

Работа с базой данных MySQL

База данных – совокупность связанных данных, сохраняемая в двумерных таблицах информационной системы. Программное обеспечение информационной системы, обеспечивающее создание, ведение и совместное использование баз данных, называется системой управления базами данных (СУБД). Ниже будут рассмотрены функции PHP, предназначенные для работы с одной из самых популярных СУБД – MySQL. В PHP есть функции для «общения» и с другими системами управления базами данных (например, Sybase, Oracle и т. д.), но мы будем рассматривать именно MySQL в силу ее простоты и универсальности для большинства приложений. Конечно, прежде чем работать с MySQL, нужно установить соответствующее программное обеспечение – программу-сервер MySQL.

Замечание

Данный материал ни в коей мере не претендует на исчерпывающее описание языка SQL и системы управления базами данных MySQL. Здесь приведен только основной минимум материала. Имея его под рукой, можно начинать писать сценарии, использующие MySQL. Если вам понадобится подробная документация, вы сможете найти ее в любом дистрибутиве MySQL.

Итак, с точки зрения программы база данных MySQL представляет собой удачно организованный набор поименованных *таблиц*. Каждая таблица – массив (возможно, очень большой) из однородных элементов, которые будем называть *записями*. В принципе, запись – неделимая единица информации в базе данных, хотя по запросу можно получать и не всю ее целиком, а только какую-то часть.

Запись может содержать в себе одно или несколько именованных *полей*. Число и имена полей задаются при создании таблицы. Каждое поле имеет определенный *тип* (например, целое число, строка текста, массив символов и т. д.).

В таблицу всегда можно добавить новую запись. Другая операция, которую часто производят с записью (точнее, с таблицей) – это поиск. Например, запрос поиска может быть таким: «Выдать все записи, в первом поле которых содержится число, меньшее 10, во втором – строка, включающая слово word, а в третьем – не должен быть ноль». Из найденных записей в программу можно извлекать какие-то части данных (или не извлекать), также записи таблицы можно удалить.

Следует еще раз заметить, что обычно все упомянутые операции осуществляются очень быстро. Например, Microsoft SQL Server может за 0,01 секунды из 10 миллионов записей выделить ту, у которой значение определенного поля совпадает с нужным числом или строкой. Высокое быстродействие в большей мере обусловлено тем, что данные не просто «свалены в кучу», а определенным образом упорядочены и все время поддерживаются в таком состоянии.

Неудобство работы с файлами

Прежде чем мы займемся базами данных MySQL и их поддержкой в PHP, давайте определимся, для чего вообще в Web-программировании могут понадобиться базы данных? Ответ на этот вопрос не вполне очевиден, особенно для людей, сталкивающихся со «стандартными» базами данных впервые.

В самом деле, казалось бы, любой сценарий можно реализовать, основываясь только на работе с файлами. Например, иерархический форум можно хранить в файлах и каталогах: раздел форума – это директория, а конкретный вопрос в нем – файл. Однако ненужная избыточность таких сценариев, мягко говоря, удивляет. Нужно постоянно держать под контролем множество вспомогательных параметров и файлов. Кроме того, крайне усложняется поиск по форуму или создание архива. По правде сказать, работа с файлами – дело нудное и весьма и весьма утомляет.

В противоположность файловой организации хранения информации, использование баз данных дает весомые преимущества. Например, легко сортировать записи по дате/времени, организовывать поиск, различные отборы записей. Правда, многие базы данных не поддерживают иерархические, вложенные таблицы. Но и это не беда: просто достаточно у каждой записи в специальном поле хранить идентификатор ее «родителя», мы вскоре поговорим об этом чуть подробнее.

Базы данных также лишены еще одного крупного недостатка файлов: с ними нет проблем с совместным доступом к данным. Ведь вполне может оказаться, что ваш сценарий запустят два одновременно взглянувших на страничку человека. Конечно, если сценарий обновляет какой-то файл в процессе своей работы, могут возникнуть проблемы, если не принять надлежащих мер по блокировке файла. Кроме того, нужно минимизировать время обновления файла, а это не всегда возможно. С базами данных таких про-

блем не существует, потому что разработчики предусмотрели их (проблем) решение на самом низком уровне и с максимальной эффективностью.

В довершение, чаще всего работа с базами данных происходит быстрее, чем с файлами. В первых обычно предусмотрена эффективная организация хранения информации, минимизирующая время доступа и поиска. Например, вполне реально за приемлемое время найти среди десятков тысяч записей какую-то определенную (скажем, по заданному идентификатору). Или провести поиск по нескольким мегабайтам текста какого-то ключевого слова и обнаружить все записи, которые его содержат.

