таким же свойством. Поскольку первичный ключ определяет только одну уникальную запись, то никакие две записи таблицы не могут иметь одинаковых значений первичного ключа.

Например, в нашей таблице ([см. выше](#)) ФИО и адрес позволяют однозначно выделить запись о человеке. Если же говорить в общем, без связи с решаемой задачей, то такие знания не позволяют точно указать на единственного человека, поскольку существуют однофамильцы, живущие в разных городах по одному адресу. Все дело в границах, которые мы сами себе задаем. Если считаем, что знания ФИО, телефона и адреса без указания города для наших целей достаточно, то все замечательно, тогда поля ФИО и адрес могут образовывать первичный ключ. В любом случае проблема создания первичного ключа ложится на плечи того, кто проектирует базу данных (разрабатывает структуру хранения данных). Решением этой проблемы может стать либо выделение характеристик, которые естественным образом определяют запись в таблице (задание так называемого логического, или естественного, PK), либо создание дополнительного поля, предназначенного именно для однозначной идентификации записей в таблице (задание так называемого суррогатного, или искусственного, PK). Примером логического первичного ключа является номер паспорта в базе данных о паспортных данных жителей или ФИО и адрес в телефонной книге (таблица выше). Для задания суррогатного первичного ключа в нашу таблицу можно добавить поле id (идентификатор), значением которого будет целое число, уникальное для каждой строки таблицы. Использование таких суррогатных ключей

имеет смысл, если естественный первичный ключ представляет собой большой набор полей или его выделение нетривиально.

Кроме однозначной идентификации записи, первичные ключи используются для организации связей с другими таблицами.

Например, у нас есть три таблицы: содержащая информацию об исторических личностях (Persons), содержащая информацию об их изобретениях (Artifacts) и содержащая изображения как личностей, так и артефактов (Images) ([рис 10.1](#)).

Первичным ключом во всех этих таблицах является поле id (идентификатор). В таблице Artifacts есть поле author, в котором записан идентификатор, присвоенный автору изобретения в таблице Persons. Каждое значение этого поля является **внешним ключом** для первичного ключа таблицы Persons. Кроме того, в таблицах Persons и Artifacts есть поле photo, которое ссылается на изображение в таблице Images. Эти поля также являются внешними ключами для первичного ключа таблицы Images и устанавливают однозначную логическую связь Persons-Images и Artifacts-Images. То есть если значение внешнего ключа photo в таблице личности равно 10, то это значит, что фотография этой личности имеет id=10 в таблице изображений. Таким образом, **внешние ключи** используются для организации связей между таблицами базы данных (родительскими и дочерними) и для поддержания ограничений ссылочной целостности данных.

Рис. 10.1. Пример использования первичных ключей для организации связей с другими таблицами

Индексирование

Одна из основных задач, возникающих при работе с базами данных, – это задача поиска. При этом, поскольку информации в базе данных, как правило, содержится много, перед программистами встает задача не просто поиска, а эффективного поиска, т.е. поиска за сравнительно небольшое время и с достаточной точностью. Для этого (для оптимизации производительности запросов) производят **индексирование** некоторых полей таблицы. Использовать индексы полезно для быстрого поиска строк с указанным значением одного столбца. Без индекса чтение таблицы осуществляется по всей таблице, начиная с первой записи, пока не будут найдены соответствующие строки. Чем больше таблица, тем больше накладные расходы. Если же таблица содержит индекс по рассматриваемым столбцам, то база данных может быстро определить позицию для поиска в середине файла данных без просмотра всех данных. Это происходит потому, что база данных помещает проиндексированные поля поближе в памяти, так, чтобы можно было побыстрее найти их значения. Для таблицы, содержащей 1000 строк, это будет как минимум в 100 раз быстрее по сравнению с последовательным перебором всех записей. Однако в случае, когда необходим доступ почти ко всем 1000 строкам, быстрее будет последовательное чтение, так как при этом не требуется операций поиска по диску. Так что иногда индексы бывают только помехой. Например, если копируется большой объем данных в таблицу, то лучше

не иметь никаких индексов. Однако в некоторых случаях требуется задействовать сразу несколько индексов (например, для обработки запросов к часто используемым таблицам).

Если говорить о MySQL, то там существует три вида индексов: PRIMARY, UNIQUE, и INDEX, а слово ключ (KEY) используется как синоним слова индекс (INDEX). Все индексы хранятся в памяти в виде B-деревьев.

PRIMARY – уникальный индекс (ключ) с ограничением, что все индексированные им поля не могут иметь пустого значения (т.е. они NOT NULL). Таблица может иметь только один первичный индекс, но он может состоять из нескольких полей.

UNIQUE – ключ (индекс), задающий поля, которые могут иметь только уникальные значения.

INDEX – обычный индекс (как мы описали выше). В MySQL, кроме того, можно индексировать строковые поля по заданному числу символов от начала строки.

СУБД MySQL

Продолжим разговор о СУБД MySQL. MySQL – это реляционная система управления базами данных. То есть данные в ее базах хранятся в виде логически связанных между собой таблиц, доступ к которым осуществляется с помощью языка запросов SQL. MySQL – свободно распространяемая система, т.е. платить за ее применение не нужно. Кроме того, это достаточно быстрая, надежная и, главное, простая в использовании СУБД, вполне подходящая для не слишком глобальных проектов.

Работать с MySQL можно не только в текстовом режиме, но и в графическом. Существует очень популярный визуальный интерфейс (кстати, написанный на PHP) для работы с этой СУБД. Называется он PhpMyAdmin. Этот интерфейс позволяет значительно упростить работу с базами данных в MySQL.

В текстовом режиме работа с базой данных выглядит просто как ввод команд в командную строку ([рис 10.2](#)), а результаты выборок возвращаются в виде своеобразных таблиц, поля в которых налезают друг на друга, если данные не помещаются на экран ([рис 10.3](#)).

Рис. 10.2. Работа с MySQL в командной строке. Команда show databases — вывести все имеющиеся базы данных

PhpMyAdmin позволяет пользоваться всеми достоинствами браузера, включая прокрутку изображения, если оно не умещается на экран. Многие из базовых SQL-

функций работы с данными в PhpMyAdmin сведены к интуитивно понятным интерфейсам и действиям, напоминающим переход по ссылкам в Internet. Но, тем не менее, стоит все же поработать и в текстовом режиме.

Рис. 10.3. Работа с MySQL в командной строке. Результат обработки команды `show databases`

Перед тем как переходить к детальному изучению языка SQL, несколько слов об установке MySQL и подготовке к работе. Если вы не собираетесь заниматься администрированием сервера, то информация, приведенная ниже, пригодится вам только для общего развития. Итак, устанавливается MySQL очень просто – автоматически, пару раз нажмите ОК, и все. После этого вы можете зайти в директорию, где лежат файлы типа `mysql.exe`, `mysqld.exe` и т.п. (у нас под Windows XP это `c:\mysql\bin`) Последний файл запускает Mysql-сервер. В некоторых системах сервер запускается в виде сервиса. После запуска сервера следует запустить mysql-клиент, запустив программу `mysql.exe`. Здесь даже пароля не спросят. Более того, если вы наберете

```
shell> mysql.exe -u root
```

или

```
shell>mysql -u root mysql
```

то получите все права администратора mysql сервера. Кстати, выполнять эти команды надо, находясь в той директории, где лежат файлы `mysql.exe`.

Для начала, не вдаваясь в подробности команд, исправим эти два недочета (отсутствие пароля у администратора и возможность входа анонимным пользователям):

```
shell> mysql -u root mysql
mysql> UPDATE user SET Password=PASSWORD('new_password')
      WHERE user='root';
mysql> DELETE FROM user WHERE user='';
mysql> FLUSH PRIVILEGES;
```

Все данные о пользователях MySQL хранит в таблице `user` в специальной базе данных `mysql`, доступ к которой имеет только администратор сервера. Поэтому, чтобы изменить какой-либо пароль, нужно изменить эту таблицу. Пароль задается с помощью функции `PASSWORD`, которая кодирует введенные данные. Кроме изменения пароля администратора, нужно еще удалить всех пользователей, не имеющих логина (команда `DELETE`). Команда `Flush Privileges` заставляет вступить в действие изменения, произошедшие в системной базе данных (`mysql`).

