tizag.com

A MySQL Tutorial For Beginners

Written and edited by Tizag.com staff

www.tizag.com

unlock your potential!

Contents

PHP / MySQL Tutorial

MySQL is currently the most popular open source database server in existence. On top of that, it is very commonly used in conjunction with PHP scripts to create powerful and dynamic server-side applications.

MySQL has been criticized in the past for not supporting all the features of other popular and more expensive DataBase Management Systems. However, MySQL continues to improve with each release (currently version 5), and it has become widely popular with individuals and businesses of many different sizes.

What is a Database?

A database is a structure that comes in two flavors: a flat database and a relational database. A relational database is much more oriented to the human mind and is often preferred over the gabble-de-gook flat database that are just stored on hard drives like a text file. MySQL is a relational database.

In a relational structured database there are tables that store data. The columns define which kinds of information will be stored in the table. An individual column must be created for each type of data you wish to store (i.e. Age, Weight, Height).

On the other hand, a row contains the actual values for these specified columns. Each row will have 1 value for each and every column. For example a table with columns (Name, Age, Weight-lbs) could have a row with the values (Bob, 65, 165). If all this relational database talk is too confusing, don't despair. We will talk about and show a few examples in the coming lessons.

Why Use a Database?

Databases are most useful when it comes to storing information that fits into logical categories. For example, say that you wanted to store information of all the employees in a company. With a database you can group different parts of your business into separate tables to help store your information logically. Example tables might be: Employees, Supervisors, and Customers. Each table would then contain columns specific to these three areas. To help store information related to each employee, the Employees table might have the following columns: Hire, Date, Position, Age, and Salary.

Learn MySQL

Before you begin this tutorial you should have a basic knowledge of the information covered in our PHP and HTML tutorials.

This tutorial focuses heavily on using MySQL in a PHP environment. It is aimed at teaching those who have web hosts with PHP and MySQL already installed. If you are unsure, please contact your web host.

MySQL Setup Guide

The easiest way to experiment with MySQL and PHP is to purchase some space on a shared web host.

Although you can set up MySQL manually on your home PC, it can be rather difficult for a beginner to do, and would require more than a few lessons! If you think you've got what it takes, or you're just mentally unstable, head on over to MySQL.com for more information on installing MySQL yourself.

Setting Up MySQL in CPanel

There are many different types of control panels that your shared hosting provider may have. This tutorial assumes that you are using the most popular, CPanel.

First, find the link that allows you to administer MySQL. Within CPanel the icon is labeled *MySQL Databases*. Once there, you will need to do the following before you can start using MySQL.

- Create a new database
- Create a new user with password
- Assign the user to the database

If you have problems with this steps, seek help from your web hosting provider or ask a question in the Tizag Forums.

Helpful Tool - phpMyAdmin!

Also supplied by most hosting services is phpMyAdmin (you can also install it anywhere you want, as it's open source and free). This tool will allow you to view all the MySQL database, tables, and entries, as well as perform SQL queries remotely through a web browser.

Although we will be teaching how to create databases, tables and all other MySQL tasks through PHP, we encourage you to learn about phpMyAdmin. It's easy-to-use interface will allow you to do many common MySQL tasks quickly and easily, saving you many beginner headaches and helping you understand what's going on in a more visual manner.

MySQL Admin

This lesson covers the different options you have available to you for administering your MySQL service after it is successfully installed. If you already have that base covered feel free to skip on to the next lesson.

MySQL Command Line

If you are an old-school programmer that has no need for a graphical user interface, then you can simply use any command line interface to execute MySQL queries.

Those of you with MySQL installed on your Microsoft Windows operating system can reach the command line by going to the Start Menu and choosing "**Run**...". Type the keyword "**cmd**" into the text field and press **Enter** to launch Window's command line interface.

MySQL GUI

With so many free MySQL administration tools available, many developers favor these free Graphical User Interfaces over the command line. The most popular options include:

- **.:.** phpMyAdmin - A popular web interface that is included with almost every type of Shared, Virtual or Dedicated hosting solution.
- **.:.** MySQL Administrator - A powerful tool developed by the folks at MySQL.com.
- **.:.** Navicat - A purchasable MySQL admin tool for Windows, Mac and Linux.

MySQL phpMyAdmin

As previously mentioned, the very popular phpMyAdmintool should come with your web hosting plan.