Устройство MySQL

Одна из самых популярных СУБД, которые используются в Web-программировании, – MySQL. Она предназначена для создания небольших (сравнительно, конечно – скажем, не более 100 Мбайт) баз данных, и поддерживает некоторое подмножество языка запросов SQL.

SQL – специально разработанный стандарт языка запросов к базам данных. В нем присутствуют такие команды, как:

- создание/удаление таблицы;
- создание записей в заданной таблице;
- поиск/удаление записей;
- обновление некоторых полей указанной записи.

Немного подробнее с языком SQL мы будем разбираться чуть позже. А пока давайте посмотрим, что из себя представляет MySQL.

MySQL – это программа-сервер, постоянно работающая на компьютере. Клиентские программы (например, сценарии) посылают ей специальные *запросы* через механизм сокетов (то есть при помощи сетевых средств), она их обрабатывает и запоминает результат. Затем, также по специальному запросу клиента, весь этот результат или его часть передается обратно.

Почему всегда передается не весь результат? Очень просто: дело в том, что размер результирующего набора данных может быть слишком большим, и на его передачу по сети уйдет чересчур много времени. Да и редко когда бывает нужно получать сразу весь вывод запроса (то есть все записи, удовлетворяющие выражению запроса). Например, нам может потребоваться лишь подсчитать, сколько записей удовлетворяет тому или иному условию, или же выбрать из данных только первые 10 записей.

Механизм использования сокетов подразумевает технологию *клиент-сервер*, а это означает, что в системе должна быть запущена специальная программа – MySQL-сервер, которая принимает и обрабатывает запросы от программ. Так как вся работа происходит в действительности на одной машине, накладные расходы по работе с сетевыми средствами незначительны (установка и поддержание соединения с MySQL-сервером обходится довольно дешево).

Как уже было сказано выше, структура MySQL трехуровневая: базы данных – таблицы – записи. Один сервер MySQL может поддерживать сразу несколько баз данных, доступ к которым может разграничиваться логином и паролем. Зная эти логин и пароль, можно работать с конкретной базой данных. Например, можно создать или удалить в ней таблицу, добавить записи и т. д. Обычно имя-идентификатор и пароль назначаются хостинг-провайдерами, которые и обеспечивают поддержку MySQL для своих пользователей.

Соединение с базой данных

Но прежде чем работать с базой данных, необходимо установить с ней сетевое соединение, а также провести авторизацию пользователя. Для этого служит функция `mysql_connect()`.

```
int mysql_connect([string $hostname] [,string $username] [,string $password])
```

Функция `mysql_connect()` устанавливает сетевое соединение с базой данных MySQL, расположенной на хосте `$hostname` (по умолчанию это `localhost`, т. е. текущий компьютер), и возвращает идентификатор открытого соединения. Вся дальнейшая работа ведется именно с этим идентификатором. При регистрации указывается имя пользователя `$username` и пароль `$password` (по умолчанию имя пользователя, от которого запущен текущий процесс, и пустой пароль). Строка `$hostname` также может включать в себя

номер порта в формате: `имя_хоста:порт` (если сервер MySQL настроен не на стандартный, а на какой-то другой порт, что делать, вообще говоря, не рекомендуется).

При следующем запуске функции с теми же самыми аргументами второе соединение не будет открыто, а функция возвратит идентификатор уже существующего. Соединение с MySQL-сервером будет автоматически закрыто по завершении работы сценария, либо же при вызове функции `mysql_close()`. Если вы планируете открывать только одно соединение с базой данных за все время работы сценария, то можете не сохранять возвращенное значение, а также не указывать идентификатор соединения при вызове всех остальных функций.

```
int mysql_select_db(string $dbname [,int $link_identifier])
```

До того как послать первый запрос серверу MySQL, необходимо указать, с какой базой данных мы собираемся работать. Для этого и предназначена описываемая функция. Она уведомляет PHP, что в дальнейших операциях с соединением `$link_identifier` (или с последним открытым соединением, если указанный параметр не задан) будет использоваться база данных `$dbname`.

Обработка ошибок

Если в процессе работы с MySQL возникают ошибки (например, в запросе не сбалансированы скобки или же не хватает параметров), то сообщение об ошибке и ее номер можно получить с помощью следующих двух функций.

```
int mysql_errno([int $link_identifier])
```

Функция возвращает номер последней зарегистрированной ошибки. Идентификатор соединения `$link_identifier` можно не указывать, если за время работы сценария было установлено только одно соединение.

```
string mysql_error([int $link_identifier])
```

Эта функция возвращает не номер, а строку, содержащую текст сообщения об ошибке. Ее удобно применять в отладочных целях.