Теперь создадим базу данных, с которой будем работать (мы все еще работаем как администратор сервера):

```
mysql>create database book;
```

Как можно заметить, все команды в MySQL заканчиваются точкой с запятой. Если вы забыли поставить этот знак, то выдается приглашение его поставить до тех пор, пока это не будет сделано:

```
mysql> show tables
->
->
```

Теперь последнее действие – создадим простого пользователя, предоставим ему доступ к созданной базе данных, и начнем работать.

```
mysql> GRANT ALL PRIVILEGES ON book.* TO nina@localhost
IDENTIFIED BY '123';
```

Команда GRANT наделяет пользователя nina, зашедшего на сервер с этой же машины (с localhost) и идентифицируемого паролем «123», определенными правами (в данном случае всеми) на все таблицы базы данных book. Теперь мы можем выйти и зайти как пользователь nina с соответствующим паролем:

```
shell>mysql -u nina -p
Enter password: ***
Welcome to the MySQL monitor!...
mysql>
```

Если вы собираетесь пользоваться базой данных на чужом сервере, то его администратор проделает все описанные выше действия за вас, т.е. все настроит и создаст пользователя и базу данных. В следующей главе описаны команды языка SQL, которые пригодятся для работы с данными, хранящимися в СУБД MySQL.

Язык SQL

Итак, мы в общих чертах познакомились с основными понятиями теории баз данных, установили и настроили для работы MySQL. Теперь самое время научиться манипулировать данными, хранящимися в базах данных. Для этого нам понадобится SQL – структурированный язык запросов. Этот язык дает возможность создавать, редактировать и удалять информацию, хранящуюся в базах данных, создавать новые базы данных и многое другое. SQL является стандартом ANSI (Американский национальный институт стандартов) и ISO (Международная организация по стандартизации).

Немного истории

Первый международный стандарт языка SQL был принят в 1989 г., его часто называют SQL/89. Среди недостатков этого стандарта выделяют в первую очередь то, что многие важные свойства он устанавливал как определяемые в реализации. Отсюда произошло множество расхождений в реализациях языка разными производителями. Кроме того, высказывались претензии по поводу отсутствия в этом стандарте упоминаний о практических аспектах языка, таких как его встраивание в язык программирования Си.

Следующий международный стандарт языка SQL был принят в конце 1992 г. И стал называться SQL/92. Он получился гораздо более точным и полным, чем SQL/89, хотя и не был лишен недостатков. В настоящее время большинство систем почти полностью реализуют этот стандарт. Однако, как известно, прогресс не остановишь, и в 1999 году появился новый стандарт SQL:1999, также известный как SQL3. SQL3 характеризуется как «объектно-ориентированный SQL» и является основой нескольких объектно-реляционных систем управления базами данных (например, ORACLE8 компании Oracle, Universal Server компании Informix и DB2 Universal Database компании IBM). Этот стандарт является не просто слиянием SQL-92 и объектной технологии. Он содержит ряд расширений традиционного SQL, а сам документ составлен таким образом, чтобы добиться более эффективной работы в области стандартизации в будущем.

Если говорить о MySQL, то она соответствует начальному уровню SQL92, содержит несколько расширений этого стандарта и стремится к полной поддержке стандарта ANSI SQL99, но без ущерба для скорости и качества кода.

Далее, говоря об основах языка SQL, будем придерживаться его реализации в СУБД MySQL.

Основные операторы языка SQL

Функции любой СУБД включают:

- создание, удаление, изменение базы данных (БД);
- добавление, изменение, удаление, назначение прав пользователя;
- внесение, удаление и изменение данных в БД (таблиц и записей);
- выборку данных из БД.

К первым двум функциям имеют доступ только администраторы СУБД или привилегированные пользователи. Рассмотрим, как решаются последние две задачи (на самом деле это семь задач).

Прежде чем что-либо делать с данными, нужно создать таблицы, в которых эти данные будут храниться, научиться изменять структуру этих таблиц и удалять их, если потребуется. Для этого в языке SQL существуют операторы CREATE TABLE, ALTER TABLE и DROP TABLE.

Оператор CREATE TABLE

Оператор CREATE TABLE создает таблицу с заданным именем в текущей базе данных. Правила для допустимых имен таблицы приведены в документации. Если нет активной текущей базы данных или указанная таблица уже существует, то возникает ошибка выполнения команды.

В версии MySQL 3.22 и более поздних имя таблицы может быть указано как имя_базы_данных.имя_таблицы. Эта форма записи работает независимо от того, является ли указанная база данных текущей.

В версии MySQL 3.23 при создании таблицы можно использовать ключевое слово TEMPORARY. Временная таблица автоматически удаляется по завершении соединения, а ее имя действительно только в течение данного соединения. Это означает, что в двух разных соединениях могут использоваться временные таблицы с одинаковыми именами без конфликта друг с другом или с существующей таблицей с тем же именем (существующая таблица скрыта, пока не удалена временная таблица). В версии MySQL 4.0.2 для создания временных таблиц необходимо иметь привилегии CREATE TEMPORARY TABLES.

В версии MySQL 3.23 и более поздних можно использовать ключевые слова IF NOT EXISTS для того, чтобы не возникала ошибка, если указанная таблица уже существует. Следует учитывать, что при этом идентичность структур этих таблиц не проверяется.

Каждая таблица представлена набором определенных файлов в директории базы данных.

Синтаксис

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS]
    имя_таблицы [(определение_столбца, ...)]
    [опции_таблицы] [select_выражение]
```

В выражении определение_столбца перечисляют, какие столбцы должны быть созданы в таблице. Каждый столбец таблицы может быть пустым (NULL), иметь значение по умолчанию, являться ключом или автоинкрементом. Кроме того, для каждого столбца обязательно указывается тип данных, которые будут в нем храниться. Если не указывается ни NULL, ни NOT NULL, то столбец интерпретируется так, как будто указано NULL. Если поле помечают как автоинкремент (AUTO_INCREMENT), то его значение автоматически увеличивается на единицу каждый раз, когда происходит добавление данных в таблицу и в это поле записывается пустое значение (NULL, т.е. ничего не записывается) или 0. Автоинкремент в таблице может быть только один, и при этом он обязательно должен быть проиндексирован. Последовательность AUTO_INCREMENT начинается с 1. Наличие автоинкремента является одной из

особенностей MySQL. Формально описание столбца (определение_столбца) выглядит так:

```
имя_столбца тип [NOT NULL | NULL]
  [DEFAULT значение_по_умолчанию]
  [AUTO_INCREMENT] [PRIMARY KEY]
  [reference_definition]
```

Тип столбца (тип в выражении определение_столбца) может быть одним из следующих:

- целый: INT[(length)] [UNSIGNED] [ZEROFILL]
- действительный: REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]
- символьный: CHAR(length) [BINARY] и VARCHAR(length) [BINARY]
- дата и время: DATE и TIME
- для работы с большими объектами: BLOB
- текстовый: TEXT
- перечислимое множество: ENUM(value1,value2,value3,...) и SET(value1,value2,value3,...)

Полный список типов смотрите в документации MySQL.

Вместо перечисления столбцов и их свойств в определении_столбца можно задавать списки ключевых и индексных полей, ограничения и проверки:

```
PRIMARY KEY (имя_индексируемого_столбца, ...)
```

или

```
KEY [имя_индекса] (имя_индексируемого_столбца, ...)
```

или

```
INDEX [имя_индекса] (имя_индексируемого_столбца, ...)
```

или

```
UNIQUE [INDEX] [имя_индекса]
  (имя_индексируемого_столбца, ...)
```

или

```
FULLTEXT [INDEX] [имя_индекса]
  (имя_индексируемого_столбца, ...)
```

или

```
[CONSTRAINT symbol]
FOREIGN KEY [имя_индекса]
    (имя_индексируемого_столбца,...)
[reference_definition]
```

или

```
CHECK (expr)
```

При задании всех этих элементов указывается список полей (столбцов), которые будут входить в индекс, ключ или ограничение, имя_индексируемого_столбца записывается следующим образом:

```
имя_столбца [(длина_индекса)]
```

FOREIGN KEY, CHECK и REFERENCES на самом деле ничего не делают в MySQL. Они добавлены только для совместимости с другими SQL-серверами. Поэтому на них мы останавливаться не будем.

Кроме всего перечисленного, при создании таблицы можно указать некоторые ее свойства (опции_таблицы), например такие:

- тип таблицы: TYPE = {BDB | HEAP | ISAM | InnoDB | MERGE | MRG_MYISAM | MYISAM }
- начальное значение счетчика автоинкремента: AUTO_INCREMENT = число
- средняя длина строк в таблице: AVG_ROW_LENGTH = число
- комментарии к таблице (строка из 60 символов): COMMENT = "строка"
- максимальное и минимальное предполагаемое число строк: MAX_ROWS = число и MIN_ROWS = число

И последний (опять же опциональный) элемент команды CREATE – это выражение SELECT (select_выражение). Синтаксис такой:

```
[IGNORE | REPLACE] SELECT ...
    (любое корректное выражение SELECT)
```

Если при создании таблицы в команде CREATE указывается выражение SELECT, то все поля, полученные выборкой, добавляются в создаваемую таблицу.

Пример 10.1. Создадим таблицу Persons, структура которой была приведена на [рисунке 10.1](#).

