MySQL

MySQL is an open source relational database management system. Like many modern database management systems, the program uses SQL (Standard Query Language), a programming language for creating, updating and showing data contained within relational databases. SQL is the language you use to talk to a database management system. Other database management systems besides MySQL also use the SQL language. These include Microsoft’s SQL Server, Sybase Adaptive Server, and Oracle. So, knowing the SQL language is a useful skill that can be used with multiple data management solutions.

With a relatively small subset of commands, MySQL users can carry out many of the most useful routines for setting up databases and manipulating the information those databases contain.

Logging Into MySQL

Once you have logged into laci, you can log into MySQL. To log into MySQL you need to run the program from the UNIX command line and verify that you are a registered user. The syntax for this is "mysql -u username -p", where -u signifies that your user id follows and -p prompts the system to allow you to enter your password.

```
INPUT  mysql -u yourusername -p
```

```
OUTPUT Enter password:
```

Now enter your password next to the prompt.

```
INPUT  Enter password:******
```

```
OUTPUT Welcome to the MySQL monitor. Commands end with ; or g.
Your mysql connection id is 33 to server version 3.23.43
Type 'help;' or 'h' for help. Type 'c' to clear the buffer.
```

Changing Your Password

You’ve all been given random passwords that are very difficult to remember. The first thing we’ll do now that we’re inside of the MySQL program is change your passwords to ones you’ll be able to remember.

```
INPUT  set password = password('yourpassword');
```

```
OUTPUT Query OK, 0 rows affected (0.00 sec)
```
Showing the Available Databases

Next you will want to decide which database you want to use. Before you choose, though, you can get the lay of the land by asking the system to list all the databases that are available.

```
INPUT    SHOW DATABASES;
```

```
OUTPUT   +-----------------------+
        | Database              |
        +-----------------------+
        | movies                |
        | animalhusbandry       |
        | alexc                |
        +-----------------------+

3 rows in set (0.00 sec)
```

Using a Database

Although it doesn’t show up in the example because of space constraints, each of you has access to a database named with a combination of your first name and the first letter of your last name. For this example, we’ll choose Alex’s database.

```
INPUT    USE alexc;
```

```
OUTPUT   Database changed
```

Creating a Table

Tables consist of columns and rows. To define a table, you must tell the system what the columns in the table are and what types of values the system should expect to see in each column (also referred to as attribute or field).

Let’s define a table that holds information about movies. Each movie will be identified by a unique primary key that can be used to reference a specific movie. These lines tell the system to create a table with two columns, one column holds a unique integer for referring to each movie and the other column holds each movie’s title.

```
INPUT    CREATE TABLE movies
        (id INT(10) NOT NULL PRIMARY KEY,
        title VARCHAR(100) NOT NULL);
```

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

To confirm that the table was created as you intended, you can ask the system to describe the table.
DESCRIBE movies;

```
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| id    | int(10)      |      | PRI | 0       |       |
| title | varchar(100) |      |     |         |       |
+-------+--------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

### Inserting a Record

Rows are data in a table. The insert command allows you to create a row in your table by adding a new record with values.

```
INPUT
INSERT INTO movies (id, title) values
(1, "Breakfast At Tiffany\'s");
```

```
OUTPUT
Query OK, 1 row affected (0.00 sec)
```

### Selecting Records

Selecting records allows you to view the data contained in your table. In order to view all the rows contained in a table, you use the * character to establish that the subset of values you wish to view is all of the records the table contains.

```
INPUT
SELECT * FROM movies;
```

```
OUTPUT
<table>
<thead>
<tr>
<th>id</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Breakfast At Tiffany's</td>
</tr>
</tbody>
</table>
1 row in set (0.00 sec)
```

### Exiting the MySQL Monitor

We’re done for the day, so we’ll exit the MySQL monitor.

