Методические указания по теме
“Работа с данными в PHP - часть 1”

Базы данных

Базы данных занимают центральное место во многих веб-приложениях. В базе данных можно хранить практически любую информацию, которую вы планируете использовать для поиска и обновления: списки пользователей, каталоги продуктов, заголовки новостей и т. д. Одной из причин, по которым PHP так хорошо подходит для веб-программирования, является всесторонняя поддержка баз данных. PHP может взаимодействовать практически с любыми базами данных — как с реляционными, так и с другими. Также в PHP реализована поддержка ODBC, и даже если ваша любимая база данных не входит в список, но поддерживает ODBC, ее все равно можно будет использовать с PHP. Но для начала давай разберемся, что такое база данных.

База данных — это структурированная коллекция записей или данных, хранящихся в компьютерной системе и организованных так, что можно осуществлять быстрый поиск и извлечение нужной информации.

В названии MySQL составляющая SQL означает Structured Query Language — язык структурированных запросов. Если характеризовать его в общих чертах, то это язык, основанный на словах английского языка и используемый также в других системах управления базами данных, например Oracle и Microsoft SQL Server. Он разработан для предоставления возможности создания простых запросов к базе данных посредством команд следующего вида:

```
SELECT title FROM publications WHERE author = 'Charles Dickens';
```

В базе данных MySQL имеются одна или несколько таблиц, каждая из которых состоит из записей или строк. Внутри строк находятся разные столбцы или поля, в которых и содержатся данные. В таблице ниже показана учебная база данных, в которой присутствует информация о пяти книгах, структурированная по авторам, названиям, категориям и годам издания.

Автор	Название	Категория	Год
Марк Твен	«Приключения Тома Сойера»	Художественная	1876
Джейн Остен	«Гордость и предубеждение»	Художественная	1811
Чарльз Дарвин	«Происхождение видов»	Научная	1856
Чарльз Диккенс	«Лавка древностей»	Художественная	1841
Вильям Шекспир	«Ромео и Джульетта»	Пьеса	1594

Каждая строка таблицы подобна строке в таблице MySQL, и каждый элемент в этой строке подобен полю MySQL. Как вы уже заметили, все эти издания относятся к классической литературе, поэтому таблицу в базе данных, содержащую сведения о них, я буду называть *classics*.

Сводка понятий, используемых в базах данных

Основными понятиями, с которыми следует ознакомиться на данном этапе, являются:

- **база данных** — контейнер для всей коллекции данных MySQL;
- **таблица** — вложенный в базу данных контейнер, в котором хранятся сами данные;
- **строка** — отдельная запись, в которой могут содержаться несколько полей;
- **столбец** — имя поля внутри строки.

Доступ к MySQL из командной строки

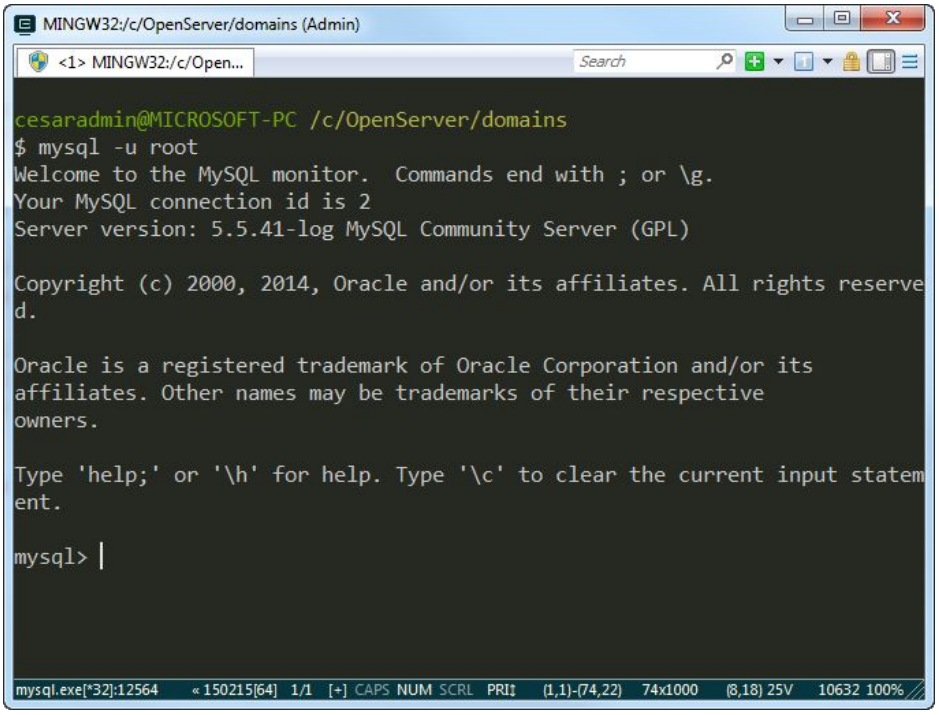
Работать с MySQL можно тремя основными способами: используя командную строку, применяя веб-интерфейс наподобие phpMyAdmin и задействуя такой язык программирования, как PHP.

Для пользователей Windows

Если у вас установлена программа Open Server, то доступ к исполняемой программе MySQL можно получить из следующих каталогов:

```
C:\OpenServer\modules\database\MySQL-5.1\bin
C:\OpenServer\modules\database\MySQL-5.5\bin
C:\OpenServer\modules\database\MySQL-5.6\bin
```

Если вам не удобно каждый раз вбивать данный путь в командной строке, то их можно прописать в переменной окружения PATH.



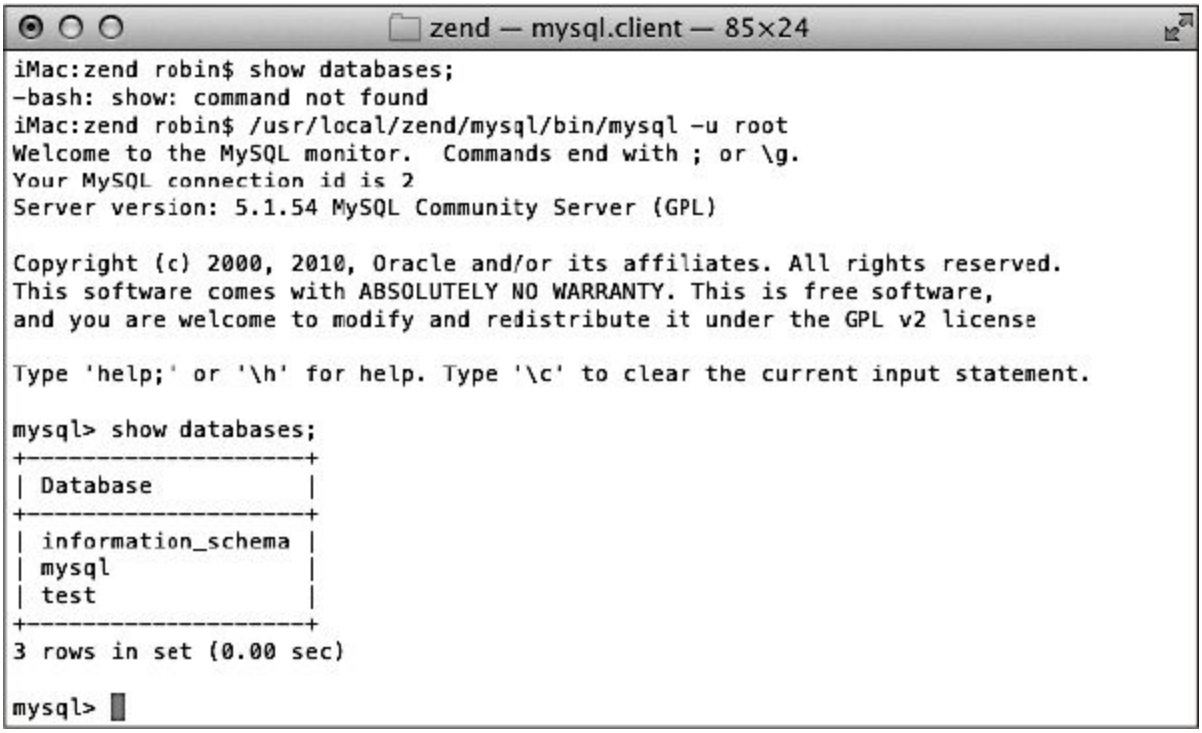
Для пользователей Mac OS X

Если у вас установлена программа Zend Server, то для входа в интерфейс командной строки MySQL необходимо запустить программу Terminal (доступную в меню Utilities программы Finder). Затем вызвать программу MySQL, которая должна быть установлена в каталоге `/usr/local/zend/mysql/bin`.
По умолчанию исходным пользователем для MySQL будет пользователь по имени root, и пароль у него также будет root. Поэтому для запуска программы наберите следующую команду:

```
/usr/local/zend/mysql/bin/mysql -u root
```