MySQL Administrator

This tool comes from the creators of MySQL, so you can be assured they have a solid understanding of database optimization and stability for power users. There are currently two versions of MySQL Administrator: 1.0 and 1.1. MySQL.com recommends you use 1.1 if your MySQL installation is 4.0, 4.1 or 5.0. Read more about theMySQL Administrator on MySQL.com's web site.

MySQL Navicat

Navicat comes with a 30-day trial so you can play around and see if you feel like dropping the cash for this MySQL administration product. A brief overview of their productNavicat Admin can be found on their website. The cost of this product is around $100.

MySQL Syntax

The great thing about everything you do in MySQL is that the "code" is very easy for humans to read, as opposed to harder programming languages like C or C++. Very few special characters and symbols are required to create a MySQL query, and most queries consist entirely of English words!

Strategies to Learn MySQL

The MySQL language is not as complicated as most programming languages, so the best way to learn is with direct examples. Because this tutorial focuses on the combination of MySQL and PHP, most of the examples are ready for you to copy, paste, and run on your web server.

CAPITALIZATION in MySQL Queries

There are many keywords in MySQL, and a good programming habit when using ANY of these words is to capitalize them. This helps draw them out from the rest of the code and makes them much easier to read. Below is an example of a MySQL query written in PHP that retrieves all the data from a MySQL table named "example".

.:. $result = mysql_query("SELECT * FROM example")

That line of code is valid PHP, but it also contains valid MySQL. The text that appears between the quotations "SELECT * FROM example", is the MySQL code.

As you probably can tell "SELECT" and "FROM" are the MySQL keywords used in this query. Capitalizing them allows you to tell from a quick glance that this query selects data from a table.

You can view a complete list of MySQL keywords at List of Reserved MySQL Words, but don't worry about memorizing them. You could program relentlessly in MySQL for years without ever using all of the MySQL keywords.

Learn at Your Own Pace

When learning MySQL, it is best to take it one lesson at a time and stop when you feel frustrated. Do not worry if it takes you more than a week to finish this tutorial. If you take the time to progress slowly, you will be much more well-informed about the MySQL database system than if you rushed through it in one sitting.

MySQL Database

A MySQL database is nothing in itself. Rather a MySQL database is a way of organizing a group of tables. If you were going to create a bunch of different tables that shared a common theme, you would group them into one database to make the management process easier.

Creating Your First Database

Most web hosts do not allow you to create a database directly through a PHP script. Instead they require that you use the PHP/MySQL administration tools on the web host control panel to create these databases. Create a database and assign a new user to this database. For all of our beginning examples we will be using the following information:

- **Server** - localhost
- **Database** - test
- **Table** - example
- **Username** - admin
- **Password** - 1admin

Note: The table may change in the advanced lessons, but everything else will remain the same!

The server is the name of the server we want to connect to. Because all of our scripts are going to be placed on the server where MySQL is located the correct address is *localhost*. If the MySQL server was on a different machine from where the script was running, then you would need to enter the correct url (ask your web host for specifics on this).

Your database, table, username, and password do not have to match ours. If you choose a different set of login information, remember to insert your own information when copying the scripts in this tutorial.

Status Check

So far, you should have created a new database and assigned a user to it. You should **not** have created a table yet. If you are up-to-date, then continue the tutorial. We will be making our first table in an upcoming lesson.

MySQL Connect

Before you can do anything with MySQL in PHP you must first establish a connection to your web host's MySQL database. This is done with the MySQL connect function.

MySQL localhost

If you've been around the internet a while, you'll know that IP addresses are used as identifiers for computers and web servers. In this example of a connection script, we assume that the MySQL service is running on the same machine as the script.

When the PHP script and MySQL are on the same machine, you can use *localhost* as the address you wish to connect to. *localhost* is a shortcut to just have the machine connect to itself. If your MySQL service is running at a separate location you will need to insert the IP address or URL in place of *localhost*. Please contact your web host for more details if *localhost* does not work.

PHP & MySQL Code:

```
<?php
mysql_connect("localhost", "admin", "1admin") or die(mysql_error());
echo "Connected to MySQL<br />";
?>
```

Display:

```
Connected to MySQL
```

If you load the above PHP script to your webserver and everything works properly, then you should see "Connected to MySQL" displayed when you view the .php page.

The mysql_connect function takes three arguments. Server, username, and password. In our example above these arguments were:

- **Server** - localhost
- **Username** - admin
- **Password** - 1admin

The "or die(mysql..." code displays an error message in your browser if --you've probably guessed it -- there is an error in processing the connection! Double-check your username, password, or server if you receive this error.

