O'REILLY
ONLamp.com
LAMP: THE OPEN SOURCE WEB PLATFORM

Published on **ONLamp.com** (http://www.onlamp.com/)
See this if you're having trouble printing code examples

MySQL Crash Course

by John Coggeshall
12/23/2003

Welcome to another issue of PHP Foundations. Last time, I wrapped up a discussion of an ongoing topic of security and good practices I've dubbed "PHP Paranoia." Today I will be changing gears yet again and discuss a cornerstone of any sizable web application — a database back end. Specifically, I will be spending the next few columns discussing the MySQL database, starting with today's article, which will serve as a crash course in SQL. For those of you who are familiar with relational databases such as MySQL, today's column may not be necessary for you; it contains no PHP code. Instead, I will introduce the concepts of relational databases and the language used to manipulate the data within them. Those of who haven't been exposed to relational databases, however, read on!

Note: In order to work with the examples in this (and the next two) columns, you must already have access to a MySQL database via a MySQL client such as the `mysql` application. If you do not have access to MySQL, it can be downloaded, free of charge, from the MySQL home page in both Unix and Windows flavors. For more information regarding the installation and use of the MySQL server and client, please see the MySQL web site for more information.

Principles of Relational Databases

Relational databases are the cornerstones of today's serious web applications. They provide the "back end" — efficient and, if used properly, fast methods of storing and retrieving mass quantities of data. Although many different relational databases (formally called "Relational Database Management Systems," or RDBMS) exist, such as Oracle and PostgreSQL, all of my discussions will focus on the MySQL RDBMS. The fundamental principles, as well as much of the coming discussion of SQL, will be relevant regardless of the database package used.

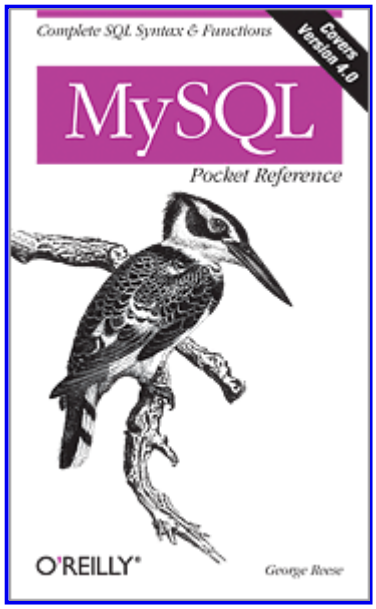
Relational databases are designed to store incredible amounts of data: addresses, email addresses, images, and whatever is required. However, RDBMS packages do not get their strength from what types of data they can store, per se. Instead, the organization of that data within an RDBMS provide the benefits to its users.

Data is organized into one or more databases. These databases are then organized into tables that actually store the data. The best analogy to RDBMS systems is this: consider your day-to-day filing cabinet that is filled with a number of folders, each of which contains a simple table of rows and columns that store the data for that folder. From this perspective, the filing cabinet itself could be considered a database, while each folder could be considered a separate table within that database. To retrieve a specific piece of data from your filing cabinet, you must choose the correct database and folder within that cabinet. Then, you search within that folder for the data you requested.

From a RDBMS standpoint, this is a fairly accurate portrayal of your digital database. Instead of folders, there are simply named tables, and instead of manually searching each table for a specific piece (or pieces) of data, you use a language called SQL (or "Structured Query Language") to retrieve the needed data.

Related Reading

The Structured Query Language



MySQL Pocket Reference
By George Reese

Read Online--Safari
Search this book on Safari:
Go
Only This Book
Code Fragments only

Before you begin to worry about trying to learn an entirely new language on top of attempting to learn PHP, let me calm your fears. SQL is perhaps one of the easiest languages in existence. SQL is, however, absolutely indispensable because it provides the means by which you store and retrieve information in the RDBMS. Although this is not to say SQL queries cannot become incredibly complex, be assured that the underlying principles of the language itself are easy to grasp. Since the best way to understand this is to start right in, let's introduce a few of the major statements.

Note: As I mentioned, SQL can be a very complex language, where statements can take many different forms and, to a beginner, be quite confusing. To elevate that confusion, be aware that the following statements do not reflect the magnitude of the language and have been significantly simplified to focus on practical day to day use of SQL for a beginner. For the complete syntax and use of the following SQL statements, see the MySQL documentation.