Эта команда предпишет MySQL зарегистрировать вас как пользователя root и запросить пароль. Чтобы проверить, что все в порядке, наберите следующую команду:

```
SHOW databases;
```



Работа в консоли

Точка с запятой

Начнем с самого простого. Набирая команду, вы наверняка заметили точку с запятой (;)Этот символ используется в MySQL для завершения команд или отделения их друг от друга. Если забыть поставить этот символ, MySQL выдаст приглашение и будет ожидать от вас его ввода. Запрашиваемая точка с запятой стала частью синтаксиса, позволяющего вводить длинные команды, разбивая их на несколько строк. Она также позволяет вводить сразу несколько команд, после каждой из которых стоит точка с

запятой. После нажатия клавиши Enter интерпретатор получит все эти команды в едином пакете и выполнит их в порядке следования.

На экране могут появляться шесть разных приглашений MySQL, позволяющих определить, на каком именно этапе многострочного ввода вы находитесь.

Приглашение MySQL	Значение
mysql>	Готова к работе и ждет ввода команды
->	Ожидание следующей строки команды
'>	Ожидание следующей строки строкового значения, которое начиналось с одинарной кавычки
">	Ожидание следующей строки строкового значения, которое начиналось с двойной кавычки
`>	Ожидание следующей строки строкового значения, которое начиналось с символа засечки (`)
/*>	Ожидание следующей строки комментария, который начинался с символов /*

Отмена команды

Если, набрав часть команды, вы решили, что ее вообще не следует выполнять, то ни в коем случае не пользуйтесь сочетанием Ctr+C! Оно закроет программу. Вместо нее можно ввести символы `\c` и нажать клавишу Enter.

При наборе этой строки MySQL проигнорирует все ранее введенные символы и выдаст новое приглашение. Без \c программа выведет сообщение об ошибке. Но этой парой символов нужно пользоваться с оглядкой: если у вас уже есть открытая строка или комментарий, то прежде чем применить \c, вам придется их закрыть, иначе MySQL примет \c за часть строки.

Следует также заметить, что комбинация \c после точки с запятой работать не будет, поскольку это уже будет новая инструкция.

Команды MySQL

Нам уже приходилось встречаться с командой SHOW, которая выводит список таблиц, баз данных и многих других элементов. В таблице приведен перечень наиболее востребованных команд.

Команда	Действие
ALTER	Внесение изменений в базу данных или таблицу
BACKUP	Создание резервной копии таблицы
\c	Отмена ввода
CREATE	Создание базы данных
DELETE	Удаление строки из таблицы
DESCRIBE	Описание столбцов таблиц
DROP	Удаление базы данных или таблицы
EXIT (Ctrl+C)	Выход
GRANT	Изменение привилегий пользователя
HELP (\h, \?)	Отображение подсказки
INSERT	Вставка данных
LOCK	Блокировка таблицы (таблиц)
QUIT (\q)	То же самое, что и EXIT
RENAME	Переименование таблицы
SHOW	Список сведений об объекта
SOURCE	Выполнение команд из файла
STATUS (\s)	Отображение текущего состояния
TRUNCATE	Опустошение таблицы
UNLOCK	Снятие блокировки таблицы (таблиц)
UPDATE	Обновление существующей записи
USE	Использование базы данных

- Команды и ключевые слова SQL нечувствительны к регистру. Все три команды — CREATE, create и CrEaTe — абсолютно идентичны по смыслу. Но чтобы было понятнее, для команд рекомендуется использовать буквы верхнего регистра.
- Имена таблиц нечувствительны к регистру в Windows, но чувствительны к регистру в Linux и Mac OS X. Поэтому из соображений переносимости нужно всегда выбирать буквы одного из регистров и пользоваться только ими. Для имен таблиц рекомендуется использовать буквы нижнего регистра или комбинацию из букв верхнего и нижнего регистра.

Создание базы данных

Если вы работаете на удаленном сервере, у вас только одна учетная запись пользователя и вы имеете допуск только к одной созданной для вас базе данных, то можете перейти к изучению пункта «Создание таблицы» далее. А если это не так, то продолжим, введя следующую команду для создания новой базы данных по имени publications:

```
CREATE DATABASE publications;
```

При успешном выполнении команды будет выведено сообщение, пока не имеющее для нас особого смысла, — `Query OK, 1 row affected` (0.38 sec) (Запрос выполнен, обработана 1 строка за 0,38 с), но вскоре все станет на свои места. После создания базы данных с ней нужно будет работать, поэтому даем следующую команду:

```
USE publications;
```

Теперь должно быть выведено сообщение об изменении текущей базы данных (`Database changed`), и после этого база будет готова к продолжению работы со следующими примерами

Создание таблицы

Теперь наберите построчно команды, которые приведены в примере:

```
CREATE TABLE classics (
  author VARCHAR(128),
  title VARCHAR(128),
  type VARCHAR(16),
  year CHAR(4)) ENGINE MyISAM;
```

Чтобы проверить факт создания новой таблицы, наберите команду:

```
DESCRIBE classics;
```

Если все в порядке, то вы увидите последовательность команд и ответов, показанных в примере:

```
mysql> USE publications;
Database changed
mysql> CREATE TABLE classics (
    ->  author VARCHAR(128),
    ->  title VARCHAR(128),
    ->  type VARCHAR(16),
    ->  year CHAR(4)) ENGINE MyISAM;
Query OK, 0 rows affected (0.03 sec)
mysql> DESCRIBE classics;
```

```
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| author | varchar(128) | YES  |     | NULL    |       |
| title  | varchar(128) | YES  |     | NULL    |       |
| type   | varchar(16)  | YES  |     | NULL    |       |
| year   | char(4)      | YES  |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

Команда DESCRIBE является неоценимым средством отладки, когда нужно убедиться в успешном создании таблицы MySQL. Этой командой можно воспользоваться также для того, чтобы просмотреть имена полей или столбцов таблицы и типы данных в каждом из них. Рассмотрим подробнее все заголовки:

- Field — имя каждого из полей или столбцов таблицы;
- Type — тип данных, сохраняемых в поле;
- Null — заголовок, который показывает, может ли поле содержать значение NULL;
- Key — MySQL поддерживает ключи, или индексы, позволяющие ускорить просмотр и поиск данных. Под заголовком Key показан тип применяемого ключа (если таковой имеется);
- Default — исходное значение, присваиваемое полю, если при создании новой строки не указано никакого значения;
- Extra — дополнительная информация, например, о настройке поля на автоматическое приращение его значения.

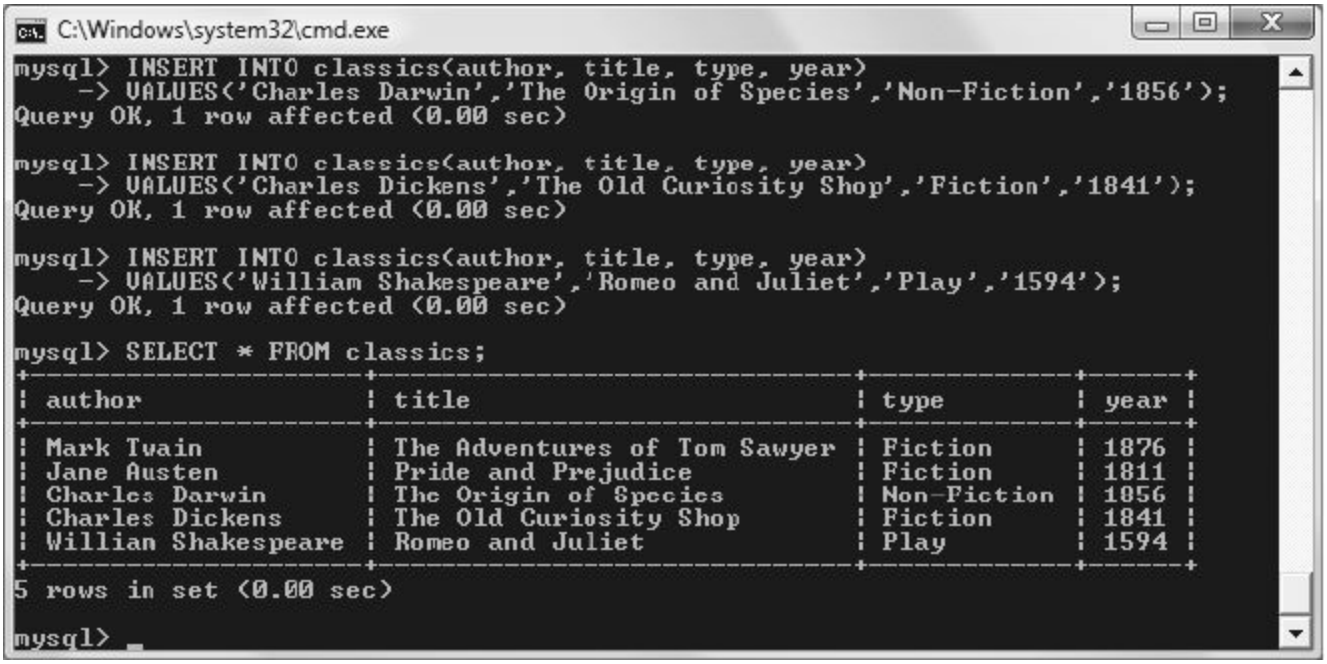
Добавление данных к таблице

Для добавления данных к таблице предназначена команда *INSERT*. Рассмотрим ее в действии, заполнив таблицу *classics* данными.