```
mysql>CREATE TABLE Persons
    (id INT PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(50), last_name
    VARCHAR(100), death_date INT,
```

```
description TEXT, photo INT,  
citizenship CHAR(50) DEFAULT 'Russia');
```

Пример 10.1. Создание таблицы Persons

С помощью специфичной для MySQL команды SHOW можно просмотреть существующие базы данных, таблицы в базе данных и поля в таблице.

Показать все базы данных:

```
mysql>SHOW databases;
```

Сделать текущей базу данных book и показать все таблицы в ней:

```
mysql>use book;  
mysql>show tables;
```

Показать все столбцы в таблице Persons:

```
mysql> show columns from Persons;
```

Оператор DROP TABLE

Оператор DROP TABLE удаляет одну или несколько таблиц. Все табличные данные и определения удаляются, так что при работе с этой командой следует соблюдать осторожность.

Синтаксис:

```
DROP TABLE [IF EXISTS] имя_таблицы  
[, имя_таблицы,...]  
[RESTRICT | CASCADE]
```

В версии MySQL 3.22 и более поздних можно использовать ключевые слова IF EXISTS, чтобы предупредить ошибку, если указанные таблицы не существуют.

Опции RESTRICT и CASCADE позволяют упростить перенос программы с других СУБД. В данный момент они не задействованы.

```
mysql> DROP TABLE IF EXISTS Persons,  
Artifacts, test;
```

Пример 10.2. Использование оператора DROP TABLE

Оператор ALTER TABLE

Оператор ALTER TABLE обеспечивает возможность изменять структуру существующей таблицы. Например, можно добавлять или удалять столбцы,

создавать или уничтожать индексы или переименовывать столбцы либо саму таблицу. Можно также изменять комментарий для таблицы и ее тип.

Синтаксис:

```
ALTER [IGNORE] TABLE имя_таблицы
    alter_specification
    [, alter_specification ...]
```

Можно производить следующие изменения в таблице (все они записываются в alter_specification):

- добавление поля:

```
ADD [COLUMN] определение_столбца
[FIRST | AFTER имя_столбца ]
```

или

```
ADD [COLUMN] (определение_столбца,
определение_столбца,...)
```

Здесь, как и далее, определение_столбца записывается так же, как при создании таблицы.

- добавление индексов:

```
ADD INDEX [имя_индекса] (имя_индексируемого_столбца,...) или ADD
PRIMARY KEY (имя_индексируемого_столбца,...) или ADD UNIQUE
[имя_индекса] (имя_индексируемого_столбца,...) или ADD FULLTEXT
[имя_индекса] (имя_индексируемого_столбца,...)
```

- изменение поля:

```
ALTER [COLUMN] имя_столбца {SET DEFAULT literal | DROP DEFAULT} или
CHANGE [COLUMN] старое_имя_столбца определение_столбца или
MODIFY [COLUMN] определение_столбца
```

- удаление поля, индекса, ключа:

```
DROP [COLUMN] имя_столбца
DROP PRIMARY KEY
DROP INDEX имя_индекса
```

- переименование таблицы:

```
RENAME [TO] новое_имя_таблицы
```

- переупорядочение полей таблицы:

ORDER BY поле

или

опции_таблицы

Если оператор ALTER TABLE используется для изменения определения типа столбца, но DESCRIBE имя_таблицы показывает, что столбец не изменился, то, возможно, MySQL игнорирует данную модификацию по одной из причин, описанных в специальном разделе документации. Например, при попытке изменить столбец VARCHAR на CHAR MySQL будет продолжать использовать VARCHAR, если данная таблица содержит другие столбцы с переменной длиной.

Оператор ALTER TABLE во время работы создает временную копию исходной таблицы. Требуемое изменение выполняется на копии, затем исходная таблица удаляется, а новая переименовывается. Это делается для того, чтобы в новую таблицу автоматически попадали все обновления, кроме неудавшихся. Во время выполнения ALTER TABLE исходная таблица доступна для чтения другими клиентами. Операции обновления и записи в этой таблице приостанавливаются, пока не будет готова новая таблица. Следует отметить, что при использовании любой другой опции для ALTER TABLE, кроме RENAME, MySQL всегда будет создавать временную таблицу, даже если данные, строго говоря, и не нуждаются в копировании (например, при изменении имени столбца).

Пример10.3. Добавим в созданную таблицу Persons поле для записи года рождения человека:

```
mysql> ALTER TABLE Persons
      ADD bday INTEGER AFTER last_name;
```

Пример 10.3. Добавление в таблицу Persons поля для записи года рождения человека

Итак, мы научились работать с таблицами: создавать, удалять и изменять их. Теперь разберемся, как делать то же самое с данными, которые в этих таблицах хранятся.

Оператор SELECT

Оператор SELECT применяется для извлечения строк, выбранных из одной или нескольких таблиц. То есть с его помощью мы задаем столбцы или выражения, которые надо извлечь (select_выражения), таблицы (table_references), из которых должна производиться выборка, и, возможно, условие (where_definition), которому должны соответствовать данные в этих столбцах, и порядок, в котором эти данные нужно выдать.

Кроме того, оператор SELECT можно использовать для извлечения строк, вычисленных без ссылки на какую-либо таблицу. Например, чтобы вычислить, чему равно 2*2, нужно просто написать

```
mysql> SELECT 2*2;
```

Упрощенно структуру оператора SELECT можно представить следующим образом:

```
SELECT select_выражение1, select_выражение2,  
...  
[FROM table_references  
  [WHERE where_definition]  
  [ORDER BY {число | имя_столбца |  
            формула}  
  [ASC | DESC], ...]]
```

Квадратные скобки [] означают, что использование находящегося в них оператора необязательно, вертикальная черта | означает перечисление возможных вариантов. После ключевого слова ORDER BY указывают имя столбца, число (целое беззнаковое) или формулу и способ упорядочения (по возрастанию – ASC, или по убыванию – DESC). По умолчанию используется упорядочение по возрастанию.

Когда в select_выражении мы пишем «*», это значит выбрать все столбцы. Кроме «*» в select_выражения могут использоваться функции типа max, min и avg.

Пример 10.4. Выбрать из таблицы Persons все данные, для которых поле first_name имеет значение 'Александр':

```
mysql> SELECT * FROM Persons  
        WHERE first_name='Александр';
```

Пример 10.4. Использование оператора SELECT

Выбрать название и описание (title, description) артефакта под номером 10:

```
mysql> SELECT title,description  
        FROM Artifacts WHERE id=10;
```

Оператор INSERT

Оператор INSERT вставляет новые строки в существующую таблицу. Оператор имеет несколько форм. Параметр имя_таблицы во всех этих формах задает таблицу, в которую должны быть внесены строки. Столбцы, для которых задаются значения, указываются в списке имен столбцов (имя_столбца) или в части SET.

Синтаксис:

- INSERT [LOW_PRIORITY | DELAYED] [IGNORE]

```
[INTO] имя_таблицы [(имя_столбца,...)]  
VALUES (выражение,...), (...),...
```

Эта форма команды INSERT вставляет строки в соответствии с точно указанными в команде значениями. В скобках после имени таблицы перечисляются столбцы, а после ключевого слова VALUES – их значения.

Например:

```
mysql> INSERT INTO Persons  
      (last_name, bday) VALUES  
      ('Иванов', '1934');
```

вставит в таблицу Persons строку, в которой значения фамилии (last_name) и даты рождения (bday) будут заданы соответственно как «Иванов» и «1934».

- INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
 [INTO] имя_таблицы [(имя_столбца,...)]
 SELECT ...

Эта форма команды INSERT вставляет строки, выбранные из другой таблицы или таблиц.

Например:

```
mysql> INSERT INTO Artifacts (author)  
      SELECT id FROM Persons  
      WHERE last_name='Иванов'  
      AND bday='1934';
```

вставит в таблицу Artifacts в поле «автор» (author) значение идентификатора, выбранного из таблицы Persons по условию, что фамилия человека Иванов.

- INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
 [INTO] имя_таблицы
 SET имя_столбца=выражение,
 имя_столбца=выражение, ...

Например:

```
mysql> INSERT INTO Persons  
      SET last_name='Петров',  
      first_name='Иван';
```

Эта команда вставит в таблицу Persons в поле last_name значение «Петров», а в поле first_name – строку «Иван».

Форма INSERT ... VALUES со списком из нескольких значений поддерживается в версии MySQL 3.22.5 и более поздних. Синтаксис выражения имя_столбца=выражение поддерживается в версии MySQL 3.22.10 и более поздних.

Действуют следующие соглашения.

- Если не указан список столбцов для INSERT ... VALUES или INSERT ... SELECT, то величины для всех столбцов должны быть определены в списке VALUES() или в результате работы SELECT. Если порядок столбцов в таблице неизвестен, для его получения можно использовать DESCRIBE имя_таблицы.
- Любой столбец, для которого явно не указано значение, будет установлен в свое значение по умолчанию. Например, если в заданном списке столбцов не указаны все столбцы в данной таблице, то не упомянутые столбцы устанавливаются в свои значения по умолчанию.
- Выражение expression может относиться к любому столбцу, который ранее был внесен в список значений. Например, можно указать следующее:

```
mysql> INSERT INTO имя_таблицы (col1,col2)
VALUES (15,col1*2);
```

Но нельзя указать:

```
mysql> INSERT INTO имя_таблицы (col1,col2)
VALUES (col2*2,15);
```

Мы еще не обсудили три необязательных параметра, присутствующих во всех трех формах команды: LOW_PRIORITY, DELAYED и IGNORE.

Параметры LOW_PRIORITY и DELAYED используются, когда с таблицей работает большое число пользователей. Они предписывают устанавливать приоритет данной операции перед операциями других пользователей. Если указывается ключевое слово LOW_PRIORITY, то выполнение данной команды INSERT будет задержано до тех пор, пока другие клиенты не завершат чтение этой таблицы. В этом случае клиент должен ожидать, пока данная команда вставки не будет завершена, что в случае интенсивного использования таблицы может потребовать значительного времени. В противоположность этому команда INSERT DELAYED позволяет данному клиенту продолжать операцию сразу же, независимо от других пользователей.

Если в команде INSERT указывается ключевое слово IGNORE, то все строки, имеющие дублирующиеся ключи PRIMARY или UNIQUE в этой таблице, будут проигнорированы и не внесены в таблицу. Если не указывать IGNORE, то данная операция вставки прекращается при обнаружении строки, имеющей дублирующееся значение существующего ключа.

Оператор UPDATE

Синтаксис:

```
UPDATE [LOW_PRIORITY] [IGNORE] имя_таблицы
  SET имя_столбца1=выражение1
  [, имя_столбца2=выражение2, ...]
  [WHERE where_definition]
  [LIMIT число]
```

Оператор UPDATE обновляет значения существующих столбцов таблицы в соответствии с введенными значениями. В выражении SET указывается, какие именно столбцы следует модифицировать и какие величины должны быть в них установлены. В выражении WHERE, если оно присутствует, задается, какие строки подлежат обновлению. В остальных случаях обновляются все строки. Если задано выражение ORDER BY, то строки будут обновляться в указанном в нем порядке.

Если указывается ключевое слово LOW_PRIORITY, то выполнение данной команды UPDATE задерживается до тех пор, пока другие клиенты не завершат чтение этой таблицы.

Если указывается ключевое слово IGNORE, то команда обновления не будет прервана, даже если возникнет ошибка дублирования ключей. Строки, из-за которых возникают конфликтные ситуации, обновлены не будут.

Если в выражении, которое задает новое значение столбца, используется имя этого поля, то команда UPDATE использует для этого столбца его текущее значение. Например, следующая команда устанавливает столбец death_date в значение, на единицу большее его текущей величины:

```
mysql> UPDATE Persons
  SET death_date=death_date+1;
```

В версии MySQL 3.23 можно использовать параметр LIMIT #, чтобы убедиться, что было изменено только заданное количество строк.

Например, такая операция заменит в первой строке нашей таблицы экспонатов название title на строку «Ламповая ЭВМ»:

```
mysql> UPDATE Artifacts
  SET title='Ламповая ЭВМ' Limit 1;
```

Оператор DELETE

Оператор DELETE удаляет из таблицы имя_таблицы строки, удовлетворяющие заданным в where_definition условиям, и возвращает число удаленных записей.

Если оператор DELETE запускается без определения WHERE, то удаляются все строки.

Синтаксис:

```
DELETE [LOW_PRIORITY] FROM имя_таблицы
    [WHERE where_definition]
    [LIMIT rows]
```

Например, следующая команда удалит из таблицы Persons все записи, у которых поле «год рождения» (bday) больше 2003:

```
mysql> DELETE FROM Persons WHERE bday>2003;
```

Удалить все записи в таблице можно еще и с помощью такой команды:

```
mysql> DELETE FROM Persons WHERE 1>0;
```

Но этот метод работает гораздо медленнее, чем использование той же команды без условия:

```
mysql> DELETE FROM Persons;
```

Специфическая для MySQL опция LIMIT для команды DELETE указывает серверу максимальное количество строк, которые следует удалить до возврата управления клиенту. Эта опция может использоваться для гарантии того, что данная команда DELETE не потребует слишком много времени для выполнения.

В дистрибутив PHP входит расширение, содержащее встроенные функции для работы с базой данных MySQL. В этой лекции мы познакомимся с некоторыми основными функциями для работы с MySQL, которые потребуются для решения задач построения web-интерфейсов с целью отображения и наполнения базы данных. Возникает вопрос, зачем строить такие интерфейсы? Для того чтобы вносить информацию в базу данных и просматривать ее содержимое могли люди, не знакомые с языком запросов SQL. При работе с web-интерфейсом для добавления информации в базу данных человеку нужно просто ввести эти данные в html-форму и отправить их на сервер, а наш скрипт сделает все остальное. А для просмотра содержимого таблиц достаточно просто щелкнуть по ссылке и зайти на нужную страницу.

Для наглядности будем строить эти интерфейсы для таблицы Artifacts, в которой содержится информация об экспонатах виртуального музея информатики. В предыдущей лекции мы уже приводили структуру этой коллекции, а также ее связи с коллекциями описания персон (Persons) и изображений (Images). Напомним, что каждый экспонат в коллекции Artifacts описывается с помощью следующих характеристик:

- название (title);
- автор (author);
- описание (description);
- альтернативное название (alternative);
- изображение (photo).

Название и альтернативное название являются строками менее чем 255 символов длиной (т.е. имеют тип VARCHAR(255)), описание - текстовое поле (имеет тип TEXT), а в полях "автор" и "изображение" содержатся идентификаторы автора из коллекции Persons и изображения экспоната из коллекции Images соответственно.

Построение интерфейса для добавления информации

Итак, у нас есть какая-то таблица в базе данных. Чтобы построить интерфейс для добавления информации в эту таблицу, нужно ее структуру (т.е. набор ее полей) отобразить в html-форму.

Разобьем эту задачу на следующие подзадачи:

- установка соединения с БД;
- выбор рабочей БД;
- получение списка полей таблицы;
- отображение полей в html-форму.

После этого данные, введенные в форму, нужно записать в базу данных. Рассмотрим все эти задачи по порядку.

Установка соединения

Итак, первое, что нужно сделать, - это установить соединение с базой данных. Воспользуемся функцией `mysql_connect`.

Синтаксис `mysql_connect`