Выполнение запросов к базе данных

Теперь мы подходим непосредственно к тому, как формировать и посылать запросы к базе данных. Для этого существует одна-единственная функция – `mysql_query()` – и возвращает она не что иное, как идентификатор результирующего набора данных.

Помните, мы говорили, что результат сразу не пересылается клиенту? Так вот, чтобы до него добраться, и служит этот идентификатор. Существует очень много функций, которые принимают его в качестве параметра и возвращают те или иные данные. Их мы рассмотрим чуть позже.

```
int mysql_query(string $query [,int $link_identifier])
```

Эта функция в своем роде универсальна: она посылает MySQL-серверу запрос `$query` и возвращает идентификатор ответа, или результата. Параметр `$query` представляет собой строку, составленную по правилам языка SQL. Используется установленное ранее соединение `$link_identifier`, а в случае его отсутствия – последнее открытое соединение.

Есть несколько команд SQL, которые возвращают только признак, успешно они выполнены или нет (например, это команды `UPDATE`, `INSERT` и т. д.). В таком случае этот признак и будет возвращен функцией. Наоборот, для запроса `SELECT` возвращается как раз идентификатор вывода, нулевое значение которого свидетельствует о том, что произошла ошибка.

На самом деле существует еще одна функция для выполнения запроса, но использовать ее менее удобно, поскольку всякий раз приходится указывать имя базы данных, к которой осуществляется доступ.

```
int mysql(string $dbname, string $query [,int $link_identifier])
```

Служит для тех же целей, что и функция `mysql_query()`, только обращение осуществляется не к текущей выбранной базе данных, а к указанной в параметре `$dbname`. Если вы владеете сразу несколькими базами данных и обращаетесь к ним одновременно, то, возможно, применение этой функции окажется для вас оправданным. Как обычно, параметр `$link_identifier` можно опустить, тогда используется последнее открытое соединение.

Язык запросов MySQL

Разумеется, весь язык запросов SQL в рамках данного материала описать просто невозможно. О нем сочиняют (и будут сочинять) «объемистые» книги. Однако самые основные команды будут приведены.

Все без исключения запросы к базе данных посылаются при помощи одной-единственной функции – `mysql_query()` (или `mysql()`, см. рассуждения выше). Они должны передаваться ей в виде строкового параметра. Этот параметр, впрочем, может быть и многострочным – т. е., содержать символы перевода строки. MySQL допускает включение любого количества пробелов, символов табуляции или перевода строки везде, где разрешено использование одного пробела (в этом смысле он похож на PHP и большинство других языков программирования).

Язык SQL позволяет нам создавать довольно сложные запросы. Ниже перечислены наиболее употребительные команды MySQL.

Создание таблицы

```
create table ИмяТаблицы(ИмяПоля тип, ИмяПоля тип, ...)
```

Этой командой в базе данных создается новая таблица с колонками (полями), определяемыми своими именами (имяПоля) и указанными *типами*.

Типы полей

Ниже будут перечислены практически все типы полей, которые могут применяться в MySQL. Их довольно много. Квадратными скобками будут помечаться необязательные элементы.

Целые числа

Существует несколько разных типов целых чисел, различающихся количеством байтов данных, которые отводятся в базе данных для их хранения. Все эти типы рознятся только названиями и (с некоторыми сокращениями) записываются так:

```
префиксINT [UNSIGNED]
```

Необязательный флаг `UNSIGNED` задает, что будет создано поле для хранения беззнаковых чисел (больших или равных 0). Имена типов, в общем виде обозначенные здесь как префиксINT, приводятся в табл.1.

Таблица 1. Типы целочисленных данных MySQL.

Тип	Описание
TINYINT	Может хранить числа от -128 до +127
SMALLINT	Диапазон от -32 768 до 32 767
MEDIUMINT	Диапазон от -8 388 608 до 8 388 607
INT	Диапазон от -2 147 483 648 до 2 147 483 647
BIGINT	Диапазон от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807

Дробные числа

Точно так же, как целые числа подразделяются в MySQL на несколько разновидностей, MySQL поддерживает и несколько типов дробных чисел. В общем виде они записываются так:

```
ИмяТипа [(length, decimals)] [UNSIGNED]
```

Здесь `length` – количество знаковых мест (ширина поля), в которых будет размещено дробное число при его передаче в PHP, а `decimals` – количество знаков после десятичной точки, которые будут учитываться. Как обычно, `UNSIGNED` задает беззнаковые числа. Строка `ИмяТипа` замещается на предопределенные значения, соответствующие возможным вариантам представления вещественных чисел (табл.2).