```
INPUT
exit;
```

```
OUTPUT
Bye
```
Troubleshooting

Before we go any farther with using MySQL, let’s review a basic rule that may keep you from making mistakes and talk about an invaluable resource that will help you in your quest to learn the language.

Commands End with ;

After you have entered the MySQL program, all commands that you want the machine to execute should end in ‘;’ and then return. The program will not execute the command until you type the semicolon. So, remember that you must type both semicolon and return to let the computer know you are finished issuing a command.

More Information on MySQL

For more information on MySQL’s available commands visit www.mysql.com.
MySQL Part Two

All Your Databases Are Belong To Us

Welcome to the MySQL sequel, this week we will teach you many new database styles. Now that you are well on your way to a black belt in MySQL, remember there are many sites on the web that can help you along your way. A simple web search will reveal many sensei’s dedicated to helping you improve your database kung-fu.

Official documentation, tutorials and reference materials are available at mysql.com. They are the one true master, but their lessons can prove cryptic to young students. When in doubt consult www.mysql.com/documentation. It is truly a dojo that knows no pain.

Last week many of you were already asking how to do create more complex queries and tables. First we will review the lessons covered last week and then progress to more advanced moves such as:

- UPDATE RECORD
- DELETE RECORD
- ALTER TABLE
- RENAME TABLE
- DROP TABLE
- AUTO_INCREMENTING FIELDS
- SYNTHETIC KEYS
- SOURCE or using text files as MySQL commands
- SELECT QUERIES
- JOINS

Logging Into MySQL

Once you have logged into laci, you can log into MySQL. To log into MySQL you need to run the program from the UNIX command line and verify that you are a registered user. The syntax for this is "mysql -u username -p", where -u signifies that your user id follows and -p prompts the system to allow you to enter your password.

**INPUT**  mysql -u yourusername -p

**OUTPUT** Enter password:

Now enter your password next to the prompt.

**INPUT**  Enter password:******
Welcome to the MySQL monitor. Commands end with ; or g.
Your mysql connection id is 33 to server version 3.23.43
Type 'help;' or 'h' for help. Type 'c' to clear the buffer.

Showing the Available Databases
Next you will want to decide which database you want to use. Before you choose, though, you can get the lay of the land by asking the system to list all the databases that are available.

```
SHOW DATABASES;
```

```
+-----------------------+
| Database              |
+-----------------------+
| movies                |
| animalhusbandry       |
| alexc                 |
+-----------------------+
3 rows in set (0.00 sec)
```

Using a Database
Just like last week, we want to use your personal database.

```
USE alexc;
```

```
Database changed
```

Describing a Table
By way of review, let’s look at the definition for the table we created last week. To do so, we use the DESCRIBE command.

```
DESCRIBE movies;
```

```
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
|-------+--------------+------+-----+---------+-------|
| id    | int(10)      |      | PRI | 0       |       |
| title | varchar(100) |      |     |         |       |
+-------+--------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
Understanding Synthetic Keys

For our movies table, we’re using the id column as a synthetic key, meaning that we’re artificially creating some data that is unique in order to make sure that there is always a way to access each entry in the table. If we could be absolutely sure that we had another column that contained data which we could use a unique identifier for each record, we could save memory by using the already occurring key rather than a synthetic one.

Altering a Table

We can alter an already existing table in order to refine its utility.

Modifying a Column

Last week, we created a table with a mandatory primary key that was a unique integer for each movie title. Our primary key is sequential, going from 1 to 9999999999. Each record is identified by an id number that is one higher than the previously used id number. The way we defined the table last week, we have to do the addition ourselves. Let’s alter the table so that MySQL will take care of the mundane process of incrementing the key for us.

In order to affect a table that we have already created, we ALTER the table and MODIFY a particular column by adding to its definition. AUTO_INCREMENT tells the system that we want MySQL to supply the id number for us by incrementing the id each time a new entry is added.

**Input**

```
ALTER TABLE movies MODIFY id INT(10) AUTO_INCREMENT;
```

**Output**

```
Query OK, 1 row affected (0.03 sec)
  Records: 1  Duplicates: 0  Warnings: 0
```

Let’s make sure our auto incrementing works by inserting a new entry without specifically stating the new id number ourselves.

**Input**

```
INSERT INTO movies (title) values
("Blow Up");
```

**Output**

```
Query OK, 1 row affected (0.00 sec)
```

As you’ll remember from last week, “SELECT * FROM movies;” will let you view the contents of all the columns and rows in a table. So, we can use the statement to see if the new entry automatically received an id number as we intended.

Adding a Column

So, far we’ve created a table that keeps track of our movie titles, but we haven’t truly taken advantage of the effectiveness of a table because we haven’t related the movie titles
to any significant information. Let’s alter our table so that it can hold pointers to URLs that will give us more information about each movie.

```sql
ALTER TABLE movies ADD url VARCHAR(40) NOT NULL;
```

**Output**

Query OK, 2 rows affected (0.05 sec)
Records: 2 Duplicates: 0 Warnings: 0

### Updating a Record

**UPDATE** updates columns in existing table rows with new values. Let’s change the title of the first record in our movies table to the comedy classic Uncle Buck.

The **SET** clause indicates which columns to modify and the values they should be given. The **WHERE** clause, if given, specifies which rows should be updated. Otherwise all rows are updated.

```sql
UPDATE movies SET url = "http://us.imdb.com/Title?0054698"
WHERE id = 1;
```

**Output**

Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

Remember to use the **SELECT** command to view your results.

