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 2

by John Coggeshall
01/08/2004

Welcome back! In my previous column, I introduced some of the fundamental concepts behind using MySQL and SQL, explaining how to create tables, populate them with data, and retrieve that data. Today's column will introduce the `UPDATE`, `DELETE`, `ALTER`, and `DROP` statements, as well as a few new clauses to the `SELECT` statement. Although I will not be discussing any PHP code today, these concepts are requirements before we can move forward. With that said, let's get started.

More Fun SQL

The previous article explained how to use the `SELECT` statement to retrieve data from a table within the database. As you may have suspected, the `SELECT` statement is much more complex. There are several different clauses that can control exactly what data you will retrieve from a table. The first of these is the `WHERE` clause.

The `WHERE` clause in SQL provides conditions that must be met in order for a particular row within a table to be part of the result set. To use this clause, simply append it to the end of any applicable query as shown:

```
mysql> SELECT * FROM books WHERE author_id=1;

+---------+-----------+------------------+------------+
| book_id | author_id | title            | pub_date   |
+---------+-----------+------------------+------------+
| 1       | 1         | PHP Unleashed    | 2003-11-01 |
| 2       | 1         | PHP4 Programming | 2002-10-01 |
+---------+-----------+------------------+------------+

2 rows in set (0.00 sec)
```

In this case, I have selected only those rows from the books table where the value of the `author_id` column is `1`. Like any conditional operation in a programming language, the SQL `WHERE` clause supports several different logical operations. Here are the most common, represented in terms of `A` and `B`:

Expression	Evaluation
A = B	True if A equals B
A != B	True if A does not equal B
A <= B	True if A is less than or equal to B
A > B	True if A is greater than or equal to B
A < B	True if A is less than B
A > B	True if A is greater than B
A <=> B	True if A is equal to B (NULL Safe)
A IS NULL	True if A is NULL

A IS NOT NULL	True if A is not NULL
A BETWEEN M AND N	True if A is between values M and N
A NOT BETWEEN M AND N	True if A is not between values M and N
A IN(value, value2, ...)	True if A is one of the listed values
A NOT IN (value, value2, ...)	True if A is not one of the listed values

Note: The reason for a special `NULL` Safe comparison operator (the `<=>` operator) relates to the way MySQL handles `NULL` values. By definition, the comparison of two `NULL` values is false (since `NULL` is an undefined value). You must use this operator when comparing two values in SQL that may be `NULL`.

Along with these operators (which focus mainly on numeric comparisons), the `LIKE` operator matches parts of strings. For example, to return a result set containing all of the books that have `PHP` in their title:

```
mysql> SELECT * FROM books WHERE title LIKE "%PHP%";

+---------+-----------+------------------+------------+
| book_id | author_id | title            | pub_date   |
+---------+-----------+------------------+------------+
| 1       | 1         | PHP Unleashed    | 2003-11-01 |
| 2       | 1         | PHP4 Programming | 2002-10-01 |
+---------+-----------+------------------+------------+

2 rows in set (0.04 sec)
```

The syntax for the `LIKE` operator is `LIKE <pattern>` where the percent sign (`%`) serves as a wildcard. In the above example, the pattern `%PHP%` matches every row whose `title` column has the string `PHP`. This is case-insensitive.

Note: If you would like to match the literal percent character, escape it by doubling it; i.e., `%%`.

Although I have introduced the `WHERE` clause to you in terms of the `SELECT` statement, this technique of identifying particular rows within a table works when deleting and updating records as well. To update specific records within a table, use an `UPDATE` statement instead of a `SELECT` statement. For instance, assume I no longer live in Michigan, having moved to warmer climes. The first step in modifying the records is to identify which record in the `authors` table needs to change:

```
mysql> SELECT * FROM authors WHERE name LIKE "%Coggeshall";

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

1 row in set (0.00 sec)
```

As you can see, my `author_id` is `1`. Since this number is assumed in our table to be a unique identifier (called a primary key) for my record, we can use it in conjunction with the `UPDATE` statement to modify my specific record in the table:

```
mysql> UPDATE authors SET state='FL' WHERE author_id=1;

Query OK, 1 row affected (0.05 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

Note: Although we technically could have used the same `WHERE` clause as the `SELECT` statement used to identify the `author_id` value, it is considered general bad practice to rely on such conditionals as the basis of which rows within a table are modified. For instance, what would have happened had there been two people with the last name `Coggeshall`? Always use a unique identifier as the conditional when performing modifications.

As you can see, the above query modifies a single row (since there is only one row with an `author_id` value of `1`), changing the `state` column to `FL`. The general syntax of the `UPDATE` statement is:

```
UPDATE <table> SET <column>=<value>, <column>=<value>, ... WHERE <conditions>
```

`<table>` represents the table name being manipulated, and the column/value pairs represent the individual values to modify in the rows identified by the `WHERE` clause. Although it is rare, there are times where you would like to universally modify the value of a particular column in a table. In these cases, you may omit the `WHERE` clause entirely. Be very cautious, though.

If it's possible to modify existing records within a table, obviously it must also be possible to remove certain records from tables. This operation is handled through the appropriately named `DELETE` statement. For instance, to remove my record from the `authors` table, use the following statement:

```
mysql> DELETE FROM authors WHERE author_id=1;

Query OK, 1 row affected (0.06 sec)
```

Note: To follow along with the examples later in the column, the row deleted in this example must actually exist. The above is shown only as an example.

As you can see, deleting rows from a table is a fairly straightforward process. The general syntax for the `DELETE` statement is as follows:

```
DELETE FROM <table> WHERE <conditions>
```

As was the case with the `UPDATE` statement, you can omit the `WHERE` clause in the `DELETE` statement, causing the entire contents of the table to be deleted. In SQL, there is no (reasonable) way to recover data once it has been deleted from a table, so take extreme caution to ensure that any `DELETE` queries will behave as expected. It's often useful to issue a `SELECT COUNT(*)` query with the `WHERE` clause to verify that you're affecting only the rows you want to delete.

Also note that unlike the example provided, when removing the entire contents of a table, MySQL will not report how many rows were affected. This is due to the fact that a table could potentially contain thousands of rows and to count each individual row when they all are being removed is simply inefficient.

The final two statements of the day are used to modify tables themselves within a database. The `ALTER` statement changes the properties of tables themselves (for instance, adding an entirely new column to the table definition). If you wanted to add a column that kept track of the ISBN for each book in the `books` table, you could add that column to the table in the following fashion:

```
mysql> ALTER TABLE books ADD isbn VARCHAR(25) AFTER pub_date;

Query OK, 4 rows affected (0.10 sec)

Records: 4 Duplicates: 0 Warnings: 0
```

In this case, I have added a new column named `isbn` of type `VARCHAR` as the last item in the table by placing it after the `pub_date` column (itself previously the last column in the `books` table). To visualize this, use the `DESC` 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    |                |
| isbn      | varchar(25)  | latin1_swedish_ci  | YES  |     | NULL    |                |
+-----------+--------------+--------------------+------+-----+---------+----------------+
```

```
5 rows in set (0.00 sec)
```

To remove this column from the table, use the `ALTER` statement again. However, instead of using the `ADD` qualifier, use `DROP`:

```
mysql> ALTER TABLE books DROP isbn;
Query OK, 4 rows affected (0.00 sec)
Records: 4 Duplicates: 0 Warnings: 0
```

To add a new column to a table as the first column, use the `FIRST` qualifier used instead of `AFTER`:

```
mysql> ALTER TABLE books ADD isbn VARCHAR(25) FIRST;
Query OK, 4 rows affected (0.00 sec)
Records: 4 Duplicates: 0 Warnings: 0
```

Here's the general syntax of the `ALTER` statement:

```
ALTER TABLE <table> [ADD | DROP] <column definition> [FIRST | AFTER <column>]
```

Note: When altering a table that already contains data, the default value for the additional column will be used to populate those records which already exist.

The final statement I will introduce is the `DROP` statement. This statement can delete entire databases as well as individual tables within a database. Since the syntax for this statement is incredibly simple, I will just give you the general syntax:

```
DROP [TABLE | DATABASE] <table or database name>
```

As was the case with the `DELETE` statement, it is extremely important to be cautious when deleting any table or database! Although technically feasible, the process of recovering deleted data from a database can be extremely time-consuming (not to mention extremely costly).

More to Come Next Time

That's it for today. Although we have barely scratched the surface of everything that can be done with an RDBMS such as MySQL, at this point I have introduced every statement that is of immediate use to you. Of course, for those of you who would like a more information, the MySQL web site is an excellent resource. My next column will finish my discussion of MySQL by introducing you to complex `SELECT` statements and some of the more useful internal MySQL functions. Finally, we will be ready to discuss using MySQL from within your PHP scripts! See you then.

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.