Таблица 2. Типы рациональных чисел в MySQL

Тип	Описание
FLOAT	Число с плавающей точкой небольшой точности
DOUBLE	Число с плавающей точкой двойной точности
REAL	Синоним для DOUBLE
DECIMAL	Дробное число, хранящееся в виде строки
NUMERIC	Синоним для DECIMAL

Строки

Строки представляют собой массивы символов. Обычно при поиске по текстовым полям по запросу `SELECT` не берется в рассмотрение регистр символов, т. е. строки «Вася» и «ВАСЯ» считаются одинаковыми. Кроме того, если база данных настроена на автоматическую перекодировку текста при его помещении и извлечении, эти поля будут храниться в указанной вами кодировке.

Для начала давайте ознакомимся с типом строки, которая может хранить не более `length` символов, где `length` принадлежит диапазону от 1 до 255.

```
VARCHAR(length) [BINARY]
```

При занесении некоторого значения в поле такого типа из него автоматически вырезаются концевые пробелы (как будто по вызову функции `rtrim()`). Если указан флаг `BINARY`, то при запросе `SELECT` строка будет сравниваться с учетом регистра. Тип `VARCHAR` неудобен тем, что может хранить не более 255 символов. Вместо него рекомендуется использовать другие текстовые типы, перечисленные в табл.3.

Таблица 3. Строковые типы данных таблиц MySQL

Тип	Описание
TINYTEXT	Может хранить максимум 255 символов
TEXT	Может хранить не более 65 535 символов
MEDIUMTEXT	Может хранить максимум 16 777 215 символов
LONGTEXT	Может хранить 4 294 967 295 символов

Чаще всего применяется тип `TEXT`, но если вы не уверены, что данные не будут всегда короче 65 536 байтов, используйте `LONGTEXT`.

Замечание

Слухи о том, что `TEXT`-типы занимают намного больше места, чем аналогичные `VARCHAR`-поля, сильно преувеличены.

Бинарные данные

Бинарные данные – это почти то же самое, что и данные в формате `TEXT`, но только при поиске в них учитывается регистр символов («abc» и «ABC» – разные строки). Всего имеется 4 типа бинарных данных (табл.4).

Таблица 4. Типы бинарных данных, используемые в MySQL

Тип	Описание
TINYBLOB	Может хранить максимум 255 символов
BLOB	Может хранить не более 65 535 символов
MEDIUMBLOB	Может хранить максимум 16 777 215 символов
LONGBLOB	Может хранить 4 294 967 295 символов

`BLOB`-данные не перекодируются автоматически, если при работе с установленным соединением включена возможность перекодирования текста «на лету».

Дата и время

MySQL поддерживает несколько типов полей, специально приспособленных для хранения дат и времени в различных форматах (табл.5).

Таблица 5. Представление дат и времени в базах данных MySQL

Тип	Описание
DATE	Дата в формате ГГГГ-ММ-ДД
TIME	Время в формате ЧЧ:ММ:СС
DATETIME	Дата и время в формате ГГГГ-ММ-ДД ЧЧ:ММ:СС
TIMESTAMP	Время и дата в формате TIMESTAMP. Однако при получении значения поля оно отображается не в формате TIMESTAMP, а в виде ГГГГММДДЧЧММСС, что сильно уменьшает преимущества его использования в PHP

Надо заметить, что в PHP будет проще самостоятельно генерировать дату и время при вставке данных в таблицу, а не задействовать встроенные в MySQL типы. Например, привлекательный с виду тип TIMESTAMP на деле оказывается довольно неудобным, потому что отображается не в том виде, который мы ожидаем.

Перечисления и множества

MySQL поддерживает еще несколько специфических типов данных, использовать которые в PHP вряд ли целесообразно. Например, тип перечисления задает, что значение соответствующего поля может быть не любой строкой или числом, а только одним из нескольких указанных при создании таблицы значений: value1, value2 и т. д. Вот как выглядит имя типа перечисления:

```
ENUM(value1,value2,value3,...)
```

В отличие от всех остальных типов, множества означают, что в соответствующем поле может содержаться не одно, а сразу несколько значений (value1, value2 и т. д., т. е. — множество значений). Формат задания данных такого типа имеет следующий вид:

```
SET(value1,value2,value3,...)
```

Замечание

Значений в множестве может быть не сколько угодно, а не более 64 штук. Иногда это сильно мешает при программировании.

Модификаторы и флаги типов

К типу можно также присоединять модификаторы, которые задают его «поведение» и те операции, которые можно (или, наоборот, запрещено) выполнять с соответствующими столбцами. Самые распространенные из них сведены в табл.6.

