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, Part 3

by John Coggeshall
01/29/2004

Welcome back to another installment of PHP Foundations. In my last two articles, I have been on a bit of a tangent from PHP while I discuss the fundamentals of working with the MySQL RDBMS and MySQL SQL syntax. These concepts are essential to understanding how to use MySQL effectively from PHP. In any case, you should be happy to know today's column will wrap up this tangent so that I can return to the more familiar ground of PHP. I will wrap up my crash-course series on MySQL by discussing some useful functions available when using SQL and PHP, introducing a clause or two for the `SELECT` statement, and explaining a few of the principles behind relating one table to another (after all, it is a "relational" database system).

More `SELECT` Usage

As you have surely realized by now, the `SELECT` statement is incredibly complex and important when working with SQL. To further expand your knowledge of this statement, let's examine a few more `SELECT`-specific clauses. As you may have noticed in the previous columns, the data within the result sets did not appear in any particular order after a query. Since it's often useful to work with sorted results, the `SELECT` statement provides the `ORDER BY` clause, as demonstrated here:

```
mysql> SELECT name FROM authors ORDER BY name;

+-----------------+
| name            |
+-----------------+
| Jennifer Author |
| Joe Coolguy     |
| John Coggeshall |
+-----------------+

3 rows in set (0.00 sec)
```

This asks MySQL to provide a result set consisting of the names of every author in the authors table, sorting them in ascending order alphabetically. If I'd wanted to sort this list in descending order instead, I'd have written:

```
mysql> SELECT name FROM authors ORDER BY name DESC;

+-----------------+
| name            |
+-----------------+
| John Coggeshall |
| Joe Coolguy     |
| Jennifer Author |
+-----------------+

3 rows in set (0.00 sec)
```

Although the default sorting order is ascending, you can explicitly specify it by replacing the `DESC` keyword above with `ASC`. To order a result set by more than one column, simply provide a list of columns (in the order by which they

should be sorted) separated by commas. Although it won't mean much for such a small table, here's how it would look:

```
mysql> SELECT name, state FROM authors ORDER BY name, state;

+-----------------+-------+
| name            | state |
+-----------------+-------+
| Jennifer Author | KS    |
| Joe Coolguy     | AZ    |
| John Coggeshall | MI    |
+-----------------+-------+

3 rows in set (0.00 sec)
```

Note: When specifying multiple columns in an `ORDER BY` clause, each column may have its own `ASC` or `DESC` keyword to specify its sorting order.

`LIMIT`ing `SELECT` Results

It's often also useful to limit the range and number of results. Use the `LIMIT` clause:

```
mysql> SELECT * FROM authors LIMIT 1;

+-----------+-----------------+-------+
| author_id | name            | state |
+-----------+-----------------+-------+
|         1 | John Coggeshall | MI    |
+-----------+-----------------+-------+

1 row in set (0.00 sec)
```

When executed, this will return a single row of the result set. To return a subset of the result set (a range of rows), the `LIMIT` clause can also accept two parameters. The first parameter represents the start row (zero is the first row — it's not tied to any unique ID or primary key) and the second parameter is the number of results to return, as shown:

```
mysql> SELECT * FROM authors LIMIT 1,2;

+-----------+-----------------+-------+
| author_id | name            | state |
+-----------+-----------------+-------+
|         2 | Joe Coolguy     | AZ    |
|         3 | Jennifer Author | KS    |
+-----------+-----------------+-------+

2 rows in set (0.00 sec)
```

Exploring Table Relationships

At this point, we have at least touched on most of the common operations used in day-to-day SQL. However, recall that MySQL is not just a database management system, but rather a relational database management system. To understand why relational databases are important, let's re-examine our two tables:

```
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)

mysql> DESC authors;

+-----------+--------------+--------------------+------+-----+---------+----------------+
| Field     | Type         | Collation          | Null | Key | Default | Extra          |
+-----------+--------------+--------------------+------+-----+---------+----------------+
| author_id | int(11)      | binary             | YES  | PRI | NULL    | auto_increment |
| name      | varchar(255) | latin1_swedish_ci | YES  |     | NULL    |                |
| state     | char(2)      | latin1_swedish_ci | YES  |     | NULL    |                |
+-----------+--------------+--------------------+------+-----+---------+----------------+

3 rows in set (0.01 sec)
```

Looking at the above tables, how would you determine the name of the author of each individual book? As you can see, each table does have an `author_id` column in common, though the title of each book is stored in the `books` table, while author names live in the `authors` table. With what we have learned thus far, there is no reasonable way to create a single result set that contains both the title of a book and the name of its author. To do this, we must somehow merge these two tables together.