```
ресурс mysql_connect ( [строка server  
    [, строка username [, строка password  
    [, логическое new_link  
    [, целое client_flags]]]])
```

Данная функция устанавливает соединение с сервером MySQL и возвращает указатель на это соединение или FALSE в случае неудачи. Для отсутствующих параметров устанавливаются следующие значения по умолчанию:

```
server = 'localhost:3306'  
username = имя пользователя владельца  
           процесса сервера  
password = пустой пароль
```

Если функция вызывается дважды с одними и теми же параметрами, то новое соединение не устанавливается, а возвращается ссылка на старое соединение. Чтобы этого избежать, используют параметр `new_link`, который заставляет в любом случае открыть еще одно соединение.

Параметр `client_flags` - это комбинация следующих констант: `MYSQL_CLIENT_COMPRESS` (использовать протокол сжатия), `MYSQL_CLIENT_IGNORE_SPACE` (позволяет вставлять пробелы после имен функций), `MYSQL_CLIENT_INTERACTIVE` (ждать `interactive_timeout` секунд - вместо `wait_timeout` - до закрытия соединения).

Параметр `new_link` появился в PHP 4.2.0, а параметр `client_flags` - в PHP 4.3.0.

Соединение с сервером закрывается при завершении исполнения скрипта, если оно до этого не было закрыто с помощью функции `mysql_close()`.

Итак, устанавливаем соединение с базой данных на локальном сервере для пользователя `nina` с паролем "123":

```
<?  
$conn = mysql_connect(  
    "localhost", "nina", "123")  
or die("Невозможно установить  
    соединение: ". mysql_error());  
echo "Соединение установлено";  
mysql_close($conn);  
?>
```

Действие `mysql_connect` равносильно команде

```
shell>mysql -u nina -p123
```

Выбор базы данных

После установки соединения нужно выбрать базу данных, с которой будем работать. Наши данные хранятся в базе данных `book`. В MySQL выбор базы данных осуществляется с помощью команды `use`:

```
mysql>use book;
```

В PHP для этого существует функция `mysql_select_db`.

Синтаксис `mysql_select_db`:

```
логическое mysql_select_db (  
    строка database_name  
    [, ресурс link_identifier])
```

Эта функция возвращает TRUE в случае успешного выбора базы данных и FALSE - в противном случае.

Сделаем базу данных `book` рабочей:

```
<?  
$conn = mysql_connect(  
    "localhost", "nina", "123")  
or die("Невозможно установить  
    соединение: ". mysql_error());  
echo "Соединение установлено";  
mysql_select_db("book");  
?>
```

Получение списка полей таблицы

Теперь можно заняться собственно решением задачи. Как получить список полей таблицы? Очень просто. В PHP и на этот случай есть своя команда - `mysql_list_fields`.

Синтаксис `mysql_list_fields`

```
ресурс mysql_list_fields (  
    строка database_name,  
    строка table_name  
    [, ресурс link_identifier])
```

Эта функция возвращает список полей в таблице `table_name` в базе данных `database_name`. Получается, что выбирать базу данных нам было необязательно, но это пригодится позже. Как можно заметить, результат работы этой функции - переменная типа ресурс. То есть это не совсем то, что мы хотели получить. Это ссылка, которую можно использовать для получения информации о полях таблицы, включая их названия, типы и флаги.

Функция `mysql_field_name` возвращает имя поля, полученного в результате выполнения запроса. Функция `mysql_field_len` возвращает длину поля. Функция `mysql_field_type` возвращает тип поля, а функция `mysql_field_flags` возвращает список флагов поля, записанных через пробел. Типы поля могут быть `int`, `real`, `string`, `blob` и т.д. Флаги могут быть `not_null`, `primary_key`, `unique_key`, `blob`, `auto_increment` и т.д.

Синтаксис у всех этих команд одинаков:

```

строка mysql_field_name (
    ресурс result, целое field_offset)
строка mysql_field_type (
    ресурс result, целое field_offset)
строка mysql_field_flags (
    ресурс result, целое field_offset)
строка mysql_field_len (
    ресурс result, целое field_offset)

```

Здесь `result` - это идентификатор результата запроса (например, запроса, отправленного функциями `mysql_list_fields` или `mysql_query` (о ней будет рассказано позднее)), а `field_offset` - порядковый номер поля в результате.

Вообще говоря, то, что возвращают функции типа `mysql_list_fields` или `mysql_query`, представляет собой таблицу, а точнее, указатель на нее. Чтобы получить из этой таблицы конкретные значения, нужно задействовать специальные функции, которые построчно читают эту таблицу. К таким функциям и относятся `mysql_field_name` и т.п. Чтобы перебрать все строки в таблице результата выполнения запроса, нужно знать число строк в этой таблице. Команда `mysql_num_rows(ресурс result)` возвращает число строк во множестве результатов `result`.

А теперь попробуем получить список полей таблицы `Artifacts` (коллекция экспонатов).

```

<?
$conn = mysql_connect(
    "localhost","nina","123")
or die("Невозможно установить
    соединение: ". mysql_error());
echo "Соединение установлено";
mysql_select_db("book");
$list_f = mysql_list_fields (
    "book","Artifacts",$conn);
$n = mysql_num_fields($list_f);
for($i=0;$i<$n; $i++){
    $type = mysql_field_type($list_f, $i);
    $name_f = mysql_field_name($list_f,$i);
    $len = mysql_field_len($list_f, $i);
    $flags_str = mysql_field_flags (
        $list_f, $i);
echo "<br>Имя поля: ". $name_f;
echo "<br>Тип поля: ". $type;
echo "<br>Длина поля: ". $len;
echo "<br>Строка флагов поля: ".
    $flags_str . "<hr>";
}
?>

```

В результате должно получиться примерно вот что (если в таблице всего два поля, конечно):

```

Имя поля: id
Тип поля: int

```

```
Длина поля: 11
Строка флагов поля:
    not_null primary_key auto_increment
Имя поля: title
Тип поля: string
Длина поля: 255
Строка флагов поля:
```

Отображение списка полей в html-форму

Теперь немножко подкорректируем предыдущий пример. Будем не просто выводить информацию о поле, а отображать его в подходящий элемент html-формы. Так, элементы типа BLOB переведем в textarea (заметим, что поле description, которое мы создавали с типом TEXT, отображается как имеющее тип BLOB), числа и строки отобразим в текстовые строки ввода `<input type=text>`, а элемент, имеющий метку автоинкремента, вообще не будем отображать, поскольку его значение устанавливается автоматически.

Все это решается довольно просто, за исключением выделения из списка флагов флага `auto_increment`. Для этого нужно воспользоваться функцией `explode`.

Синтаксис `explode`:

```
массив explode( строка separator,
               строка string [, int limit])
```

Эта функция разбивает строку `string` на части с помощью разделителя `separator` и возвращает массив полученных строк.

В нашем случае в качестве разделителя нужно взять пробел " ", а в качестве исходной строки для разбиения - строку флагов поля.

Итак, создадим форму для ввода данных в таблицу `Artifacts`:

```
<?
$conn=mysql_connect("localhost","nina","123");
    // устанавливаем соединение
$database = "book";
$table_name = "Artifacts";
mysql_select_db($database); // выбираем базу данных для
    // работы
$list_f = mysql_list_fields($database,$table_name);
    // получаем список полей в базе
$n = mysql_num_fields($list_f); // число строк в результате
    // предыдущего запроса (т.е. сколько всего
    // полей в таблице Artifacts)
echo "<form method=post action=insert.php>";
    // создаем форму для ввода данных
echo "&nbsp;<TABLE BORDER=0 CELLSPACING=0 width=50% ><tr>
    <TD BGCOLOR='#005533' align=center><font color='#FFFFFF'>
```


Синтаксис mysql_query

```
ресурс mysql_query ( строка query  
[, ресурс link_identifier])
```

mysql_query() посылает SQL-запрос активной базе данных MySQL сервера, который определяется с помощью указателя link_identifier (это ссылка на какое-то соединение с сервером MySQL). Если параметр link_identifier опущен, используется последнее открытое соединение. Если открытые соединения отсутствуют, функция пытается соединиться с СУБД, аналогично функции mysql_connect() без параметров. Результат запроса буферизируется.

Замечание: строка запроса НЕ должна заканчиваться точкой с запятой.

Только для запросов SELECT, SHOW, EXPLAIN, DESCRIBE, mysql_query() возвращает указатель на результат запроса, или FALSE, если запрос не был выполнен. В остальных случаях mysql_query() возвращает TRUE, если запрос выполнен успешно, и FALSE - в случае ошибки. Значение, не равное FALSE, говорит о том, что запрос был выполнен успешно. Оно не говорит о количестве затронутых или возвращенных рядов. Вполне возможна ситуация, когда успешный запрос не затронет ни одного ряда. mysql_query() также считается ошибочным и вернет FALSE, если у пользователя недостаточно прав для работы с указанной в запросе таблицей.

Итак, теперь мы знаем, как отправить запрос на вставку строк в базу данных. Заметим, что в предыдущем примере элементы формы мы назвали именами полей таблицы. Поэтому они будут доступны в скрипте insert.php, обрабатывающем данные формы, как переменные вида

```
$_POST['имя_поля'].  
  