Таблица 6. Основные модификаторы MySQL

Модификатор	Описание
not null	Означает, что поле не может содержать неопределенное значение — в частности, поле обязательно должно быть инициализировано при вставке новой записи в таблицу (если не задано значение по умолчанию)
primary key	Отражает, что поле является первичным ключом, т. е. идентификатором записи, на которой можно ссылаться
auto_increment	При вставке новой записи поле получит уникальное значение, так что в таблице никогда не будут существовать два поля с одинаковыми номерами. (Мы поговорим об этом чуть позже.)
Default	Задаёт значение по умолчанию для поля, которое будет использовано, если при вставке записи поле не было проинициализировано явно

Удаление таблицы

```
drop table ИмяТаблицы
```

Удаляет таблицу `ИмяТаблицы`. Таблица не обязательно должна быть пустой, так что будьте внимательны, чтобы случайно не «аннулировать» нужную таблицу с данными.

Вставка записи

```
insert into ИмяТаблицы(ИмяПоля1 ИмяПоля2 ...) values('зн1','зн2',...)
```

Добавляет в таблицу `ИмяТаблицы` запись, у которой поля, обозначенные как `ИмяПоляN`, установлены в значения, соответственно, `знN`. Те поля, которые в этой команде не перечислены, получают «неопределенные» значения (неопределенное значение – это не пустая строка, а просто признак, который говорит MySQL, что у данного поля нет никакого значения). Впрочем, если для не указанного здесь поля при создании таблицы был задан `not null`, то данная команда закончится неуспешно. Значения полей можно заключать и в обычные кавычки, но, пожалуй, апострофы тут использовать удобнее. При вставке в таблицу бинарных данных (или текстовых, содержащих апострофы и слэши) некоторые символы должны быть «защищены» обратными слэшами, а именно, символы `\`, `'` и символ с нулевым кодом.

Удаление записей

```
delete from ИмяТаблицы where Выражение
```

Удаляет из таблицы `ИмяТаблицы` все записи, для которых выполнено выражение. Параметр `Выражение` – это просто логическое выражение, составленное почти что по правилам РНР. Вот показательный пример:

```
(id<10) and (name regexp 'a*b') and (age=25)
```

В выражении, помимо имен полей, констант и операторов, могут также встречаться простейшие «вычисляемые» части, например: `(id<10+11*234)`.

Вообще говоря, формат выражения един для всех команд запросов, которые будут приведены ниже. Например, он же используется и в операции `select`, и в операции `update`.

Поиск записей

```
select * from Таблица where Выражение [order by ИмяПоля [desc]]
```

Эта команда – основная и очень мощная. Предназначена она для того, чтобы искать все записи, удовлетворяющие выражению `Выражение`. Ее возможности гораздо более богаты, чем то сжатое изложение, которое здесь приведено, и о них можно прочитать в книгах, посвященных SQL. Если записей несколько, то при указанном предложении `order by` они будут отсортированы по тому полю, имя которого записывается правее этого ключевого слова (если задан описатель `desc`, то упорядочивание происходит в обратном порядке). В предложении `order by` могут также задаваться несколько полей.

Особое значение имеет символ `*`. Он предписывает, что из отобранных записей следует извлечь *все* поля, когда будет выполнена команда получения выборки. С другой стороны, вместо звездочки можно через запятую непосредственно перечислить имена полей, которые требуют извлечения. Но чаще всего все же используется именно `*`.

Обновление записей

```
update Таблица set(ИмяПоля1='зн1', ИмяПоля1='зн2', ...) where Выражение
```

В таблице `таблица` для всех записей, удовлетворяющих выражению `Выражение`, указанные поля устанавливаются в соответствующие значения. Эта команда часто отдается, если не требуется обновлять сразу все поля какой-то записи, а нужно затронуть только некоторые.

Получение числа записей, удовлетворяющих выражению

Стоит рассмотреть еще одну часто востребуемую возможность MySQL – получение числа записей, удовлетворяющих некоторому выражению. Вообще говоря, существует несколько способов сделать это. Вот один из них:

```
select count(if(Выражение,1,NULL)) from Таблица
```

Уже этот пример показывает, насколько богаче язык MySQL по сравнению с тем, что было описано...

Получение уникальных значений столбцов

При использовании базы данных часто бывает крайне удобно узнать, какие *уникальные* значения существуют в данном столбце таблицы. Например, если у каждой записи в некоей статистической таблице, содержащей сведения о людях, у нас есть поле Country (страна), в котором указана страна проживания конкретного человека, и мы хотим выяснить, в каких же странах проживают все люди, дожившие до 30 лет, занесенные в таблицу, можно выполнить запрос:

```
select distinct ИмяПоля from Таблица where Выражение
```

В нашем случае ИмяПоля=Country, а Выражение – что-то вроде age>=30.