```
INSERT INTO classics(author, title, type, year) VALUES('Mark Twain','The
Adventures of Tom Sawyer','Fiction','1876');
INSERT INTO classics(author, title, type, year) VALUES('Jane Austen','Pride and
Prejudice','Fiction','1811');
INSERT INTO classics(author, title, type, year) VALUES('Charles Darwin','The
Origin of Species','Non-Fiction','1856');
INSERT INTO classics(author, title, type, year) VALUES('Charles Dickens','The
Old Curiosity Shop','Fiction','1841');
INSERT INTO classics(author, title, type, year) VALUES('William
Shakespeare','Romeo and Juliet','Play','1594');
```

После каждой второй строки вы должны увидеть сообщение об успешной обработке запроса — `Query OK`. Как только будут введены все строки, наберите следующую команду, которая отобразит содержимое таблицы:

```
SELECT * FROM classics;
```



Теперь вернемся назад и посмотрим, как используется команда INSERT. Ее первая часть, INSERT INTO classics, сообщает MySQL, куда нужно вставлять следующие данные в таблицу classics. Затем в круглых скобках перечисляются четыре имени столбцов: author, title, type и year, которые отделяются друг от друга запятыми. Таким образом MySQL сообщается, что именно в эти четыре поля будут вставляться данные.

В каждой команде INSERT содержится ключевое слово VALUES, за которым следуют четыре строковых значения, взятых в кавычки и отделенных друг от друга запятыми. Они обеспечивают MySQL теми четырьмя значениями, которые будут вставлены в четыре ранее указанных столбца. (Как и во всех остальных примерах, разбиение команды на строки было моим собственным решением, придерживаться которого не обязательно.)

Каждый элемент данных будет вставлен по порядку в соответствующие столбцы. Если порядок перечисления столбцов и данных будет случайно перепутан, данные попадут не в те столбцы. А количество указанных столбцов должно соответствовать количеству элементов данных.

Переименование таблиц

Переименование таблиц, как и любые другие изменения ее структуры или метаданных, осуществляются посредством команды ALTER. Поэтому, чтобы, к примеру, изменить имя таблицы classics на pre1900, воспользуйтесь следующей командой:

```
ALTER TABLE classics RENAME pre1900;
```

Изменение типа данных столбца

Для изменения типа данных столбца также используется команда ALTER, но в этом случае вместе с ней применяется ключевое слово MODIFY. Поэтому для изменения типа данных столбца year с CHAR(4) на SMALLINT (для которого потребуется только 2 байта памяти, что способствует экономии дискового пространства) нужно ввести следующую команду:

```
ALTER TABLE classics MODIFY year SMALLINT;
```

После этого, если для MySQL есть смысл конвертировать тип данных, система автоматически изменит данные, сохраняя их значение.

Добавление нового столбца

Предположим, что таблица создана и заполнена большим объемом данных и тут выяснилось, что нужен еще один столбец. Не стоит расстраиваться. Посмотрите, как можно добавить к таблице новый столбец pages, который будет использоваться для хранения количества страниц, имеющихся в книге:

```
ALTER TABLE classics ADD pages SMALLINT UNSIGNED;
```

Эта команда добавляет новый столбец по имени pages, в котором используется тип данных UNSIGNED SMALLINT, подходящий для хранения значений вплоть до 65 535. Этого наверняка более чем достаточно для любой когда-либо изданной книги!

Переименование столбца

Посмотрев еще раз, можно заметить, что наличие в таблице столбца type может привести к путанице, поскольку такое же имя используется MySQL для идентификации типа данных. Но это не проблема — изменим имя этого столбца на category:

```
ALTER TABLE classics CHANGE type category VARCHAR(16);
```

Удаление столбца

Поразмыслив, можно прийти к выводу, что столбец pages, в котором хранится количество страниц, не представляет для этой базы данных особой ценности, поэтому его можно удалить, используя ключевое слово DROP:

```
ALTER TABLE classics DROP pages;
```

Функции MySQL

Стремление применять функции MySQL при таком обилии достаточно мощных функций PHP может вызвать недоумение. Ответ предельно прост: функции MySQL работают с данными непосредственно в самой базе. А при использовании PHP приходится сначала извлекать строку данных из MySQL, выполнять обработку, а затем выдавать первоначально задуманный запрос к базе данных. Применение встроенных функций MySQL не только существенно сокращает время обработки сложных запросов, но и упрощает сами запросы. При желании подробные сведения обо всех доступных строковых функциях и функциях даты и времени можно найти по следующим адресам:

- http://tinyurl.com/mysqlstrfuncs
- http://tinyurl.com/mysqldatefuncs

Проектирование базы данных

Перед тем как создавать базу данных, очень важно ее удачно спроектировать, в противном случае, скорее всего, придется возвращаться назад и изменять ее структуру, разбивая одни и объединяя другие таблицы и перемещая различные графы из таблицы в таблицу с целью достижения рациональных связей, которыми MySQL будет легче воспользоваться.
Для начала было бы неплохо сесть за стол с листом бумаги и карандашом и набросать подборку тех запросов, которые, на ваш взгляд, чаще всего будут нужны вам и вашим пользователям. Для базы данных книжного интернет-магазина могут быть записаны следующие вопросы.

- Сколько авторов, книг и покупателей имеется в базе данных?
- Каким автором написана та или иная книга?
- Какие книги написаны тем или иным автором?
- Какая книга продается по самой высокой цене?
- Какая книга является лидером продаж?
- Какие книги не покупались в этом году?
- Какие книги приобретены тем или иным покупателем?
- Какие книги были приобретены вместе с какими-нибудь другими книгами?

Разумеется, к такой базе данных может быть сделано и множество других запросов, но даже эта подборка даст вам представление о том, как следует спланировать структуру таблиц. Например, книги и номера ISBN должны быть, наверное, скомбинированы в одной таблице, поскольку они тесно взаимосвязаны (некоторые тонкости этого вопроса будут исследованы чуть позже). В отличие от этого книги и покупатели должны находиться в разных таблицах, поскольку они слабо взаимосвязаны. Покупатель может купить любую книгу и даже несколько экземпляров одной и той же книги, а книга может быть приобретена многими покупателями, и может не привлечь внимания еще большего количества потенциальных покупателей.

Когда планируется множество поисковых операций по каким-нибудь столбцам, зачастую их лучше всего поместить в общую таблицу. А когда какие-то элементы слабо связаны друг с другом, их лучше

поместить в отдельные таблицы. Если принять во внимание эти элементарные правила, то можно предположить, что для удовлетворения всех этих запросов нам понадобятся как минимум три таблицы.

- *authors (авторы).* Предполагается большое количество поисков по авторам, многие из которых сотрудничали при написании книг, а значит, будут показаны вместе. Оптимальных результатов поиска можно добиться, если о каждом авторе будет дана вся относящаяся к нему информация, следовательно, нам нужна таблица авторов — authors.
- *books (книги)*. Многие книги появляются в различных изданиях. Иногда у них разные издатели, а иногда разные книги имеют одно и то же название. Связи между книгами и авторами настолько сложны, что для книг нужна отдельная таблица.
- *customers (покупатели).* Причина, по которой покупатели должны находиться в собственной таблице, еще более прозрачна — покупатели могут приобрести любую книгу любого автора.

Первичные ключи

Используя возможности реляционных баз данных, мы можем задавать всю информацию для каждых автора, книги и покупателя в одном и том же месте. Очевидно, что нас интересуют связи между ними, например, кто написал каждую книгу и кто ее приобрел, и мы можем сохранить эту информацию лишь за счет создания связей между тремя таблицами. Я покажу вам основные принципы, которые нетрудно будет усвоить на практике.

Секрет заключается в присвоении каждому автору уникального идентификатора. То же самое делается для каждой книги и каждого покупателя. Смысл всего этого был объяснен в предыдущей главе: нам нужен первичный ключ. Для книги имеет смысл использовать в этом качестве номер ISBN, хотя вам, может быть, придется столкнуться с несколькими одинаковыми книгами, имеющими разные номера ISBN. Авторам и покупателям можно просто назначить произвольные ключи, имеющие свойство автоприращения — AUTO_INCREMENT, что, судя по предыдущей главе, делается весьма просто.

Проще говоря, каждая таблица будет спроектирована вокруг какого-нибудь объекта, в котором, скорее всего, будет вестись интенсивный поиск, — в данном случае вокруг автора, книги или покупателя, и этот объект должен иметь первичный ключ. В качестве ключа не следует выбирать ничего, что могло бы иметь одинаковое значение для разных объектов. Ситуация с номером ISBN является тем самым редким случаем, когда сама издательская индустрия предоставила нам первичный ключ, который можно считать уникальным для каждого продукта. В большинстве случаев для этих целей следует создавать произвольный ключ, использующий свойство AUTO_INCREMENT.

Нормализация

Процесс распределения данных по таблицам и создания первичных ключей называется нормализацией. Основная цель нормализации — обеспечить, чтобы каждая порция информации появлялась в базе данных только один раз. Дублирование данных приводит к крайне неэффективной работе, поскольку неоправданно увеличивает объем базы данных и замедляет тем самым доступ к информации. Еще важнее то, что дубликаты создают большой риск обновления только одной строки

продублированных данных, приводят к несогласованности в базе данных, являющейся потенциальным источником серьезных ошибок.

Если, к примеру, названия книг перечисляются и в таблице авторов, и в таблице книг и возникает необходимость исправить опечатку в названии, нужно будет вести поиск в обеих таблицах и вносить одинаковые изменения везде, где встречаются названия книг. Лучше хранить названия в одном месте, а в других местах использовать номер ISBN.

В процессе разбиения базы данных на несколько таблиц важно не зайти слишком далеко и не создать больше таблиц, чем требуется, что может также привести к неэффективности конструкции и замедлению доступа к данным. К счастью, изобретатель реляционной модели Эдгар Кодд проанализировал понятие нормализации и разбил его на три отдельные схемы, названные первой, второй и третьей нормальными формами.

Чтобы понять, как выполняется нормализация, начнем с весьма несуразной базы данных, представленной в таблице ниже, в которой имеется одна таблица, содержащая все сведения об авторах, книгах и вымышленных покупателях. Ее можно рассматривать в качестве первой попытки создания таблицы, отслеживающей, кто из покупателей какие книги заказал. Неэффективность такой конструкции не вызывает сомнений, поскольку данные повсеместно дублируются.

Автор 1	Автор 2	Название	ISBN	Цена	Имя покупателя	Адрес покупателя	Дата покупки
David Skla	Adam Trachten-berg	PHP Cookbook	0596101015	44,99	Emma Brown	1565 Rainbow Road, Los Angeles, CA 90014	Март 03 2009
Danny Goodman		Dynamic HTML	0596527403	59,99	**Darren Ryder**	**4758 Emily Drive, Richmond, VA 23219**	**Декабрь19 2008**
Hugh E. Williams	David Lane	PHP and MySQL	0596005436	44,95	Earl B. Thurston	862 Gregory Lane, Frankfort, KY 40601	Июнь 22 2009
David Sklar	Adam Trachten-berg	PHP Cookbook	0596101015	44,99	**Darren Ryder**	**4758 Emily Drive, Richmond, VA 23219**	**Декабрь 19 2008**

Rasmus Lerdorf	Kevin Tatroe & Peter MacIntyre	Program-ming PHP	0596006815	39,99	David Miller	3647 Cedar Lane, Waltham, MA 02154	Январь 16 2009

Первая нормальная форма

Чтобы база данных соответствовала первой нормальной форме, она должна выполнять три требования.

- В ней не должно быть повторяющихся столбцов, содержащих одни и те же типы данных.
- Все графы должны содержать только одно значение.
- Для уникальной идентификации каждой строки должен быть первичный ключ.

Рассматривая по порядку эти требования, вы заметите, что в столбцы Author 1 и Author 2 заложены повторяющиеся типы данных. Итак, у нас уже появилась та самая графа, которую следует поместить в отдельную таблицу, поскольку повторяющаяся графа Author противоречит правилу 1.

Второе несоответствие связано с тем, что для последней книги, Programming PHP, указаны три автора. Я считаю, что использование одного и того же столбца Author 2 для имен двух авторов — Kevin Tatroe и Peter MacIntyre — нарушает правило 2. Это еще одна причина перемещения всех сведений об авторах в отдельную таблицу.

А вот правило 3 здесь соблюдается, потому что первичный ключ в столбце ISBN уже создан.

Новая таблица Авторы, приведенная в таблице.

ISBN	**Author (Автор)**
0596101015	David Skla
0596101015	Adam Trachtenberg
0596527403	Danny Goodman
0596005436	Hugh E Williams
0596005436	David Lane
0596006815	Rasmus Lerdorf
0596006815	Kevin Tatroe
0596006815	Peter MacIntyre

Как было отмечено ранее, ISBN будет служить в качестве первичного ключа для таблицы книг — Books, когда дело дойдет до ее создания. При практической разработке для таблицы Authors также нужно создать первичный ключ, обеспечивающий его уникальную идентификацию.

Поэтому для таблицы Authors графа ISBN является простой графой, для которой в целях ускорения поиска может быть, наверное, создан ключ, но этот ключ будет уже не первичным. Фактически в этой таблице он и не может быть первичным, поскольку не обладает уникальностью: один и тот же номер ISBN появляется по нескольку раз в тех случаях, когда над одной книгой работали несколько авторов. Поскольку мы будем использовать такой ключ для связи авторов с книгами в другой таблице, эта графа называется внешним ключом.

Вторая нормальная форма

Первая нормальная форма позволяет разобраться с продублированными данными (или избыточностью) в нескольких столбцах. Вторая нормальная форма имеет отношение только к решению проблемы избыточности в нескольких строках.

Чтобы привести базу данных ко второй нормальной форме, ваши таблицы должны уже иметь первую нормальную форму. Как только это будет сделано, для определения столбцов, данные в которых повторяются в разных местах, и последующего их перемещения в собственные таблицы применяется вторая нормальная форма.

Давайте еще раз посмотрим на таблице. Видите, Darren Ryder приобрел две книги, и поэтому его данные продублированы. Это говорит о том, что графы, имеющие отношение к покупателю (имя и адрес), следует переместить в их собственные таблицы. В таблице ниже показан результат удаления двух столбцов, касающихся покупателя.

Таблица книг

ISBN	Название	Цена
0596101015	PHP Cookbook	44,99
0596527403	Dynamic HTML	59,99
0596005436	PHP and MySQL	44,95
0596101015	PHP Cookbook	44,99

Таким образом, в таблице остались только графы номера ISBN, названия и цены для четырех уникальных книг, поэтому теперь это эффективная в использовании и независимая таблица, удовлетворяющая требованиям как первой, так и второй нормальной формы. Попутно мы справились с сокращением информации до уровня тех данных, которые имеют непосредственное отношение к книгам с определенными названиями. Эта таблица может также включать год издания, количество страниц, количество переизданий и т. д., поскольку все эти данные имеют тесную связь друг с другом. Единственное правило гласит: сюда нельзя помещать графы, которые могут содержать несколько

значений для одной книги, поскольку тогда нам придется указывать одну и ту же книгу в нескольких строках, нарушая таким образом правила второй нормальной формы.

К примеру, к нарушениям на этом этапе нормализации может привести восстановление столбца авторов. Но изучая извлеченные графы, относящиеся к покупателям, которые теперь показаны в таблице ниже, можно заметить, что эта таблица все же требует дополнительной нормализации, поскольку сведения о покупателе Darren Ryder по-прежнему продублированы. Следует также признать, что правило 2 первой нормальной формы (все графы должны содержать только одно значение) здесь не соблюдается, поскольку адресные данные нужно разбить на отдельные графы для адреса, города, штата и почтового индекса.

ISBN	Имя покупателя	Адрес покупателя	Дата покупки
0596101015	Emma Brown	1565 Rainbow Road, Los Angeles, CA 90014	Март 03 2009
0596527403	Darren Ryder	4758 Emily Drive, Richmond, VA 23219	Декабрь19 2008
0596005436	Earl B. Thurston	862 Gregory Lane, Frankfort, KY 40601	Июнь 22 2009
0596101015	Darren Ryder	4758 Emily Drive, Richmond, VA 23219	Декабрь 19 2008
0596101015	David Miller	3647 Cedar Lane, Waltham, MA 02154	Январь 16 2009

Нужно продолжить разбиение этой таблицы, чтобы обеспечить однократный ввод каждого из сведений, касающихся покупателя. Поскольку ISBN не относится к таким сведениям и не может использоваться в качестве первичного ключа для идентификации покупателей (или авторов), должен быть создан новый ключ.

В таблице ниже показан результат нормализации таблицы покупателей в соответствии с правилами первой и второй нормальных форм. Теперь у каждого покупателя есть уникальный номер покупателя, который применяется в качестве первичного ключа и который, скорее всего, был создан с использованием свойства автоприращения — AUTO_INCREMENT. Все составляющие адресов были также распределены по разным столбцам, для того чтобы упростить их поиск и обновление.

Таблица покупателей

Номер покупателя	Имя	Адрес	Город	Штат	Почтовый индекс
1	Emma Brown	1565 Rainbow Road	Los Angeles	CA	90014

2	Darren Ryder	4758 Emily Drive	Richmond	VA	23219
3	Earl B. Thurston	862 Gregory Lane	Frankfort	KY	40601
4	David Miller	3647 Cedar Lane	Waltham	MA	02154

В то же время для нормализации таблицы с покупками необходимо было удалить информацию о покупках, поскольку в противном случае в ней встречались бы одни и те же сведения о покупателе для каждой купленной им книги. Вместо этого данные о покупках теперь помещены в новую таблицу **Покупки**.

Таблица Покупки

Номер покупателя	ISBN	Дата покупки
1	0596101015	Март 03 2009
2	0596527403	Декабрь19 2008
3	0596005436	Июнь 22 2009
4	0596101015	Декабрь 19 2008

Здесь в качестве ключа, связывающего вместе таблицы Покупатели и Покупки, опять используется графа Номер Покупателя. Поскольку здесь повторно появляется графа ISBN, эта таблица может быть связана также либо с таблицей Авторов, либо с таблицей Названий.

Третья нормальная форма

После приведения в соответствие первой и второй нормальным формам база данных приобрела подходящий вид, и в дальнейшем вам, возможно, уже не придется что-либо в ней изменять. Но если применить к базе данных более суровые требования, то можно довести ее до соответствия правилам третьей нормальной формы, которые требуют, чтобы данные, не имеющие непосредственной зависимости от первичного ключа, но имеющие зависимость от другого значения в таблице, были также перемещены в отдельные таблицы в соответствии с тем, к чему они имеют отношение.

Например, касательно таблицы покупателей, можно утверждать, что ключи штат, город и почтовый индекс не имеют прямого отношения к каждому покупателю, поскольку эти же составляющие будут присутствовать в адресах многих других людей. Но они напрямую связаны друг с другом тем, что улица в адресе относится к городу, а город относится к штату. Поэтому, чтобы соблюсти правила третьей нормальной формы для таблицы покупателей.

Таблица покупателей

Номер покупателя	Имя	Адрес	Почтовый индекс
1	Emma Brown	1565 Rainbow Road	90014
2	Darren Ryder	4758 Emily Drive	23219
3	Earl B. Thurston	862 Gregory Lane	40601
4	David Miller	3647 Cedar Lane	02154

Таблица Почтовых индексов

Почтовый индекс	Идентификатор города
90014	4341
23219	1234
40601	1231
02154	6543

Таблица городов

Идентификатор города	Название	Идентификатор штата
4341	Los Angeles	5
1234	Richmond	46
1231	Frankfort	11
6543	Waltham	21

Таблица штатов

Идентификатор штата	Название	Аббревиатура
5	California	CA
46	Virginia	VA
11	Kenthucky	KY
21	Massachussetts	MA

Доступ к MySQL с помощью PHP

Смысл использования PHP в качестве интерфейса к MySQL заключается в форматировании результатов SQL-запросов и придании им внешнего вида, предназначенного для вывода на веб-страницу. Обладая возможностью входа в установленную систему MySQL с помощью своего имени пользователя и пароля, вы можете сделать то же самое и из PHP. Но вместо использования командной строки MySQL для ввода команд и просмотра выходной информации нужно будет создать строки запроса, а затем передать их MySQL. Ответ MySQL поступит в виде структуры данных, которую PHP сможет распознать, а не в виде того отформатированного экранного вывода, который вы наблюдали в процессе работы с командной строкой. Затем с помощью команд PHP можно будет извлекать данные и приводить их к формату веб-страницы.

Процесс использования базы данных с помощью PHP заключается в следующем.

1. Подключение к серверу баз данных.
2. Выбор базы данных, которая будет использоваться.
3. Создание строки запроса.
4. Выполнение запроса.
5. Извлечение результатов и вывод их на веб-страницу.
6. Повторение шагов с 3-го по 5-й до тех пор, пока не будут извлечены все необходимые данные.
7. Отключение от базы.

Запросы к базе данных MySQL с помощью mysqli

Для разбора интерфейса mysqli, встроенного в PHP, давайте выполним небольшой пример.

1. Создайте файл login.php, который будет хранить в себе доступы к нашей БД, и добавьте в него следующий код