Choosing the Working Database

After establishing a MySQL connection with the code above, you then need to choose which database you will be using with this connection. This is done with the *mysql_select_db* function.

PHP & MySQL Code:

```
<?php
mysql_connect("localhost", "admin", "1admin") or die(mysql_error());
echo "Connected to MySQL<br />";
mysql_select_db("test") or die(mysql_error());
echo "Connected to Database";
?>
```

Display:

```
Connected to MySQL
Connected to Database
```

Status Check

So far you should have made a MySQL connection and chosen the working database. If you are up-to-date then continue the tutorial. We will be making our first table in the next lesson.

MySQL Tables

A MySQL table is completely different than the normal table that you eat dinner on. In MySQL and other database systems, the goal is to store information in an orderly fashion. The table gets this done by making the table up of columns and rows.

The columns specify what the data is going to be, while the rows contain the actual data. Below is how you could imagine a MySQL table. (C = Column, R = Row)

	C1 (Namge)	C2 (Age)	C3 (Weight)
R1	R1 C1 (John)	R1 C2 (21)	R1 C3 (120)
R2	R2 C1 (Big Sally)	R2 C2 (27)	R2 C3 (400)
R3	R3 C1 (Tiny Tim)	R3 C2 (6)	R3 C3 (35)
R4	R4 C1 (Normal Ned)	R4 C2 (35)	R4 C3 (160)

We added the row and column number (R# C#) so that you can see that a row is side-to-side, while a column is up-to-down. In a real MySQL table only the value would be stored, not the R# and C#!

This table has three categories, or "columns", of data: Name, Age, and Weight. This table has four entries, or in other words, four rows.

Create Table MySQL

Before you can enter data (rows) into a table, you must first define what kinds of data will be stored (columns). We are now going to design a MySQL query to summon our table from database land. In future lessons we will be using this table, so be sure to enter this query correctly!

PHP & MySQL Code:

```
<?php
// Make a MySQL Connection
mysql_connect("localhost", "admin", "1admin") or die(mysql_error());
mysql_select_db("test") or die(mysql_error());

// Create a MySQL table in the selected database
mysql_query("CREATE TABLE example(
id INT NOT NULL AUTO_INCREMENT,
PRIMARY KEY(id),
 name VARCHAR(30),
 age INT)")
 or die(mysql_error());

echo "Table Created!";

?>
```

Display:

```
Table Created!
```

Wow! That's a lot of code all at once! Let's get down in the dirt and figure this stuff out. We will be going through the code line by line.

'mysql_query ("CREATE TABLE example'

The first part of the *mysql_query* told MySQL that we wanted to create a new table. The two capitalized words are reserved MySQL keywords.

The word "example" is the name of our table, as it came directly after "CREATE TABLE". It is a good idea to use descriptive names when creating a table, such as: employee_information, contacts, or customer_orders. Clear names will ensure that you will know what the table is about when revisiting it a year after you make it.

'id INT NOT NULL AUTO_INCREMENT'

Here we create a column "id" that will automatically increment each time a new entry is added to the table. This will result in the first row in the table having an *id* = 1, the second row *id* = 2, the third row *id* = 3, and so on.

The column "id" is not something that we need to worry about after we create this table, as it is all automatically calculated within MySQL.

Reserved MySQL Keywords: Here are a few quick definitions of the reserved words used in this line of code:

.:. INT - This stands for integer or whole number. 'id' has been defined to be an integer.
.:. NOT NULL - These are actually two keywords, but they combine together to say that this column cannot be null. An entry is NOT NULL only if it has some value, while something with no value is NULL.
.:. AUTO_INCREMENT - Each time a new entry is added the value will be incremented by 1.

'PRIMARY KEY (id)'

PRIMARY KEY is used as a unique identifier for the rows. Here we have made "id" the PRIMARY KEY for this table. This means that no two ids can be the same, or else we will run into trouble. This is why we made "id" an auto-incrementing counter in the previous line of code.

'name VARCHAR(30),'

Here we make a new column with the name "name"! VARCHAR stands for "variable character". "Character" means that you can put in any kind of typed information in this column (letters, numbers, symbols, etc). It's "variable" because it can adjust its size to store as little as 0 characters and up to a specified maximum number of characters.

We will most likely only be using this name column to store characters (A-Z, a-z). The number inside the parentheses sets the maximum number of characters. In this case, the max is 30.