Этот запрос сгенерирует результат, состоящий из одного столбца, в котором и будут перечислены искомые страны.

Получение результата

После того как запрос выполнен и идентификатор результирующего набора данных возвращен, вы, возможно, захотите получить этот самый результат. Поговорим о том, что же он из себя представляет.

Результат – это просто *набор данных*, и количество вошедших в него записей можно узнать через `mysql_num_rows()`. Например, если в предыдущем примере при выборке из таблицы оказалось, что в таблице имеются записи о 10 людях старше 30 лет, то мы в идентификаторе результата получим «ссылку» на 10 «строчек». Теперь мы можем считать в программу на PHP любую из них с помощью специальных функций, которые будут описаны ниже.

Каждая запись – это *список значений полей*, а именно, тех полей и в том же порядке, которые были указаны в запросе `select ... from Таблица` на месте многоточия (если там была звездочка, то все поля). Таким образом, результат – это такой своеобразный двумерный массив: первый индекс – номер записи и второй – имя поля. Можете называть его прямоугольной таблицей или матрицей данных – как угодно.

Параметры результата

```
int mysql_num_rows(int $result)
```

Функция `mysql_num_rows()` возвращает число записей в результате запроса. Таким образом, функция позволяет определить вертикальную размерность «двумерного массива результата».

```
int mysql_num_fields(int $result)
```

Эта функция возвращает число полей в одной строке результата, т. е., число колонок в результате `$result`. В силу сказанного, функция позволяет определить горизонтальную размерность «двумерного массива результата».

Получение поля результата

```
int mysql_result(int $result, int $row, mixed $field)
```

Функция возвращает значение поля `$field` в строке результата с номером `$row`. Параметр `$field` может задавать не только имя поля, но и его номер – позицию, на которой столбец «стоял» при создании таблицы. Тем не менее, рекомендуется везде, где это только возможно, использовать именно имена полей.

Примечание

Если мы опять будем рассматривать результат как двумерный массив полей, то параметр `$field` надо поставить в соответствие его X-координате, а `$row` – Y-координате. В этом понимании X-координата чаще всего будет *ассоциативной* – т. е. в ней задается не число, а имя столбца.

Функция универсальна: с ее помощью можно «обойти» весь результат по одной ячейке. И хотя это не возбраняется, но делать, однако, не рекомендуется, т. к. `mysql_result()` работает довольно медленно. Лучше воспользоваться функциями, которые описываются дальше.

Получение целой строки результата

Разработчики MySQL предусмотрели другой, более быстрый, способ получения результата. Он чем-то похож на работу с файлами: появляется понятие текущей записи результата, и следующая операция считывания передвигает этот указатель на одну позицию вперед. Также можно установить указатель на любую указанную запись.

```
array mysql_fetch_row(int $result)
```

Функция возвращает массив-список со значениями полей очередной строки результата `$result`. Если указатель текущей позиции результата был установлен за последней записью (то есть строки кончились), возвращает `false`. Текущая позиция сдвигается к следующей записи, так что очередной вызов `mysql_fetch_row()` вернет следующую строку результата.

Эту функцию чаще всего применяют в таком контексте:

```
$r=mysql_query("select * from OurTable where age<30");
while($Row=mysql_fetch_row($r)) {
    // обрабатываем строку $Row
}
```

Как видим, цикл оборвется, как только строки закончатся, т. е. когда `mysql_fetch_row()` вернет `false`.

Работать с числовыми индексами полей, как до сих пор предлагалось, согласитесь, не очень-то удобно. Гораздо предпочтительнее было бы использовать для адресации поля внутри результата его имя. Функция `mysql_fetch_array()` как раз и извлекает из результата очередную запись и помещает ее в ассоциативный массив.

```
array mysql_fetch_array(int $result)
```

Функция `mysql_fetch_array()` возвращает очередную строку результата в виде ассоциативного массива, где каждому полю сопоставлен элемент с ключом, совпадающим с именем поля. Дополнительно в массив записываются элементы с числовыми ключами и значениями, соответствующими величинам полей с этими индексами. В возвращаемом массиве они размещаются сразу за элементами с «обычными» ключами.