### Deleting a Record

The **DELETE** command is very similar to the **UPDATE** command, but it can be potentially hazardous.

**ALL DELETE COMMANDS MUST INCLUDE WHERE CLAUSES.**

**DELETE** deletes rows (or records in more common terms) from table_name that satisfy the condition given by where_definition, and returns the number of records deleted. If you issue a **DELETE** with no **WHERE** clause, all rows are deleted.

We will now delete the first row using the id to choose that particular row.

```sql
DELETE FROM movies WHERE id = 2;
```

**Output**

Query OK, 1 row affected (0.00 sec)
To view the changes we will again use the SELECT command. If you accidentally delete all of your rows, MySQL will return the phrase Empty set to indicate there are no records.

Now we want to DELETE all rows because we will start over with our movies table.

```
INPUT DELETE FROM movies;
```

```
OUTPUT Query OK, 1 row affected (0.00 sec)
```

### Dropping a Table

The DROP command will completely erase your table. All the data and its entire existence. Be very careful when exercising this command. Right now we want to DROP our movies table and then recreate it later.

ALL DELETE COMMANDS MUST INCLUDE WHERE CLAUSES.

DELETE deletes rows (or records in more common terms) from table_name that satisfy the condition given by where_definition, and returns the number of records deleted. If you issue a DELETE with no WHERE clause, all rows are deleted.

We will now delete the first row using the id to choose that particular row.

```
INPUT DROP TABLE movies;
```

```
OUTPUT Query OK, 0 rows affected (0.01 sec)
```

### The SOURCE Command

We will now create two tables, one called movies and one called directors. Instead of using the command line to create and populate the records, we will use a command called SOURCE. SOURCE allows users to run an external file as a MySQL command. This allows you to develop and save commands as scripts in your favorite text editor.

We have already written some scripts for you on LACI in /projects/tutorials. The name of file is directors.sql. When you copy this file to your home directory on LACI you can run the file.

1. Appleshare to the LACI server by selecting the Chooser and then Appleshare.
2. Select Server IP address and then enter laci.gatech.edu
3. When prompted enter your user name and password for LACI.
4. Select the Projects directory, hold down the Apple key and then select the users directory. Hit OK.
5. In the Projects directory select the Tutorials folder and the file directors.sql.
6. Copy this file to the Users directory and the folder with your user name.
7. Now return to the MySQL command line.

```sql
SOURCE directors.sql;
```

```
Query OK, 0 rows affected (0.00 sec)
Query OK, 0 rows affected (0.01 sec)
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

*The multiple outputs are for the various commands in directors.sql*

**Serious Queries**
Welcome again boys and girls to the whimsical land of dynamic database driven design. This time we will tell you the story of PHP, a scripting language with magical powers. Think of your MySQL database as a damsel in distress. Without our hero PHP, the fair maiden remains locked away by her wicked stepmother the command line. PHP can rescue your records and carry them to a land far away.

PHP will not only query your database, but format the output into HTML code. This code is then viewable by anyone with an Internet connection. Additionally PHP gives the programmer robust scripting capabilities.

PHP and MySQL have developed in conjunction so they have similarities in syntax and terminology. Many of the MySQL commands are built into the PHP language.

There are lots of resources on the web, but start your search at the PHP homepage. Full documentation is available at http://www.php.net/docs.php.

PHP vs HTML
To understand the difference between a PHP and HTML lets start with a basic web page. Open a text editor and enter the following:

```html
<HTML>
    Your Name
</HTML>
```

Save the file as name.html on your local drive. Appleshare into LACI and save the file in your Sites folder. View it on the web at http://www.laci.gatech.edu/~yourname/name.html

This page is not too exciting, so let’s try constructing it with PHP. Open your text editor and enter the following:

```php
<HTML>
<?php
    echo "Your Name"
?>
</HTML>
```
Save the file as name.php on your local drive. Appleshare into LACI and save the file in your sites folder. View it on the web at http://www.laci.gatech.edu/~yourname/name.html. The output to the web browser should be identical to the previous one. Again this is nothing special, but the HTML code is generated by PHP when you request the web page.

View the source of the file with your browser. It should look the same as the HTML example. This is because PHP outputs HTML code to the browser. All the user ever sees is a regular web page.

The tag “<?PHP” indicates that our PHP script is starting. The server will see this indicator and parse everything from this tag to “?>” as PHP code and not HTML. You can begin a PHP script with “<?”, but using the “<?PHP” is a better marker when reviewing your code.

The only command in this script is echo. Echo is a way of telling PHP to write the text contained in the following quotations. A semicolon follows our command to signify its end. If there were several commands in this script, they would be separated by semicolons.

PHP is case sensitive so be careful. The command “Echo” may not always work where “echo” will. Also, print is a very similar command to echo and will work as a substitution.

**Variables**

Let’s make our script more complicated and add a variable. A variable sets aside a piece of the computer’s memory where you can store a value. This value can be accessed later in your script. In this exercise, we want to create two variables called $first and $last. Then we will assign them values and have the PHP echo the results.