<?  
$conn=mysql_connect("localhost","nina","123");// устанавливаем  
    // соединение  
$database = "book";  
$table_name = "Artifacts";  
mysql_select_db($database); // выбираем базу данных  
$list_f = mysql_list_fields($database,$table_name);  
    // получаем список полей в базе  
$n = mysql_num_fields($list_f); // число строк в результате  
    // предыдущего запроса  
// составим один запрос сразу для всех полей таблицы  
$sql = "INSERT INTO $table_name SET "; // начинаем создавать  
    // запрос, перебираем все поля таблицы  
for($i=0;$i<$n; $i++){  
    $name_f = mysql_field_name ($list_f,$i); // вычисляем имя поля  
    $value = $_POST[$name_f]; // вычисляем значение поля  
    $j = $i + 1;  
    $sql = $sql . $name_f." = '$value'"; // дописываем в  
        // строку $sql пару имя=значение  
    if ($j <> $n) $sql = $sql . ", "; // если поле не  
        // последнее в списке, то ставим запятую
```

```

}
// перед тем как записывать что-то в базу,
// можно посмотреть, какой запрос получился
//echo $sql;
$result = mysql_query($sql,$conn); // отправляем запрос
// выводим сообщение успешно ли выполнен запрос
if (!$result) echo " Can't add ($table_name) ";
    else echo "Success!<br>";
?>

```

Листинг 11.0.2. insert.php

Итак, задачу добавления данных с помощью web-интерфейса мы решили. Однако тут есть одна тонкость. При решении мы не учитывали тот факт, что значения некоторых полей (author, photo) должны браться из других таблиц (Persons, Images). Поскольку MySQL с внешними ключами не работает, этот момент остается на совести разработчиков системы, т.е. на нашей совести. Нужно дописать программу таким образом, чтобы была возможность вводить в такие поля правильные значения. Но мы делать этого не будем, поскольку задача лекции состоит в том, чтобы познакомить читателя с элементами технологии, а не в том, чтобы создать работающую систему. Кроме того, имеющихся у читателя знаний вполне достаточно, чтобы решить эту проблему самостоятельно. Мы же обратимся к другой задаче - отображение данных, хранящихся в базе данных СУБД MySQL.

Отображение данных, хранящихся в MySQL

Чтобы отобразить какие-то данные в браузер с помощью PHP, нужно сначала получить эти данные в виде переменных PHP. При работе с MySQL без посредника (такого, как PHP) выборка данных производится с помощью команды SELECT языка SQL:

```
mysql> SELECT * FROM Artifacts;
```

В предыдущей главе мы говорили, что любой запрос, в том числе и на выборку, можно отправить на сервер с помощью функции `mysql_query()`; Там у нас стояла немного другая задача - получить данные из формы и отправить их с помощью запроса на вставку в базу данных. Результатом работы `mysql_query()` там могло быть только одно из выражений, TRUE или FALSE. Теперь же требуется отправить запрос на выбор всех полей, а результат отобразить в браузере. И здесь результат - это целая таблица значений, а точнее, указатель на эту таблицу. Так что нужны какие-то аналоги функции `mysql_field_name()`, только чтобы они извлекали из результата запроса не имя, а значение поля. Таких функций в PHP несколько. Наиболее популярные - `mysql_result()` и `mysql_fetch_array()`.

Синтаксис `mysql_result`

```
смешанное mysql_result (ресурс result,
    целое row [, смешанное field])
```

`mysql_result()` возвращает значение одной ячейки результата запроса. Аргумент `field` может быть порядковым номером поля в результате, именем поля или именем поля с именем таблицы через точку `tablename.fieldname`. Если для имени поля в запросе применялся алиас (`'select foo as bar from...'`), используйте его вместо реального имени поля.

Работая с большими результатами запросов, следует задействовать одну из функций, обрабатывающих сразу целый ряд результата (например, `mysql_fetch_row()`, `mysql_fetch_array()` и т.д.). Так как эти функции возвращают значение нескольких ячеек сразу, они НАМНОГО быстрее `mysql_result()`. Кроме того, нужно учесть, что указание численного смещения (номера поля) работает намного быстрее, чем указание колонки или колонки и таблицы через точку.

Вызовы функции `mysql_result()` не должны смешиваться с другими функциями, работающими с результатом запроса.

Синтаксис `mysql_fetch_array`

```
массив mysql_fetch_array ( ресурс result  
    [, целое result_type])
```

Эта функция обрабатывает ряд результата запроса, возвращая массив (ассоциативный, численный или оба) с обработанным рядом результата запроса, или `FALSE`, если рядов больше нет.

`mysql_fetch_array()` - это расширенная версия функции `mysql_fetch_row()`. Помимо хранения значений в массиве с численными индексами, функция возвращает значения в массиве с индексами по названию колонок.

Если несколько колонок в результате будут иметь одинаковые названия, будет возвращена последняя колонка. Чтобы получить доступ к первым, следует использовать численные индексы массива или алиасы в запросе. В случае алиасов именно их вы не сможете использовать настоящие имена колонок, как, например, не сможете использовать "photo" в описанном ниже примере.

```
select Artifacts.photo as art_image,  
    Persons.photo as pers_image  
from Artifacts, Persons
```

Пример 11.1. Запрос с дублирующимися именами колонок

Важно заметить, что `mysql_fetch_array()` работает НЕ медленнее, чем `mysql_fetch_row()`, и предоставляет более удобный доступ к данным.

Второй опциональный аргумент `result_type` в функции `mysql_fetch_array()` является константой и может принимать следующие значения: `MYSQL_ASSOC`,

MYSQL_NUM и MYSQL_BOTH. Эта возможность добавлена в PHP 3.0.7.
Значением по умолчанию является: MYSQL_BOTH.

Используя MYSQL_BOTH, получим массив, состоящий как из ассоциативных индексов, так и из численных. MYSQL_ASSOC вернет только ассоциативные соответствия, а MYSQL_NUM - только численные.

Замечание: имена полей, возвращаемые этой функцией, регистрозависимы.

Теперь отобразим данные из Artifacts в виде таблицы в браузере:

```
<?
/ * сначала делаем то же, что и раньше: устанавливаем
соединение, выбираем базу и получаем список и число полей в таблице Artifacts
*/
$conn=mysql_connect("localhost","nina","123");
$dbase = "book";
$table_name = "Artifacts";
mysql_select_db($dbase);
$list_f = mysql_list_fields($dbase,$table_name);
$num = mysql_num_fields($list_f);
// сохраним имена полей в массиве $names
for($j=0;$j<$num; $j++){
    $names[] = mysql_field_name ($list_f,$j);
}
$sql = "SELECT * FROM $table_name"; // создаем SQL запрос
$q = mysql_query($sql,$conn) or die(); // отправляем
// запрос на сервер
$num = mysql_num_rows($q); // получаем число строк результата
//рисует HTML-таблицу
echo "&nbsp;<TABLE BORDER=0 CELLSPACING=0 width=90%
align=center><tr><TD BGCOLOR='#005533' align=center>
<font color='#FFFFFF'><b>$table_name</b></font></td>
</tr></TABLE>";
echo "<table cellpadding=1 border=1
width=90% align=center>";
// отображаем названия полей
echo "<tr>";
foreach ($names as $val){
    echo "<th ALIGN=CENTER BGCOLOR='#C2E3B6'>
<font size=2>$val</font></th>";
}
// отображаем значения полей
echo "</tr>";
for($i=0;$i<$num; $i++){ // перебираем все строки в
// результате запроса на выборку
echo "<tr>";
foreach ($names as $k => $val) { // перебираем все
// имена полей
$value = mysql_result($q,$i,$val); // получаем
// значение поля
echo "<td><font size=2>&nbsp;$value</font></td>";
// выводим значение поля
}
echo "</tr>";
}
echo "</table>";
```

Листинг 11.1.1. Отображение данных из Artifacts в виде таблицы в браузере
Сделаем то же самое с помощью `mysql_fetch_array()`:

`mysql_query(запрос sql)` – посылает запрос активной в данный момент базе данных.

`mysql_fetch_array(resource result)` – возвращает массив, который соответствует извлеченной строке, и перемещает внутренний указатель данных вперед.

`mysql_affected_rows(resource result)` – определяет число строк, затронутых предыдущей операцией SQL.

`mysql_close(link_identifier)` – закрывает соединение MySQL.

Прежде чем можно будет применять эти функции для создания приложений обработки данных с помощью MySQL, необходимо получить подходящий доступ к серверу MySQL. Для этого требуется учетная запись пользователя и пароль с полномочиями доступа к базе данных и таблицам, содержащим данные, а также имя хоста сервера MySQL или IP-адрес.

При работе с сервером MySQL полезно также использовать инструменты управления с графическим интерфейсом, которые обеспечивают более легкий интерфейс использования данных. Популярными инструментами являются: SQLyog (доступный на <http://www.webyog.com>) и MySQL Administrator (доступный на <http://www.mysql.com/products/tools/>).

Добавление записей

С помощью рассмотренных в предыдущем разделе функций MySQL и языка SQL можно добавлять записи в таблицу базы данных. Записи добавляются с помощью формы, предоставляющей области для ввода информации. Нажатие кнопки вызывает затем сценарий PHP для записи новой информации в таблицу с помощью команды SQL INSERT.

Типичная форма ввода для добавления новой записи в таблицу Survey показана ниже.

VisitorSurvey.php

Name <input type="text"/>	Email <input type="text"/>
Web Connection <input type="text" value="Dial Up"/>	Residence (City/ST/Country) <input type="text"/>
Age <input type="radio"/> Under 20 <input type="radio"/> 21-25 <input type="radio"/> 26-30 <input type="radio"/> Over 30	Gender <input type="radio"/> Male <input type="radio"/> Female
Comments <input type="text"/>	
<input type="button" value="Submit"/> <input type="button" value="Reset"/>	

Поля формы именованы соответствующим образом:

Name - 'Name', Email - 'Email', Web Connection - 'Connection',
Residence (City/ST/Country) - 'Residence', Age - 'Age', Gender - 'Gender',
Comments - 'Comments'.

Прежде чем подробно рассматривать код, будет полезно посмотреть на синтаксис оператора SQL INSERT:

```
INSERT INTO TableName (FieldName1 [,FieldName2]...) VALUES (Value1  
[,Value2]...)
```

Более подробное рассмотрение оператора INSERT дано в приложении.

Следующий код используется для обработки данных формы VisitorSurvey.php:

```
VisitorSurvey.php  
  
<?php  
  
if ($_POST[submit] == "Submit")  
{  
  
//Получение данных формы и присвоение скалярным переменным
```

```

$name = $_POST[Name];
$email = $_POST[Email];
$connection = $_POST[Connection];
$residence = $_POST[Residence];
$age = $_POST[Age];
$gender = $_POST[Gender];
$comments = $_POST[Comments];

//Установка соединения с базой данных

$conn = mysql_connect('localhost','root','хухуху');

//Выбор базы данных MySQL

$db = mysql_select_db('Membership', $conn);

//Оператор SQL

$sql = "INSERT INTO Survey " .
      "(Name,Email,Connection,Residence,Age,Gender,Comments) VALUES ('$Name',
      '$Email', '$Connection',
      '$Residence', '$Age', '$Gender', '$Comments')";

//Выполнение оператора SQL и сохранение результатов в множестве записей

$rs = mysql_query($sql,$conn);

mysql_close($conn);

}

?>

```

Пример 10.1.

После нажатия кнопки отправки Submit создается суперглобальный массив `$_POST`, содержащий значения из формы. Значения массива присваиваются скалярным переменным. Это упрощает кодирование оператора SQL. Затем выполняется оператор `mysql_connect()`. Этот оператор требует три параметра – имя хоста сервера MySQL, имя пользователя, и пароль. Здесь для соединения с базой данных MySQL используется строка соединения без DSN. Ссылка на соединение хранится в переменной `$conn`. Это пример ссылочной переменной PHP. В отличие от скалярных переменных и массивов, ссылочные переменные не применяются непосредственно в программе, но часто используются как параметры для других функций. После соединения с сервером базы данных MySQL следующий шаг состоит в выборе базы данных. Так как экземпляр сервера MySQL может содержать большое число баз данных, то задействуется функция `mysql_select_db` для выбора одной из них для использования в приложении. Эта функция требует два параметра – имя базы данных и ссылку на соединение MySQL. Вслед за выбором базы данных формируется оператор SQL INSERT и присваивается переменной `$sql`. Затем функция `mysql_query()` выполняет оператор SQL, создающий множество записей (множество записей базы данных). Это множество записей присваивается переменной `$rs`, еще одной ссылочной переменной PHP. Отметим, что функция `mysql_query()` требует два параметра — `$sql` (ссылка на текущий оператор SQL) и `$conn` (ссылка на текущее соединение с базой данных). Наконец, вызывается функция `close()` для закрытия текущего соединения с базой данных. Для таких приложений обычно желательно выполнять проверку данных, прежде чем заносить данные в таблицу базы данных. Это

необходимо делать перед установлением соединения с базой данных с помощью методов, рассмотренных в [разделе 6-2](#).

В случае ошибки кодирования данные не заносятся в таблицу базы данных, а PHP выведет предупреждение или сообщение об ошибке. В такой ситуации полезно подавить эти критические сообщения, добавить код для проверки ошибок вручную и сгенерировать более понятный для пользователя вывод. Это можно делать сразу после оператора `mysql_query()`, проверяя статус вновь созданного множества записей — `$rs`.

Предположим, что в предыдущем коде функция `mysql_query()` содержит вместо `$sql` параметр `$sqlString`. В этом случае PHP немедленно прекратит выполнение страницы и выведет сообщение об ошибке.

Вывод сообщения об ошибке PHP можно подавить с помощью оператора управления ошибками `@`. При подавленном сообщении об ошибке можно добавить код для создания более понятного пользователю сообщения. Такой подход показан ниже:

VisitorSurvey.php

```
<?php

if ($_POST[submit] == "Submit")
{

//Получение данных формы и присвоение скалярным переменным

$Name = $_POST[Name];
$email = $_POST[Email];
$connection = $_POST[Connection];
$residence = $_POST[Residence];
$age = $_POST[Age];
$gender = $_POST[Gender];
$comments = $_POST[Comments];

//Установка соединения с базой данных

$conn = @mysql_connect('localhost','root','хухуху');

//Выбор базы данных MySQL

$db = @mysql_select_db('Membership', $conn);

//Оператор SQL

$sql = "INSERT INTO Survey " .
      "(Name,Email,Connection,Residence,Age,Gender,Comments) VALUES ('$Name',
      '$Email', '$Connection',
      '$Residence', '$Age', '$Gender', '$Comments')";

//Выполнение оператора SQL и сохранение результатов в виде множества записей

$rs = @mysql_query($sqlstring,$conn);
```

```

if (!$rs)
{
echo "Произошла ошибка. Попробуйте еще раз.";
}
else
{
echo "Запись была успешно добавлена.";
}
mysql_close($conn);
}
?>

```

Пример 10.2.

Вслед за функцией `mysql_query()` используется оператор `if` для проверки статуса множества записей `$rs`. Если множество записей успешно создается, выводится сообщение "Запись была успешно добавлена". Если возникает проблема, выводится сообщение "Произошла ошибка. Попробуйте еще раз."

Выбор записей

Кроме применения функций MySQL с оператором SQL INSERT для добавления записей в базу данных, можно также извлекать записи из таблицы базы данных с помощью оператора SQL SELECT.

Типичная форма ввода для выбора существующих записей из таблицы Directory показана ниже. В этом примере фиктивная компания, Company XYZ, имеет онлайн форму, которая позволяет пользователям ввести фамилию сотрудника и найти полное имя сотрудника, номер телефона и адрес e-mail.

Company XYZ Directory	
<input type="text"/>	<input type="button" value="Search"/>

Оператор SQL SELECT показан ниже:

```

SELECT * | [DISTINCT] field1 [,field2]... FROM TableName WHERE criteria
ORDER BY fieldName1 [ASC|DESC] [,fieldName2 [ASC|DESC] ]...

```

Более подробно оператор SELECT рассматривается в приложении.

Следующий код используется для обработки формы DirectorySearch.php:

```
DirectorySearch.php

<?php

if ($_POST[submit] == "Search")
{

    //Получение данных формы

    $string = $_POST['search'];

    //Установление соединения с данными

    $conn = mysql_connect('localhost','root','хухуху');

    //Выбор базы данных

    $db = mysql_select_db('Membership', $conn);

    //Создание оператора SQL SELECT

    $sql = "SELECT * FROM Directory WHERE LName = '$string'";

    $rs = mysql_query($sql, $conn);
}

?>

<!DOCTYPE html PUBLIC "-//W3C/DTD/XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml11-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<title>Страница Web </title>

<style>

    body {margin:15px;font:10pt Verdana}
    td {vertical-align:top;border:solid 1px gray}
    input,textarea{border:0px}

</style>

</head>

<body>

<form action="DirectorySearch.php" method="post">
<p>Введите ниже фамилию и нажмите кнопку "Search",
чтобы найти номер телефона и адрес e-mail сотрудника</p>
<table>
<tr>
<td colspan="2">Company XYZ Directory</td>
</tr>
<td><input type="text" size="15" name="SearchName"/></td>
<td><input type="submit" value="Search" name="submit"/>
</td>
</tr>
</table>
</form>
</body>
</html>
```

```

</table>
</form>

<div>

<?php

if(!empty($_POST))
{
    while($row = mysql_fetch_array($rs))
    {
        echo "Name: " . $row['FName'] . " ";
        echo $row['LName'] . "<br/>";
        echo "Telephone: " . $row['Telephone'] . "<br/>";
        echo "Email: " . $row['Email'] . "<br/>";
    }
    mysql_close($conn);
}
?>

</div>
</body>
</html>

```

Пример 10.3.

Эта страница содержит два блока кода PHP. Первый выполняется, когда нажимается кнопка отправки формы "Search". Введенная пользователем фамилия присваивается скалярной переменной '\$string'. Затем создается соединение с базой данных и формируется оператор SQL SELECT для выбора всех (*) полей таблицы записей, в которых поле 'lastname' совпадает со строкой фамилии, введенной пользователем. В конце выполняется оператор SQL. Если найдены подходящие записи, то множество записей присваивается переменной '\$rs'.

Второй блок кода появляется в теле документа XHTML. Код, содержащий операторы echo или print, помещается обычно между открывающим и закрывающим тегами <body>, так что он может выводиться или форматироваться в соответствии с другими элементами страницы. Операторы echo и print, появляющиеся в блоках PHP, закодированные выше тега <html>, всегда появляются вверху страницы и предшествуют всем другим ее элементам.

Назначение этого блока кода состоит в выводе извлеченных записей, если в предыдущем блоке кода был выполнен оператор SQL. Сначала используется оператор if для проверки, что массив \$_POST не является пустым. Если этот массив будет пустым, то это означает, что форма не была отправлена, и никакие записи не были извлечены. Если этот условный оператор отсутствует, то будет возникать ошибка, так как массив mysql_fetch_array не будет содержать никаких значений.

Затем используется цикл while для итераций по множеству записей. Во время каждой итерации по множеству записей функция mysql_fetch_array() создает ассоциативный массив (здесь этот массив назван \$row), содержащий значения полей текущей записи. Индексы массива соответствуют именам полей формы, а элемент массива соответствует значению поля. Каждая запись затем выводит

`$row['FName']` — значение поля 'FirstName', `$row['LName']` -- значение поля 'LastName', `$row['Telephone']` – значение поля 'Telephone', и `$row['Email']` – значение поля 'Email'. Этот процесс продолжается, пока не будет достигнут конец множества записей, при этом массив `$row` каждый раз будет содержать новые значения.

После вывода всех записей соединение с базой данных закрывается с помощью функции `mysql_close()`.

Ниже представлен пример вывода, созданного после поиска в каталоге.

Company XYZ Directory

Name: Molly Douglas
Telephone: 5553
Email: mdouglas@php.net

Name: John Douglas
Telephone: 8883
Email: jdouglas@php.net

Если пользователь ищет фамилию, которая не существует в таблице базы данных, то никакие записи не выводятся. Чтобы предотвратить путаницу, приведенный выше сценарий можно немного изменить так, что будет выводиться соответствующее сообщение, если подходящие записи не будут найдены.

DirectorySearch.php

```
<?php
if(!empty($_POST))
{
    while($row = mysql_fetch_array($rs))
    {
        echo "Name: " . $row['FName'] . " ";
        echo $row['LName'] . "<br/>";
        echo "Telephone: " . $row['Telephone'] . "<br/>";
        echo "Email: " . $row['Email'] . "<br/>";
    }
    if (mysql_affected_rows($rs) == 0)
    {
```

```
        echo "No records found!";
    }

    mysql_close($conn);
}
?>

</div>
</body>
</html>
```

Показанный выше измененный сценарий содержит дополнительно функцию `mysql_affected_rows()`. Эта функция требует один параметр – ссылку на текущее множество записей `$rs` и определяет число строк, затронутых последней операцией SQL. Если возвращаемое функцией `mysql_affected_rows()` значение равно 0, то оператор SQL SELECT не затронул ни одной строки. Поэтому подходящих записей найдено не было.

Удаление записей

Оператор SQL DELETE используется для удаления существующих записей в базе данных.

Синтаксис оператора SQL DELETE показан ниже:

```
DELETE FROM Имя_Таблицы WHERE критерий
```

Более подробно оператор DELETE рассматривается в приложении.

Следующая форма представляет запись пользователя, которая будет удалена из таблицы базы данных Personnel. Щелчок на кнопке Delete вызывает процедуру PHP, которая выполняет оператор SQL DELETE для удаления этой записи из таблицы базы данных.

First Name:	<input type="text" value="Molly"/>
Last Name:	<input type="text" value="Douglas"/>
Telephone:	<input type="text" value="5555"/>
Email:	<input type="text" value="mdouglas@php.net"/>
<input type="button" value="Delete Record"/>	

Кроме показанных выше элементов управления формы, страница включает также скрытое текстовое поле с именем "AutoNum" со значением, равным полю AutoNum

таблицы базы данных. Это поле используется для уникальной идентификации каждой записи. Следующий код демонстрирует, как работает страница:

DirectorySearch.php

```
<?php
if ($_POST['submitb']=="Delete Record")
{
    $conn = mysql_connect('localhost','root','xyxyxy');
    $db = mysql_select_db('Membership',$conn);
    $sqlDelete = "DELETE FROM Personnel WHERE AutoNum = " .
$_POST['AutoNum'];
    $rsDelete = mysql_query($sqlDelete,$conn);

    if(mysql_affected_rows($rsDelete) == 1)
    {
        echo "Запись успешно удалена!";
    }

    mysql_close($conn);
}
?>
```

После нажатия кнопки "Delete Record" устанавливается соединение с базой данных MySQL. Затем создается оператор SQL DELETE для удаления записи из таблицы Personnel со значением поля AutoNum, равным значению скрытого текстового поля AutoNum. Затем оператор SQL выполняется. Результаты работы функции mysql_query() присваиваются переменной \$rsDelete. Последний шаг состоит в проверке, что удаление записи прошло успешно, и в выводе подтверждающего сообщения. Функция mysql_affected_rows() используется для определения числа строк в множестве результатов ODBC или числа строк, затронутых оператором mysql_query(). Так как будет удалена только одна запись, то результат mysql_affected_rows() равный 1 означает, что запись удалена успешно. В конце соединение с базой данных закрывается.

5.1 Литература

<http://mysql.ru/docs/man/>

<http://www.intuit.ru/department/database/mysql/>