Примечание

Может возникнуть вопрос: зачем вообще тут нужны числовые индексы. Ответ прост: дело в том, что в результате выборки в действительности могут присутствовать поля (фактически, колонки) с одинаковыми именами, но, соответственно, с различными индексами. Это происходит тогда, когда выборка в `SELECT` производится одновременно из нескольких таблиц (язык SQL это позволяет). Но, как правило, в простейших приложениях такое случается нечасто.

Рекомендуется всегда вместо `mysql_fetch_row()` использовать функцию `mysql_fetch_array()`, потому что она более универсальна и к тому же, как написано в документации, не намного медленнее.

```
int mysql_data_seek(int $result, int $row_number)
```

Эта функция устанавливает указатель текущей строки в результате `$result` в позицию `$row_number`, так что следующий вызов `mysql_fetch_row()` и `mysql_fetch_array()` вернет значения полей именно этой строки. Возвращает `false` в случае ошибки или если строки кончились.

Получение информации о результате

Будет полезно рассмотреть еще несколько функций, предназначенных для получения различной информации о результате запроса.

```
string mysql_field_name(int $result, int $field_index)
```

Функция `mysql_field_name()` возвращает *имя* поля, которое расположено в результате по смещению `$field_index`. В общем-то, применяется довольно редко, что связано с существованием функции `mysql_fetch_array()`.

Примечание

Итак, с помощью функции `mysql_field_name()` можно «переводить» числовые X-координаты в двумерном массиве результата в их ассоциативные эквиваленты.

```
string mysql_field_type(int $result, int $field_offset)
```

Эта функция похожа на `mysql_field_name()`, только возвращает не имя, а *тип* соответствующей колонки в результате. Им может быть, например, `int`, `double` и т.д.

```
int mysql_field_len(int $result, int $field_offset)
```

Функция возвращает *длину* поля в результате `$result`. Поле, как обычно, задается указанием его смещения. Под длиной здесь подразумевается не размер данных поля в байтах, а тот размер, который был указан при его создании. Например, если поле имеет тип `varchar` и было создано (вместе с таблицей) с типом `varchar (100)`, то для него будет возвращено `100`.

Примечание

Описание подробностей выходит за рамки этого материала. За детальными разъяснениями необходимо обратиться к какой-нибудь книге о языке запросов SQL.

```
string mysql_field_flags(int $result, int $field_offset)
```

Эта функция возвращает флаги, которые были использованы при создании указанного поля в таблице. Возвращаемая строка представляет собой набор слов, разделенных пробелами, так что вы можете преобразовать ее в массив при помощи функции `explode()`:

```
$Flags=explode(" ",mysql_field_flags($r,$field_offset));
```

Флаги, которые поддерживаются MySQL в настоящий момент, перечислены в табл. 7.

Таблица 7. Флаги типов полей

Флаг	Описание
<code>not_null</code>	Поле обязательно должно быть проинициализировано при вставке очередной строки в таблицу
<code>Primary_key</code>	Поле является первичным ключом – т.е. идентификатором строки, который будет использован для ссылок на нее
<code>Onique_key</code>	Поле должно быть уникальным
<code>Multiple_key</code>	По этому полю построен индекс
<code>Blob</code>	Поле может содержать бинарный блок данных, который никак не интерпретируется
<code>Unsigned</code>	Поле содержит беззнаковые числа
<code>zerofill</code>	Использовать символы с нулевым кодом вместо пробелов
<code>binary</code>	Поле содержит бинарные данные
<code>enum</code>	Поле содержит элемент перечисления, т. е. только один элемент из нескольких возможных
<code>auto_increment</code>	Это поле автоматически нумеруется. Проставляется MySQL при добавлении новой записи так, чтобы в таблице никогда не образовывалось нескольких строк с одинаковым значением этого поля
<code>timestamp</code>	В поле динамически проставляется текущее время при добавлении или изменении записи

```
string mysql_field_table(int $result, int $field offset)
```

Возвращает имя таблицы, из которой было извлечено поле со смещением `$field offset` в результате `$result`. Как уже говорилось, результат запроса может быть получен из нескольких таблиц, так что вот вам средство для извлечения имени таблицы по номеру поля.