```html
<HTML>
<?PHP

$first= "Your";
$last= "Name";

echo "$first $last";

?>
</HTML>
```
Save the file as namevar.php on your local drive. Appleshare into LACI and save the file in your sites folder.

All PHP variable names must start with a “$”. After the “$”, the variable’s name follows immediately. After the variables name comes an equals sign, a space, and then double quotes. In the double quotes you place the value of the variable. A variable can be text, numeric, etc. PHP is a weakly-typed language because you do not have to specify the variable or its type before defining its value. After defining the variable use a semicolon to close the command.

To echo our two variables, we place both their names in double quotes with a space in between. Again the dollar sign indicates the variable’s presence.

If your script returns an error, there is a little trick you can use to help debug it. Save your file with a .phps extension and then view it over the web. The PHP code will appear with the syntax will be color coded.

Variations on a Theme

The power of variables is that their values can change. After the initial value is assigned, it can be replaced, updated, or even eliminated entirely. Your name will change vary rarely over the course time, so lets try a more fluctuating example. Time is always changing. This exercise uses the PHP date function to retrieve the current time from the server and place it in a webpage.

Type the following code into the text editor and then save the file as date.php. Again move it to the server. After viewing the page, reload it and the seconds variable will update. As the time changes, so does the variable and the webpage.

```php
<?php
$today = getdate();
$month = $today['month'];
$wday = $today['weekday'];
$mday = $today['mday'];
$year = $today['year'];
$hours = $today['hours'];
$minutes = $today['minutes'];
$seconds = $today['seconds'];

echo "\$wday, \$month \$mday, \$year \$hours:\$minutes:\$seconds";
?>
```
This exercise also uses a function called getdate(). Functions are used in PHP to simplify common actions. Some are built into PHP while others are user-defined.

The getdate() function returns several values which we stored in the variable “$today.” These values are kept in a list or array and are accessed by calling the variable name and then the value name.

**Loops**

Often in the course of programming, you will find yourself executing the same commands multiple times. Suppose you want to display the same information to the screen one hundred times. You could write the print statement over and over again, the exact number of times that you want the information to be displayed, but that isn’t very time efficient. It’s like being punished in elementary school by being made to write the same phase over and over again.

```php
<?
    echo (“I will not waste chalk.”)
    echo (“I will not waste chalk.”)
    echo (“I will not waste chalk.”)
    echo (“I will not waste chalk.”)
    echo (“I will not waste chalk.”)
    //etc.
?>
```

Computers are meant to make mundane tasks easier. Here’s how much easier Bart’s life would be if he could let a computer stand in for him at the beginning of each episode of the Simpsons.

```php
<?
    $i = 1;
    while ($i <= 10) {
        echo “This punishment is not boring and pointless”;
        $i++;
    }
?>
```

Try it with your name.

```php
<?
    //set initial value
    $incrementedValue = 1;
?>
```
//specify the condition on which the loop will end
while ($incrementedValue <= 100){

    //echo your name to the screen
    echo "My Name";
    echo "<BR>";

    /*make sure there that the escape condition will be met by
    incrementing our control variable*/
    $incrementedValue ++;
}
?>

Arrays

Generally, there is not much call to print your name over and over again while creating a
web site, but the same concept works when we want to perform a repetitive operation on
several similar but not necessarily equal values.

For instance, suppose we sold seashells by the seashore. Thanks to global warming there
is a shortage in our seashell supply so we have to raise prices by 20%.

A good structure for storing values that are similar in nature is an array. Here is an array
of prices for our seashells.

<?
    $shellPrices[] = 4.75;
    $shellPrices[] = 5.25;
    $shellPrices[] = 6.00;
    $shellPrices[] = 2.25;

    /*original value, quit condition, incrementing control to make
    sure the condition is met and that each index of the array is
    accessed*/
    for ($i = 0; $i <= 3; $i++) {
        $shellPrices[$i] = $shellPrices[$i]*1.20;
        /*Don’t confuse the character “$” with the $ that must appear at
        the beginning of a variable name*/
        echo "$" . $shellPrices[$i] . "<BR>";
    }
?>