```
<?php
    // хост БД
    $db_hostname = 'localhost';
    // имя базы данных
    $db_database = 'shop';
    // имя пользователя
    $db_username = 'root';
    // пароль
    $db_password = '';
```

(Дамп БД приложен к данной методичке)

2. После сохранения файла login.php для обращения к базе данных используется инструкция require_once, а подключение к серверу осуществляется в порядке, показанном в примере:

```
<?php
  require_once 'login.php';
  $connection = new mysqli($db_hostname, $db_username, $db_password, $db_
database);
  if ($connection->connect_error) die($connection->connect_error);
```

В данном примере путем вызова метода mysqli со всеми значениями, извлеченными из файла login.php, создается новый объект по имени $connection. Обратите внимание на проверку возникновения ошибок, достигаемую за счет ссылки на свойство $connection->connect_error. Если оно имеет значение TRUE, вызывается функция die и выводятся подробности, объясняющие характер ошибки. Свойство connect_error объекта $connection содержит строку, детализирующую ошибку подключения. Объект $connection будет использован для обращения к базе данных MySQL в следующих примерах.

3. Для отправки запроса к серверу БД необходимо вызвать метод query у объекта $connection, который мы создали на предыдущем шаге:

```
$query = "SELECT * FROM catalogs";
$result = $connection->query($query);

if(!$result) die($connection->error);
```

Переменной $query присвоено значение, соответствующее выполняемому запросу, значение передается методу объекта $connection, который возвращает результат, помещаемый в объект $result. Все, что нужно, было сделано с применением $connection и перешло в $result, чтобы можно было воспользоваться тем, что вернулось из подключения. При ошибке значением $result окажется FALSE; в противном случае это будет объект, к которому можно будет обратиться. Свойство error объекта $connection содержит строку с подробными сведениями об ошибке.

4. Возвращенным $result объектом можно воспользоваться для поэлементного извлечения требуемых данных, применяя для этого метод объекта fetch_assoc.