Пример использования функций поддержки MySQL

Ниже приводится пример комплексного использования описанных функций.

```
<?
// Соединяемся с сервером на локальной машине
mysql_connect("localhost");

// Выбираем текущую базу данных
mysql_select_db("my_database");

// Получаем все данные таблицы
$result = mysql_query("SELECT * FROM tbl");

// Запрашиваем идентификатор данных о полях таблицы
$fields = mysql_num_fields ($result);

// Узнаем число записей в таблице
$rows = mysql_num_rows($result);

// Получаем имя таблицы (правда, мы его и так знаем, но все же...)
$table = mysql_field_table($result, 0);

echo "Таблица '$table' содержит $fields колонок и $rows строк<BR>";
echo "Таблица содержит следующие поля:<BR>";

// "Проходимся" по всем полям и выводим информацию о них
for($i=0; $i<$fields; $i++) (
$type = mysql_field_type($result, $i) ;
$name = mysql_field_name($result, $i);
$len = mysql_field_len($result, $i) ;
$flags = mysql_field_flags($result, $i) ;
echo "$type $name $len $flags<BR>\n";
```

Уникальные идентификаторы в MySQL

Обычно в таблице содержится довольно много записей с разными значениями полей. Встает проблема выбора одной конкретной записи из этого массива. В рассмотренном нами примере таблицы с информацией о гражданах, пожалуй, запись можно однозначно идентифицировать по фамилии человека. Ну а если встретятся однофамильцы? Тогда по имени. А если же и имена совпадут? Ну, тогда...

Вы видите, насколько это все неудобно. Поэтому во избежание недоразумений подобного рода в таблицу вводят еще одно вспомогательное поле (колонок), назвав ее, скажем, `id`. Этот самый `id` уникален у каждой записи, поэтому мы можем, зная `id` нужного нам человека, тут же получить его данные. Кроме того, если нам понадобится, например, зафиксировать в таблице еще и родственные связи людей (кто является чьим отцом, например), мы можем завести в ней еще одно поле – `parent_id`, в котором будет храниться `id` родителя конкретного человека. Таким образом, описанная техника оказывается довольно удобной.

Пусть теперь мы хотим добавить в таблицу сведения о еще одном человеке. Логично было бы, чтобы его `id` проставлялся автоматически. Возникает вопрос: как нам этот `id` вычислить? В самом деле, мы же не знаем, какие `id` в таблице в данный момент «свободны»... Можно использовать для этой цели текущее время в секундах. Но вдруг именно в данную секунду кто-то еще точно так же добавляет в базу данных запись? Можно, конечно, взять максимальный `id`, прибавить к нему единичку и занести в таблицу. Но где гарантия, что, опять же в этот момент другой администратор не проделал ту же операцию – до того, как вы добавили свою информацию, но после того, как определили максимальный `id`?

Как раз для решения этой проблемы и предназначена в MySQL возможность под названием `AUTO_INCREMENT`. А именно, при создании таблицы мы можем какое-нибудь ее поле (в нашем случае как раз `id`) объявить так:

```
ИмяПоля int auto_increment primary key
```

Немного длинновато, но это стоит того! Теперь любая операция `INSERT` автоматически проставит у добавленной записи поле `ИмяПоля` так, чтобы оно было уникально во всей таблице – MySQL это гарантирует. В простейшем случае – просто увеличит на 1 некий внутренний счетчик, глобальный для всей таблицы, и занесет его новое значение в нужное поле записи. Причем гарантируется, что никакие проблемы с совместным доступом к таблице просто не могут возникнуть, как это произошло бы, используй мы «кустарные» способы.

Получить только что вставленный идентификатор можно при помощи функции `mysql_insert_id()`.

```
int mysql_insert_id([int $link_identifier])
```

Функция возвращает непосредственно перед ее вызовом сгенерированный идентификатор записи для автоинкрементного поля после выполнения команды `INSERT`. Вызывать ее разумно только сразу после выполнения инструкции `insert`, например, в таком контексте:

```
mysql_query("insert into Таблица(поле1, поле2) values('aa','bb')");  
$id=mysql_insert_id();
```

Теперь к только что вставленной записи можно обратиться, используя идентификатор `$id`:

```
$r=mysql_query("select * from Таблица where id=$id");  
$Row=mysql_fetch_array($r);
```

Работа с таблицами

```
int mysql_list_fields(string $dbname, string $tblname [,int $link])
```

Функция `mysql_list_fields()` возвращает информацию об указанной таблице `$tblname` в базе данных `$dbname`, используя идентификатор соединения `$link`, если он задан (в противном случае – последнее открытое соединение). Возвращаемое значение – идентификатор результата, который может быть проанализирован обычными средствами, либо при помощи функций `mysql_field_flags()`, `mysql_field_len()`, `mysql_field_name()` и `mysql_field_type()`. В случае ошибки возвращается `-1`, текст сообщения ошибки может быть получен обычным способом.

```
int mysql_list_tables(string $database [,int $link_identifier])
```

Функция возвращает идентификатор результата (одна колонка), в котором содержатся имена всех таблиц, присутствующих в базе данных. Для извлечения этих имен можно использовать функцию `mysql_result()` с номером колонки, равным `0`.