'age INT,'

Our third and final column is age, which stores an integer. Notice that there are no parentheses following "INT". MySQL already knows what to do with an integer. The possible integer values that can be stored in an "INT" are -2,147,483,648 to 2,147,483,647, which is more than enough to store someone's age!

'or die(mysql_error());'

This will print out an error if there is a problem in the table creation process.

Your Homework

Using the MySQL administration tool that your web host has, check to see if the table was created correctly. Afterwards, try creating a few of your own, with PHP or with a MySQL administration tool, to be sure that you have gotten the hang of it.

MySQL Insert

When data is put into a MySQL table it is referred to as inserting data. When inserting data it is important to remember the exact names and types of the table's columns. If you try to place a 500 word essay into a column that only accepts integers of size three, you will end up with a nasty error!

Inserting Data Into Your Table

Now that you have created your table, let's put some data into that puppy! Here is the PHP/MySQL code for inserting data into the "example" table we created in the previous lesson.

PHP & MySQL Code:

```
<?php
// Make a MySQL Connection
mysql_connect("localhost", "admin", "1admin") or die(mysql_error());
mysql_select_db("test") or die(mysql_error());

// Insert a row of information into the table "example"
mysql_query("INSERT INTO example
(name, age) VALUES('Timmy Mellowman', '23' ) ")
or die(mysql_error());

mysql_query("INSERT INTO example
(name, age) VALUES('Sandy Smith', '21' ) ")
or die(mysql_error());

mysql_query("INSERT INTO example
(name, age) VALUES('Bobby Wallace', '15' ) ")
or die(mysql_error());

echo "Data Inserted!";

?>
```

Display:

```
Data Inserted!
```

This code is much simpler to understand than the create table code, as will be most of the MySQL queries you will learn in the rest of this tutorial. Once again, we will cover the code line by line.

'mysql_query("INSERT INTO example'

Again we are using the mysql_query function. "INSERT INTO" means that data is going to be put into a table. The name of the table we specified to insert data into was "example".

'(name, age) VALUES('Timmy Mellowman', '23') ")'

"(name, age)" are the two columns we want to add data into. "VALUES" means that what follows

is the data to be put into the columns that we just specified. Here we enter the name Timmy Mellowman for "name", and 23 for "age".

Be sure to note the location and number of apostrophes and parentheses in the PHP code, as this is where a lot of beginner PHP/MySQL programmers run into problems.

Review & Homework

If all goes as well, this .php page will add a three people to the "example" table every time it is run. Be sure to use your MySQL administration program provided by your web host to ensure that the data was inserted into your table correctly.

Be careful not to run this script more than once, otherwise you will be inserting the same people, multiple times. This is called inserting duplicate records and is usually avoided.

MySQL Query

So far we have seen a couple different uses of PHP's *mysql_query* function and we'll be seeing more of it as nearly all MySQL in PHP is done through the MySQL Query function. We have already created a new table and inserted data into that table. In this lesson we will cover the most common MySQL Query that is used to retrieve information from a database.

Retrieving Data With PHP & MySQL

Usually most of the work done with MySQL involves pulling down data from a MySQL database. In MySQL, data is retrieved with the "SELECT" keyword. Think of SELECT as working the same way as it does on your computer. If you wanted to copy some information in a document, you would first select the desired information, then copy and paste.

Using MySQL SELECT & FROM

Before attempting this lesson, be sure that you have created a table that contains some data, preferably the same data that we had in the MySQL Insert lesson. In this example, we will output the first entry of our MySQL "examples" table to the web browser.

PHP & MySQL Code:

```
<?php
// Make a MySQL Connection
mysql_connect("localhost", "admin", "1admin") or die(mysql_error());
mysql_select_db("test") or die(mysql_error());

// Retrieve all the data from the "example" table
$result = mysql_query("SELECT * FROM example")
or die(mysql_error());

// store the record of the "example" table into $row
$row = mysql_fetch_array( $result );
// Print out the contents of the entry

echo "Name: ".$row['name'];
echo " Age: ".$row['age'];

?>
```

Display:

```
Name: Tim Mellowman Age: 23
```

This is an example of how to use MySQL's SELECT statement in PHP. Although the MySQL code is simple, printing out the information with PHP is somewhat more involved.

Below is a step-by-step walkthrough of the code.

'$result = mysql_query("SELECT * FROM example")'

When you perform a SELECT query on the database it will return a MySQL Resource that hols everything from your MySQL table, "example". We want to use this Resource in our PHP code, so we need to store it in a variable, *$result*.