```
for($i = 0, $length = $result->num_rows; $i < $length; $i++){
        // получаем $i-й элемент результирующего набора
        $result->data_seek($i);
        // получаем данные строки в виде ассоциативного массива
   $row = $result->fetch_assoc();
   echo $row['name'].'<br />';
}
// закрываем соединение с БД
$result->close();
```

Полный пример работы выглядит вот так:

```
<?php
```

```
require_once 'login.php';
$connection = new mysqli($db_hostname, $db_username, $db_password, $db_database);
  if ($connection->connect_error) die($connection->connect_error);


$query = "SELECT * FROM catalogs";
$result = $connection->query($query);

if(!$result) die($connection->error);

for($i = 0, $length = $result->num_rows; $i < $length; $i++){
   $result->data_seek($i);
   $row = $result->fetch_assoc();
   echo $row['name'].'<br />';
}

$result->close();
```

Если необходимо возвращать результат в виде объекта, то вместо вызова метода fetch_assoc() нужно вызвать метод fetch_object().

Простой SELECT-запрос с экранированием данных

Ни в коем случае нельзя подставлять в запросы данные от пользователя в чистом виде, поскольку можно получить SQL-injection. Для экранирования данных используется функция real_escape_string. Запрос может выглядеть следующим образом:

```
////////////////// начало скрипта
$connection = new mysqli($db_hostname, $db_username, $db_password, $db_database);
if ($connection->connect_error) die($connection->connect_error);

$query = "SELECT * FROM catalogs WHERE name
='".$connection->real_escape_string('Процессоры')."'";
$result = $connection->query($query);

//////////////// продолжение скрипта
```

Подготовленные запросы

В чем удобство подготовленных SELECT-запросов — на базу данных можно возложить ответственность за экранирование данных. Механизм приблизительно следующий:

```
$query = "SELECT * FROM catalogs WHERE name = ?";
// создать объект класса mysqli_stmt, который представляет подготовленное выражение
$stmt = $connection->stmt_init();
$name = 'Процессоры';

$stmt->prepare($query);

if ($stmt->error) {
    throw new Exception('Ошибка в запросе');
}

// Передаем параметры в запрос
$stmt->bind_param('s', $name);
// Выполняем запрос
$stmt->execute();

if ($stmt->error) {
    throw new Exception('Ошибка в запросе');
}

$result = $stmt->get_result();

for($i = 0, $length = $result->num_rows; $i < $length; $i++){
    $result->data_seek($i);
    $row = $result->fetch_assoc();
    echo $row['name'].'<br />';
}
```

Если подготовка запроса проходит успешно (нет ошибок в SQL-запросе) — тогда с помощью метода bind_param подвязываются переменные. Первый параметр содержит строку с типами переменных, последующие параметры — сами переменные, которые нужно "подставить" в запрос.

В нашем примере использовалась одна переменная, тип переменной представлен буквой i, что означает целое число. Всего можно подвязать 4 типа переменных:

i — целое число.
d — дробное число.
s — строковые данные.
b — двоичный объект (blob).

Подготовленный SELECT-запрос с произвольным набором переменных для экранирования

Рано или поздно возникает ситуация, когда количество данных для экранирования может быть динамическим, то есть изменяется под влиянием каких-то условий пользовательского интерфейса.

Пример такого SELECT-запроса:

```
<?php
// в переменную $bindParamTypes типа string собираем типы подставляемых переменных
$bindParamTypes = '';

// в массив $bindParamVariables собираем значения переменных для подстановки
$bindParamVariables = array();

// в массив $whereArray собираем условия для команды WHERE в SQL-запросе
$whereArray = array();

// описываем параметр № 1
$bindParamTypes .= 'i';
$bindParamVariables[] = 1;
$whereArray[] = 'id=?';

// описываем параметр № 2
$bindParamTypes .= 's';
$bindParamVariables[] = '%some_title%';
$whereArray[] = 'title LIKE ?';

// формируем SQL запрос
$selectQuery = 'SELECT * FROM table'

if (count($whereArray) > 0) {
    // если условия заданы — добавляем команду WHERE в SQL-запрос
    $selectQuery .= ' WHERE ' . implode(' AND ', $whereArray);
}

if (count($whereArray) == 0) {
     // если условия не заданы — выполняется неподготовленный SQL-запрос, результат записывается в
$resultQuery
    $resultQuery = $ourMysqli->query($selectQuery);
    if ($resultQuery === false) {
        throw new \Exception('Ошибка в SQL-запросе!');
    }
} else {
    // если условия заданы — выполняем подготовленый SQL-запрос.

    // создаем объект класса mysqli_stmt
```

```
    $mysqliStmt = $ourMysqli->stmt_init();

    // подготавливаем SQL-запрос
    $mysqliStmt->prepare($selectQuery);
    if ($mysqliStmt->errno) {
        throw new \Exception('Ошибка в SQL-запросе!');
    }

    // вызываем функцию bind_param
    $mysqliStmtParameters = array_merge(array($bindParamTypes), $bindParamVariables);
    call_user_func_array(
        array(
            $mysqliStmt,
            'bind_param'
        ),
        refArrayValues($mysqliStmtParameters)
    );

    $mysqliStmt->execute();
    if ($mysqliStmt->errno) {
        throw new \Exception('Ошибка в SQL-запросе!');
    }

    // получаем результат (работает при включенном mysqlnd драйвере)
    $resultQuery = $mysqliStmt->get_result();

    // закрываем подготовленный запрос
    $mysqliStmt->close();
}

// обработка результатов выборки
// вспомогательная функция, которая создает ссылки на переменные внутри массива
function refArrayValues($arr){
    if (strnatcmp(phpversion(),'5.3') >= 0)
    {
        $refs = array();
        foreach($arr as $key => $value)
            $refs[$key] = &$arr[$key];
        return $refs;
    }
    return $arr;
}

?>
```

Выполнение INSERT, UPDATE и DELETE запросов

Запросы на обновление данных (INSERT, UPDATE, REPLACE, DELETE) выполняются аналогично запросам SELECT. Например:

Обычный запрос:

```
<?php

$insertQuery = '
INSERT INTO table
SET
field1="' . $ourMysqli->real_escape_string($data1) . '",
field2="' . $ourMysqli->real_escape_string($data2) . '"';
$resultQuery = $ourMysqli->query($insertQuery);
if ($resultQuery === false) {
   throw new \Exception('Ошибка в SQL-запросе!');
}
?>
```

Подготовленный запрос:

```
<?php
$insertQuery = <<< SQL_QUERY
INSERT INTO table
SET
        field1=?,
        field2=?,
        field3=?,
        field4=?
SQL_QUERY;

$mysqliStmt = $ourMysqli->stmt_init();
$mysqliStmt->prepare($insertQuery);
if ($mysqliStmt->errno) {
   throw new \Exception('Ошибка в SQL-запросе!');
}
$mysqliStmt->bind_param('issi', $field1Int, $field2Str, $field3Str, $field4Int);
$mysqliStmt->execute();
if ($mysqliStmt->errno) {
   throw new \Exception('Ошибка в SQL-запросе!');
}
$insertId = $mysqliStmt->insert_id;
$mysqliStmt->close();
?>
```

Множественные запросы

MySQL поддерживает наличие нескольких SQL предложений в тексте одного запроса. Пересылка на сервер нескольких выражений в одном запроса уменьшает количество клиент-серверных взаимодействий, но требует специальной обработки.

Множественные запросы, или мультизапросы, должны запускаться функцией mysqli_multi_query(). Отдельные SQL предложения в мультизапросе отделяются точкой с запятой. После выполнения мультизапроса все результирующие наборы, которые он вернул, необходимо извлечь.

MySQL сервер поддерживает наличие в одном мультизапросе подзапросов, как возвращающих результирующий набор, так и не возвращающих.

Пример:

```
<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Не удалось подключиться к MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
}

if (!$mysqli->query("DROP TABLE IF EXISTS test") || !$mysqli->query("CREATE TABLE test(id INT)")) {
    echo "Не удалось создать таблицу: (" . $mysqli->errno . ") " . $mysqli->error;
}

$sql = "SELECT COUNT(*) AS _num FROM test; ";
$sql.= "INSERT INTO test(id) VALUES (1); ";
$sql.= "SELECT COUNT(*) AS _num FROM test; ";

if (!$mysqli->multi_query($sql)) {
    echo "Не удалось выполнить мультизапрос: (" . $mysqli->errno . ") " . $mysqli->error;
}

do {
    if ($res = $mysqli->store_result()) {
        var_dump($res->fetch_all(MYSQLI_ASSOC));
        $res->free();
    }
} while ($mysqli->more_results() && $mysqli->next_result());
```

Результат работы:

```
<?php
array(1) {
  [0]=>
```

```
  array(1) {
    ["_num"]=>
    string(1) "0"
  }
}
array(1) {
  [0]=>
  array(1) {
    ["_num"]=>
    string(1) "1"
  }
}
```

Работа с БД при помощи PDO

PHP Data Objects (PDO) — расширение для PHP, предоставляющее разработчику универсальный интерфейс для доступа к различным базам данных.

В чем преимущество PDO?

Этот вопрос можно раскрыть тремя пунктами.

1. Универсальный интерфейс для работы с различными базами данных. Разработчик может легко перевести свое веб-приложение на другу СУБД, поменяв при этом всего пару строк кода;
2. Высокая скорость работы;
3. Подготовленные выражения.
4. Поддержка транзакций и т.д.

На данный момент расширение PDO может поддерживать СУБД для которой существует PDO-драйвер:

- PDO_CUBRID (CUBRID)
- PDO_DBLIB (FreeTDS / Microsoft SQL Server / Sybase)
- PDO_FIREBIRD (Firebird/Interbase 6)
- PDO_IBM (IBM DB2)
- PDO_INFORMIX (IBM Informix Dynamic Server)
- PDO_MYSQL (MySQL 3.x/4.x/5.x)
- PDO_OCI (Oracle Call Interface)
- PDO_ODBC (ODBC v3 (IBM DB2, unixODBC and win32 ODBC))
- PDO_PGSQL (PostgreSQL)
- PDO_SQLITE (SQLite 3 and SQLite 2)
- PDO_SQLSRV (Microsoft SQL Server)
- PDO_4D (4D)

Для проверки, подключен ли у вас на сервере тот или иной драйвер PDO? используйте следующую конструкцию:

```
print_r(PDO::getAvailableDrivers());
```

Подключение

Способы подключения к разным СУБД могут незначительно отличаться. Ниже приведены примеры подключения к наиболее популярным из них. Можно заметить, что первые три имеют идентичный синтаксис, в отличие от SQLite.

```
try {

  # MS SQL Server и Sybase через PDO_DBLIB
  $DBH = new PDO("mssql:host=$host;dbname=$dbname", $user, $pass);
  $DBH = new PDO("sybase:host=$host;dbname=$dbname", $user, $pass);

  # MySQL через PDO_MYSQL
  $DBH = new PDO("mysql:host=$host;dbname=$dbname", $user, $pass);

  # SQLite
  $DBH = new PDO("sqlite:my/database/path/database.db");

}
catch(PDOException $e) {

    echo $e->getMessage();

}
```

Формат подключения к БД следующий:
тип_базы_данных:host=**имя_хоста**;db=**name**

Обратите внимание на блок try/catch – всегда стоит оборачивать в него все свои PDO-операции и использовать механизм исключений.

Закрыть любое подключение можно путем переопределения его переменной в null.

Исключения и PDO

PDO умеет выбрасывать исключения при ошибках, поэтому все должно находиться в блоке try/catch. Сразу после создания подключения, PDO можно перевести в любой из трех режимов ошибок:

- $DBH->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_SILENT);
- $DBH->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING);
- $DBH->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

Но стоит заметить, что ошибка при попытке соединения будет всегда вызывать исключение.

PDO::ERRMODE_SILENT

Это режим по умолчанию. Примерно то же самое вы, скорее всего, используете для отлавливания ошибок в расширениях mysql и mysqli. Следующие два режима больше подходят для DRY программирования.

PDO::ERRMODE_WARNING

Этот режим вызовет стандартный Warning и позволит скрипту продолжить выполнение. Удобен при отладке.

PDO::ERRMODE_EXCEPTION

В большинстве ситуаций этот тип контроля выполнения скрипта предпочтителен. Он выбрасывает исключение, что позволяет вам ловко обрабатывать ошибки и скрывать щепетильную информацию. Как, например, тут:

```
# подключаемся к базе данных
try {
  $DBH = new PDO("mysql:host=$host;dbname=$dbname", $user, $pass);
  $DBH->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION );

  # Набрал DELECT вместо SELECT!
  $DBH->prepare('DELECT name FROM people')->execute();
}
catch(PDOException $e) {
    echo "Ошибка SQL синтаксиса";
    file_put_contents('PDOErrors.txt', $e->getMessage(), FILE_APPEND);
}
```

В SQL-выражении есть синтаксическая ошибка, которая вызовет исключение. Мы можем записать детали ошибки в лог-файл и человеческим языком намекнуть пользователю, что что-то случилось.

Insert и Update

Вставка новых и обновление существующих данных являются одними из наиболее частых операций с БД. В случае с PDO этот процесс обычно состоит из двух шагов. (В следующей секции все относится как к UPDATE, так и INSERT)

Тривиальный пример вставки новых данных:

```
# STH означает "Statement Handle"
$STH = $DBH->prepare("INSERT INTO folks ( first_name ) values ( 'Cathy' )");
$STH->execute();
```

Вообще-то можно сделать то же самое одним методом exec(), но двухшаговый способ дает все преимущества prepared statements. Они помогают в защите от SQL-инъекций, поэтому имеет смысл их использовать даже при однократном запросе.

Prepared Statements

Использование prepared statements укрепляет защиту от SQL-инъекций.

Prepared statement — это заранее скомпилированное SQL-выражение, которое может быть многократно выполнено путем отправки серверу лишь различных наборов данных. Дополнительным преимуществом является невозможность провести SQL-инъекцию через данные, используемые в placeholder'ах.

Ниже находятся три примера prepared statements.

```
# без placeholders - дверь SQL-инъекциям открыта!
$STH = $DBH->prepare("INSERT INTO folks (name, addr, city) values ($name, $addr, $city)");

# безымянные placeholders
$STH = $DBH->prepare("INSERT INTO folks (name, addr, city) values (?, ?, ?)");

# именные placeholders
$STH = $DBH->prepare("INSERT INTO folks (name, addr, city) values (:name, :addr, :city)");
```

Первый пример здесь лишь для сравнения, его стоит избегать. Разница между безымянными и именными placeholder'ами в том, как вы будете передавать данные в prepared statements.

Безымянные placeholder'ы

```
# назначаем переменные каждому placeholder, с индексами от 1 до 3
$STH->bindParam(1, $name);
$STH->bindParam(2, $addr);
$STH->bindParam(3, $city);

# вставляем одну строку
$name = "Daniel"
$addr = "1 Wicked Way";
$city = "Arlington Heights";
$STH->execute();

# вставляем еще одну строку, уже с другими данными
$name = "Steve"
$addr = "5 Circle Drive";
$city = "Schaumburg";
$STH->execute();
```

Здесь два шага. На первом мы назначаем всем placeholder'ам переменные (строки 2-4). Затем назначаем этим переменным значения и выполняем запрос. Чтобы послать новый набор данных, просто измените значения переменных и выполните запрос еще раз.

Если в вашем SQL-выражении много параметров, то назначать каждому по переменной весьма неудобно. В таких случаях можно хранить данные в массиве и передавать его:

```
# набор данных, которые мы будем вставлять
$data = array('Cathy', '9 Dark and Twisty Road', 'Cardiff');

$STH = $DBH->prepare("INSERT INTO folks (name, addr, city) values (?, ?, ?)");
$STH->execute($data);
```

$data[0] вставится на место первого placeholder'a, $data[1] — на место второго, и т.д. Но будьте внимательны: если ваши индексы сбиты, это работать не будет.

Именные placeholder'ы

```
# первым аргументом является имя placeholder'a
# его принято начинать с двоеточия
# хотя работает и без них
$STH->bindParam(':name', $name);
```

Здесь тоже можно передавать массив, но он должен быть ассоциативным. В роли ключей должны выступать имена placeholder'ов.