Since chances are you have never used relational databases before, your first task is to learn the `CREATE` statement. This statement is used for creating databases as well as tables. Let's start by creating a new database called `fundamentals`:

```
mysql> CREATE
DATABASE fundamentals;
Query OK, 1 row affected (0.01 sec)
```

This statement has essentially created a filing cabinet (database) that can then be filled with folders (tables) to store our data. Obviously, an empty filing cabinet without folders cannot do us much good in storing data, so we'll also need to create tables within this database.

Before we can create any tables within the database, you must understand a bit more about tables. As mentioned before, tables are simply collections of rows and columns. When working with or creating database tables, every column within the table must be assigned a specific data type, governing what kind of data the column represents (such as integer, floating point, or string). Column types can even take on properties, such as a default value or an auto-incrementing number. Since each column within a table must be assigned a data type, let's look at a few of the available data types:

- `INT[(SIZE)] [UNSIGNED] [ZEROFILL]`

  A simple integer between -2147483648 and +2147483647 or, if the `UNSIGNED` attribute is provided, between zero and 4294967295. The `ZEROFILL` attribute indicates that the number should be prefixed with zeros until the number is `SIZE` digits in length.

- `VARCHAR[(SIZE)] [BINARY]`

  A variable-length string that is a maximum of `SIZE` characters in length (where `SIZE` cannot exceed 255). Unless the `BINARY` attribute is provided, this data type is considered case-insensitive and obviously cannot hold binary data.

- `TEXT`

  A case-sensitive string that is a maximum of 65,535 characters in length.

- `DATETIME`

  A date and time ranging from 1000-01-01 00:00:00 to 9999-12-31 23:59:59 (dates are in *YYYY-MM-DD*

HH:MM:SS format).

- `DATE`

  Similar to the `DATETIME` data type, except without the time in *YYYY-MM-DD* format.

Although this is not a complete set of all of the data types a column within a table can have, it gives you an idea of the ways and specific types of data that can be stored. With these in mind, let's create a couple of tables within the `fundamentals` database that will keep track of books. Specifically we'd like to store the title of a book, its author, the publication date, and the home state of the author. For reasons I will explain later, I will be dividing authors and books into two separate tables, `books` and `authors`, with a common `author_id` column between them.

Table 1. The `books` table

book_id	author_id	title	pub_date
1	1	PHP Unleashed	11-01-03
2	1	PHP4 Programming	10-01-02
3	2	Cool Stuff	03-23-02
4	3	Another book	02-01-01

Table 2. The `authors` table

author_id	name	state
1	John Coggeshall	MI
2	Joe Coolguy	AZ
3	Jennifer Author	KS

In order to create this table within our database, we again turn to the `CREATE` statement. However, before we can create a table in the database, we first must tell MySQL what database we are working with using the `USE` statement, as shown:

```
mysql> USE fundamentals;
Database Changed
```

Now we can use the `CREATE` statement to create the `books` and `authors` tables:

```
mysql> CREATE TABLE books(
    book_id   INT AUTO_INCREMENT PRIMARY KEY,
    author_id INT,
    title     VARCHAR(255),
    pub_date  DATE);

Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE authors(
    author_id INT AUTO_INCREMENT PRIMARY KEY,
    name      VARCHAR(255),
    state      VARCHAR(2));

Query OK, 0 rows affected (0.02 sec)
```

These two statements create the two tables we will use throughout our discussion. The `books` table has four columns: a book id number (`book_id`), an author ID number (`author_id`), the title of the book (`title`), and the publication date (`pub_date`). The `authors` table only has three columns: the author ID (`author_id`), the author's name (`name`), and the author's state of residence (`state`).

Note that the book_id and author_id columns have the AUTO_INCREMENT and PRIMARY KEY attributes set. The PRIMARY KEY attribute indicates that no two rows in the table may have the same value for that column — each must be unique. In the case of our tables, this means that the book_id column and the author_id column will each contain an integer value completely unique for each and every row in their respective tables. The AUTO_INCREMENT attribute automatically provides a value for the column. As data is inserted into the tables, the first row will have a value of zero, the second a value of one, the third a value of two, and so on.

Note: Although it is there to simplify things, the AUTO_INCREMENT attribute only takes effect if the NULL value is passed for the column in question. If you specify a value, it will be used instead. Furthermore, if the value inserted is greater than the largest value for that column, the AUTO_INCREMENT attribute will use that value plus one for its next insert value.