'SELECT * FROM example'

Yes, this is a partial repeat of the same line of code, but we wanted to explain this MySQL statement in greater detail again!

In English, this line of code reads "Select every entry from the table example". The asterisk is the wild card in MySQL which just tells MySQL to include every single column for that table.

'$row = mysql_fetch_array($result);'

$result is now a MySQL Resource containing data from your MySQL table, "example". Data is being stored in *$result*, but it is not yet visible to visitors of your website. When we pass *$result* into the *mysql_fetch_array* function -- mysql_fetch_array($result) -- an associative array (name, age) is returned.

In our MySQL table "example," there are only two fields that we care about: name and age. These names are the keys to extracting the data from our associative array. To get the name we use **$row['name']** and to get the age we use **$row['age']**.

PHP is case sensitive when you reference MySQL column names, so be sure to use capitalization in your PHP code that matches the MySQL column names!

Continuing the Example

In this lesson, we learned how to get the first entry from a MySQL table and output to the browser using PHP. In the next lesson we will see how to retrieve every entry of a table and put it into a nicely formatted table. However, we recommend that you first understand the PHP and MySQL code in this lesson before proceeding.

MySQL Fetch Array

MySQL doesn't have a Fetch Array function. *mysql_fetch_array* is actually a PHP function that allows you to access data stored in the result returned from a successful *mysql_query*. If you have been jumping around our MySQL Tutorial then you would have already seen this function popping up all over the place.

This lesson will teach you how and why to use *mysql_fetch_array* in your PHP Scripts.

mysql_fetch_array: Why Use It?

Do you know what is returned when you used the *mysql_query* function to query a MySQL database? It isn't something you can directly manipulate, that is for sure. Here is a sample *SELECT* query of a table we created in the MySQL Create Table lesson.

PHP and MySQL Code:

```
<?php
$result = mysql_query("SELECT * FROM example");
?>
```

The value that *mysql_query* returns and stores into *$result* is a special type of data, it is a MySQL Resource. Additional PHP functions are required to extract the data from this Resource.

A Row of Data

The *mysql_fetch_array* function takes a MySQL query resource as an argument (*$result*) and returns the first row of data returned by the *mysql_query*. Our table *example* basically looks like the table below.

name	age
Timmy Mellowman	23
Sandy Smith	21
Bobby Wallace	15

The first row of data in this table is "Timmy Mellowman" and "23". When we fetch an array from our MySQL Resource *$result* it should have Timmy's name and age in it.

Getting a Row of Data using mysql_fetch_array

mysql_fetch_array returns the first row in a MySQL Resource in the form of an associative array. The columns of the MySQL Result can be accessed by using the column names of the table. In our table *example* these are: name and age. Here is the code to print

out the first MySQL Result row.

PHP and MySQL Code:

```
<?php
// Make a MySQL Connection
$query = "SELECT * FROM example";

$result = mysql_query($query) or die(mysql_error());


$row = mysql_fetch_array($result) or die(mysql_error());
echo $row['name']. " - ". $row['age'];
?>
```

Display:

```
Timmy Mellowman - 23
```

This is just what we expected would happen! Now, the cool thing about *mysql_fetch_array* is that you can use it again on the same MySQL Resource to return the second, third, fourth and so on rows. You can keep doing this until the MySQL Resource has reached the end (which would be three times in our example).

Sounds like an awfully repetitive task. It would be nice if we could get all our results from a MySQL Resource in an easy to do script.

Fetch Array While Loop

As we have said, the *mysql_fetch_array* function returns an associative array, but it **also** returns FALSE if there are no more rows to return! Using a PHP While Loop we can use this information to our advantage.

If we place the statement "$row = mysql_fetch_array()" as our while loop's conditional statement we will accomplish two things:

1. We will get a new row of MySQL information that we can print out each time the while loop checks its conditional statement.
2. When there are no more rows the function will return FALSE causing the while loop to stop!

Now that we know what we need to do and how to go about doing it, the code pretty much writes itself, so let's move on to the next lesson. Just kidding! Here is the code that will print out all the rows of our MySQL Resource.

PHP and MySQL Code:

```
<?php
// Make a MySQL Connection
$query = "SELECT * FROM example";

$result = mysql_query($query) or die(mysql_error());


while($row = mysql_fetch_array($result)){
 <indent>echo $row['name']. " - ". $row['age'];
 echo "<br />";</indent>
}
?>
```

Display:

```
Timmy Mellowman - 23
Sandy Smith - 21
Bobby Wallace - 15
```

And there we have all the rows from our *example* table! You could apply this script to any MySQL table as long as you change both the *table name* in the query and the *column names* that we have in the associative array.