```
# данные, которые мы вставляем
$data = array( 'name' => 'Cathy', 'addr' => '9 Dark and Twisty', 'city' => 'Cardiff' );
$STH = $DBH->prepare("INSERT INTO folks (name, addr, city) values (:name, :addr, :city)");
$STH->execute($data);
```

Одним из удобств использования именных placeholder'ов является возможность вставки объектов напрямую в базу данных, если названия свойств совпадают с именами параметров. Вставку данных, к примеру, вы можете выполнить так:

```
# класс для простенького объекта
class person {
    public $name;
    public $addr;
    public $city;

    function __construct($n,$a,$c) {
        $this->name = $n;
        $this->addr = $a;
        $this->city = $c;
    }
    # так далее...
}

$cathy = new person('Cathy','9 Dark and Twisty','Cardiff');
```

```
# а тут самое интересное
$STH = $DBH->prepare("INSERT INTO folks (name, addr, city) values (:name, :addr, :city)");
$STH->execute((array)$cathy);
```

Преобразование объекта в массив при execute() приводит к тому, что свойства считаются ключами массива.

Выборка данных

Данные можно получить с помощью метода ->fetch(). Перед его вызовом желательно явно указать, в каком виде они вам требуются. Есть несколько вариантов:
PDO::FETCH_ASSOC: возвращает массив с названиями столбцов в виде ключей
PDO::FETCH_BOTH (по умолчанию): возвращает массив с индексами как в виде названий стобцов, так и их порядковых номеров
PDO::FETCH_BOUND: присваивает значения столбцов соответствующим переменным, заданным с помощью метода ->bindColumn()
PDO::FETCH_CLASS: присваивает значения столбцов соответствующим свойствам указанного класса. Если для какого-то столбца свойства нет, оно будет создано
PDO::FETCH_INTO: обновляет существующий экземпляр указанного класса
PDO::FETCH_LAZY: объединяет в себе PDO::FETCH_BOTH и PDO::FETCH_OBJ
PDO::FETCH_NUM: возвращает массив с ключами в виде порядковых номеров столбцов
PDO::FETCH_OBJ: возвращает анонимный объект со свойствами, соответствующими именам столбцов

На практике вам обычно хватит трех: FETCH_ASSOC, FETCH_CLASS, и FETCH_OBJ. Чтобы задать формат данных, используется следующий синтаксис:
```
$STH->setFetchMode(PDO::FETCH_ASSOC);
```

Также можно задать его напрямую при вызове метода ->fetch().

FETCH_ASSOC

При этом формате создается ассоциативный массив с названиями столбцов в виде индексов. Он должен быть знаком тем, кто использует расширения mysql/mysqli.

```
# поскольку это обычный запрос без placeholder’ов,
# можно сразу использовать метод query()
$STH = $DBH->query('SELECT name, addr, city from folks');

# устанавливаем режим выборки
$STH->setFetchMode(PDO::FETCH_ASSOC);

while($row = $STH->fetch()) {
    echo $row['name'] . "\n";
    echo $row['addr'] . "\n";
    echo $row['city'] . "\n";
}
```

Цикл while() переберет весь результат запроса.

FETCH_OBJ

Данный тип получения данных создает экземпляр класса std для каждой строки.

```
# создаем запрос
$STH = $DBH->query('SELECT name, addr, city from folks');

# выбираем режим выборки
$STH->setFetchMode(PDO::FETCH_OBJ);

# выводим результат
while($row = $STH->fetch()) {
    echo $row->name . "\n";
    echo $row->addr . "\n";
    echo $row->city . "\n";
}
```

FETCH_CLASS

При использовании fetch_class данные заносятся в экземпляры указанного класса. При этом значения назначаются свойствам объекта ДО вызова конструктора. Если свойства с именами, соответствующими названиям столбцов, не существуют, они будут созданы автоматически (с областью видимости public).

Если ваши данные нуждаются в обязательной обработке сразу после их получения из базы данных, ее можно реализовать в конструкторе класса.

Для примера возьмем ситуацию, когда вам нужно скрыть часть адреса проживания человека.

```
class secret_person {
    public $name;
    public $addr;
    public $city;
    public $other_data;

    function __construct($other = '') {
        $this->addr = preg_replace('/[a-z]/', 'x', $this->addr);
        $this->other_data = $other;
    }
}
```

При создании объекта все латинские буквы в нижнем регистре должны замениться на x. Проверим:

```
$STH = $DBH->query('SELECT name, addr, city from folks');
$STH->setFetchMode(PDO::FETCH_CLASS, 'secret_person');

while($obj = $STH->fetch()) {
```

```
    echo $obj->addr;
}
```

Если в базе данных адрес выглядит как ’5 Rosebud’, то на выходе получится ’5 Rxxxxxx’.

Конечно, иногда будет требоваться, чтобы конструктор вызывался ПЕРЕД присваиванием значений. PDO такое тоже позволяет.

```
$STH->setFetchMode(PDO::FETCH_CLASS | PDO::FETCH_PROPS_LATE, 'secret_person');
```

Теперь, когда вы дополнили предыдущий пример дополнительной опцией (PDO::FETCH_PROPS_LATE), адрес видоизменяться не будет, так как после записи значений ничего не происходит.

Наконец, при необходимости можно передавать конструктору аргументы прямо при создании объекта:
```
$STH->setFetchMode(PDO::FETCH_CLASS, 'secret_person', array('stuff'));
```

Можно даже передавать разные аргументы каждому объекту:

```
$i = 0;
while($rowObj =  $STH->fetch(PDO::FETCH_CLASS, 'secret_person', array($i))) {
    // что-то делаем
    $i++;
}
```

Другие полезные методы

Метод ->lastInsertId() возвращает id последней вставленной записи. Стоит заметить, что он всегда вызывается у объекта базы данных (в статье он именуется $DBH), а не объекта с выражением ($STH).

Метод ->exec() используется для операций, которые не возвращают никаких данных, кроме количества затронутых ими записей.

Метод ->quote() ставит кавычки в строковых данных таким образом, что их становится безопасно использовать в запросах. Пригодится, если вы не используете prepared statements.
```
$rows_affected = $STH->rowCount();
```

Метод ->rowCount() возвращает количество записей, которые поучаствовали в операции.

Транзакции

Транзакция – это совокупность запросов базу данных, которые должны быть обязательно выполнены все. Если какой-либо запрос не выполнен или выполнен с ошибкой, то транзакция отменяется и изменений данных в базе не происходит.

Это нужно, чтобы гарантировать сохранение целостности данных при нескольких запросах. например при переводе денежных средств со счета на счет.
Чтобы выполнить транзакцию в PDO необходимо перейти в режим ручного подтверждения запросов.
Чтобы выключить режим автоподтверждения, выполняем команду:

```
$db->beginTransaction();
```

После этого выполняем столько запросов к базе данных сколько необходимо сделать в этой транзакции.

И только после того как все запросы будут выполнены, Вы можете подтвердить транзакцию функцией

```
$db->commit();
```

или отменить транзакцию

```
$db->rollback();
```

Вот небольшой пример транзакций:

```
try
{
	$connect_str = DB_DRIVER . ':host='. DB_HOST . ';dbname=' . DB_NAME;
	$db = new PDO($connect_str,DB_USER,DB_PASS);

	$rows = $db->exec("CREATE TABLE `testing`(
		id INT PRIMARY KEY AUTO_INCREMENT,
		fname VARCHAR(20) NOT NULL DEFAULT '',
		email VARCHAR(50) NOT NULL DEFAULT '',
		money INT NOT NULL DEFAULT 0) ENGINE=InnoDB;");

	$rows = $db->exec("INSERT INTO `testing` VALUES
		(null, 'Ivan', 'ivan@test.com', 15000),
		(null, 'Petr', 'petr@test.com', 411000),
		(null, 'Vasiliy', 'vasiliy@test.com', 1500000)
	");

	// Попробуем от Ивана перевести сумму 50000
	// Петру

	$summ = 50000;

	$transaction = true;

	$db->beginTransaction();

	$sth1 = $db->query("SELECT money FROM testing WHERE fname='Ivan'");
```

```
        $sth2 = $db->query("SELECT money FROM testing WHERE fname='Petr'");
        $row1 = $sth1->fetch();
        $row2 = $sth2->fetch();

        if(!$row1 || !$row2) $transaction = false;

        $total2 = $summ + $row2['money'];
        $total1 = $row1['money'] - $summ;

        if($total1 < 0 || $total2 < 0) $transaction = false;

        $num_rows1 = $db->exec("UPDATE `testing` SET money='" . $total1 . "' WHERE fname='Ivan'");
        $num_rows2 = $db->exec("UPDATE `testing` SET money='" . $total2 . "' WHERE fname='Petr'");

        if($transaction)
        {
                echo "Транзакция успешно прошла";
        $db->commit();
        }
        else
        {
                echo "Транзакция не прошла";
                $db->rollback();
        }
}
catch(PDOException $e)
{
        die("Error: ".$e->getMessage());
}
```