All of my discussions of the `SELECT` statement so far has been limited to selecting from one table. Although that has served our purposes just fine thus far, it is time to introduce the concept of table joins. Although joins themselves can be fairly complex, the fundamental principles are straightforward. Consider the following query:

```
mysql> SELECT books.title, authors.name FROM books, authors
       WHERE books.author_id=authors.author_id;

+------------------+-----------------+
| title            | name            |
+------------------+-----------------+
| PHP Unleashed    | John Coggeshall |
| PHP4 Programming | John Coggeshall |
| Cool Stuff       | Joe Coolguy     |
| Another Book     | Jennifer Author |
+------------------+-----------------+

4 rows in set (0.09 sec)
```

This query told the database to retrieve the `title` column from the `books` table (that's `books.title`) and the `name` column from the `authors` table (`authors.name`). Because we are now working with more than one table, the `FROM` portion of the query must also provide the names of the tables with which we are working (`books` and `authors`).

The final portion of the query is the most important in this circumstance, as it defines what and how data from both tables will be included in the final result set. In this case, we have provided the condition `WHERE books.author_id = authors.author_id`. This condition relates the data within the `books` table to the data within the `authors` table by the `author_id` column present in both. Thus, as MySQL constructs the result set, it compares the respective `author_id` columns and matches up the data appropriately. The concept of relating tables together is the single most fundamental idea behind the modern database. It allows you to generate a wealth of reports based on data in your database quickly and easily.

Aggregate Reports Through Joins

Joins make other tricky issues possible, as well as making other functions very valuable. For instance, consider a report of how many books were published by each individual author. Since we are going to require data from two different tables, this obviously would require a join; however, that alone won't suffice. In order to produce such a query, you'll need the `COUNT()` MySQL function and the `GROUP BY` clause.

As I alluded to earlier, SQL is a language focused on extracting data from a database. To facilitate this, SQL provides a wide range of functions for everything from date manipulation to string concatenation and mathematical functions. To count the total number of books per author, we need the `COUNT()` function. As its name implies, this function counts

the number of rows within a result set. It takes a single parameter (the column to count) and returns an integer representing the number of rows counted. For instance, you could count how many rows are in the `books` table using this query:

```
mysql> SELECT COUNT(title) FROM books;

+--------------+
| COUNT(title) |
+--------------+
| 4            |
+--------------+

1 row in set (0.00 sec)
```

We'll use a similar statement to count how many rows met the criteria of our query (the number of books per author). Our query will join the two tables in a similar fashion as the previous join query:

```
mysql> SELECT authors.name, COUNT(books.title) AS books
        FROM authors,books
        WHERE books.author_id = authors.author_id GROUP BY name;

+-----------------+-------+
| name            | books |
+-----------------+-------+
| Jennifer Author |     1 |
| Joe Coolguy     |     1 |
| John Coggeshall |     2 |
+-----------------+-------+

3 rows in set (0.01 sec)
```

As you can see, the query returns two columns. The first column is the name of the author taken from the `authors.name` column, while the second column is a calculated value based on the `COUNT()` function. However, notice that I have renamed the column in the result set, using the `AS` keyword. This is an important detail that will become especially useful when we work with MySQL from a PHP script, since data from a query is usefully accessed by the column name itself.

The last thing that is significantly different about this query than any other we have seen thus far is the `GROUP BY` clause. This clause condenses a particular column (in this case, the `authors.name` column), grouping duplicate entries together into a single row. This is useful in this circumstance, since in this query it wouldn't make a whole lot of sense to have multiple copies of the same name in the result set. In fact, when using the `COUNT()` function with a multi-column result set, you must have a `GROUP BY` clause in order for the query to make sense.

More to Come Next Time

With that, I'll end my MySQL crash course! If you didn't previously know SQL, hopefully by now you have learned enough to start writing simple queries. My next column will begin my introduction to the MySQL extension for PHP. Until then, if you would like to read more about how SQL works, you can find the MySQL documentation on 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.