To see how your tables are defined in detail once they are created (very useful if you have forgotten the details of a specific table), you can use the DESC (or DESCRIBE) statement:

```
mysql> DESC books;

+-----------+--------------+--------------------+------+-----+---------+----------------+
| Field     | Type         | Collation          | Null | Key | Default | Extra          |
+-----------+--------------+--------------------+------+-----+---------+----------------+
| book_id   | int(11)      | binary             | YES  | PRI | NULL    | auto_increment |
| author_id | int(11)      | binary             | YES  |     | NULL    |                |
| title     | varchar(255) | latin1_swedish_ci  | YES  |     | NULL    |                |
| pub_date  | date         | latin1_swedish_ci  | YES  |     | NULL    |                |
+-----------+--------------+--------------------+------+-----+---------+----------------+

4 rows in set (0.00 sec)
```

Now that the tables have been created, we need to populate the tables with the data reflected in Tables 1 and 2. Use the INSERT statement to insert data into a table:

```
mysql> INSERT INTO books VALUES(1, 1, "PHP Unleashed", "2003-11-01");

Query OK, 1 row affected (0.01 sec)
```

Repeat for each row in the books table.

```
mysql> INSERT INTO authors VALUES(1, "John Coggeshall","MI");

Query OK, 1 row affected (0.01 sec)
```

Repeat for each row in the authors table.

Note: Notice that for the first set of INSERT queries, the date column is represented in *YYYY-MM-DD* format. This is the standard method of storing dates.

To simplify the process of data entry, make sure you manually enter values for the book_id and author_id columns in each respective table as shown above instead of taking advantage of the AUTO_INCREMENT attribute. It is important for later discussions that the tables in the database reflect those found in Tables 1 and 2 exactly.

At this point, you should have a database named fundamentals containing two tables named books and authors. To double check, you can ask the server to show all of the tables within a database with the SHOW statement:

```
mysql> SHOW TABLES;

+----------------------------+
| Tables in fundamentals     |
+----------------------------+
| books                      |
| authors                    |
+----------------------------+
```

```
2 rows in set (0.03 sec)
```

Assuming everything has gone as expected, we are now ready to start requesting data from the database.

Retrieving Data From Tables

Now that we have data in our database, let's retrieve it. To retrieve data from the database (called a "result set"), use the `SELECT` statement:

```
mysql> select * from books;

+---------+-----------+------------------+------------+
| book_id | author_id | title            | pub_date   |
+---------+-----------+------------------+------------+
|       1 | 1         | PHP Unleashed    | 2003-11-01 |
|       2 | 1         | PHP4 Programming | 2002-10-01 |
|       3 | 2         | Cool Stuff       | 2002-03-23 |
|       4 | 3         | Another Book     | 2001-02-01 |
+---------+-----------+------------------+------------+

4 rows in set (0.00 sec)
```

In this case, the above query has returned a result set containing all of the data stored in the `books` table. How does it work exactly? Although the `SELECT` statement is much more complex then this, the general syntax is as follows:

```
SELECT <columns> FROM <tables> ... (additional optional clauses)
```

`<columns>` can be any valid column name for the tables specified in `<tables>` or any valid MySQL function (more on those in the future). If you wanted only a list of all of the book titles in the database, you could use the following query:

```
mysql> SELECT title FROM books;

+------------------+
| title            |
+------------------+
| PHP Unleashed    |
| PHP4 Programming |
| Cool Stuff       |
| Another Book     |
+------------------+

4 rows in set (0.00 sec)
```

You could also select two columns from the `books` table by separating them by a comma:

```
mysql> SELECT title, pub_date FROM books;

+------------------+------------+
| title            | pub_date   |
+------------------+------------+
| PHP Unleashed    | 2003-11-01 |
| PHP4 Programming | 2002-10-01 |
| Cool Stuff       | 2002-03-23 |
| Another Book     | 2001-02-01 |
+------------------+------------+

4 rows in set (0.00 sec)
```

More to Come

That's it for today's PHP Fundamentals! As is sometimes the case with an introduction to a new aspect of PHP, there really isn't any PHP involved at all. Unfortunately for those who are eager to get started using RDBMS packages, it

will be another column's time before I get into using SQL from within PHP. Like all major skills, there is more to understand to make effective use of any of it. In fact, expect to see two more columns discussing pure MySQL before I begin my introduction to the MySQL extension for PHP! Until then, if you would like to learn more about MySQL, you can find complete documentation (probably more than you'd ever want) at the MySQL home page.

John Coggeshall is a a PHP consultant and author who started losing sleep over PHP around five years ago.

---

*Read more **PHP Foundations** columns.*

Return to the PHP DevCenter.