MySQL Select

You have seen two types of MySQL queries thus far: the query which we used to create a table and the query we used to insert data into our newly created table. The query in this lesson is SELECT, which is used to **get** information from the database, so that its data can be used in our PHP script.

Retrieving Information from MySQL

Finally, we get to use the data in our MySQL database to create a dynamic PHP page. In this example we will select everything in our table "example" and put it into a nicely formatted HTML table. Remember, if you don't understand the HTML or PHP code, be sure to check out the HTML and/or PHP Tutorial(s).

PHP & MySQL Code:

```
<?php
// Make a MySQL Connection
mysql_connect("localhost", "admin", "1admin") or die(mysql_error());
mysql_select_db("test") or die(mysql_error());

// Get all the data from the "example" table
$result = mysql_query("SELECT * FROM example")
or die(mysql_error());

echo "<table border='1'>";
echo "<tr> <th>Name</th> <th>Age</th>
</tr>";
// keeps getting the next row until there are no more to get
while($row = mysql_fetch_array( $result )) {
 // Print out the contents of each row into a table
 echo "<tr><td>";
 echo $row['name'];
 echo "</td><td>";
 echo $row['age'];
 echo "</td></tr>";
}

echo "</table>";
?>
```

Name	Age
Timmy Mellowman	23
Sandy Smith	21
Bobby Wallace	15

Because we only had three entries in our table, three rows appeared above. If you added more entries to your database's table, then you would see each additional row appear in the above table. If you do not understand the above PHP, you can view our PHP Array Tutorial & PHP Loop Tutorial.

'$result = mysql_query...'

When you select items from a database using *mysql_query*, the data is returned as a MySQL result. Since we want to use this data in our table we need to store it in a variable. *$result* now holds the result from our *mysql_query*.

'SELECT * FROM example'

In English, this line of code reads "Select everything from the table example". The asterisk is the wild card in MySQL which just tells MySQL to retrieve **every single field** from the table.

'while($row = mysql_fetch_array($result)'

The *mysql_fetch_array* function gets the next-in-line associative array from a MySQL result. By putting it in a while loop it will continue to fetch the next array until there is no next array to fetch. This function can be called as many times as you want, but it will return FALSE when the last associative array has already been returned.

By placing this function within the conditional statement of the while loop, we can kill two birds with one stones.

1. We can retrieve the next associative array from our MySQL Resource, *$result*, so that we can print out the name and age of that person.
2. We can tell the while loop to stop printingn out information when the MySQL Resource has returned the last array, as False is returned when it reaches the end and this will cause the while loop to halt.

In our MySQL table "example" there are only two fields that we care about: name and age. These fields are the keys to extracting the data from our associative array. To get the name we use $row['name'] and to get the age we use $row['age'].

Practice What You Have Learned

Use the query that we have provided or make a new one and try putting it into a formatted HTML table. It might be useful to try out other methods of HTML formatting as well. See which one you like best!

By now you should be starting to understand how powerful PHP and MySQL are when used together. The tasks that you can complete with MySQL and PHP would be nearly impossible to do by hand in HTML. Imagine trying to create an HTML table of 6000 entries without using a MySQL database and a PHP while loop!

MySQL Where

In a previous lesson we did a SELECT query to get all the data from our "example" table. If we wanted to select only certain entries of our table, then we would use the keyword WHERE.

WHERE lets you specify requirements that entries must meet in order to be returned in the MySQL result. Those entries that do not pass the test will be left out. We will be assuming the data from a previous lesson for the following examples.

Being Selective With Your MySQL Selection

There are three entries in our "example" table: Tim, Sandy, and Bobby. To select Sandy only we could either specify Sandy's age (21) or we could use her name (Sandy Smith). In the future there may be other people who are 21, so we will use her name as our requirement.

WHERE is used in conjuction with a mathematical statement. In our example we will want to select all rows that have the string "Sandy Smith" in the "names" column (mathematically: {name column} = "Sandy Smith"). Here's how to do it.

PHP & MySQL Code:

```
<?php
// Make a MySQL Connection
mysql_connect("localhost", "admin", "1admin") or die(mysql_error());
mysql_select_db("test") or die(mysql_error());

// Get a specific result from the "example" table
$result = mysql_query("SELECT * FROM example
 WHERE name='Sandy Smith'") or die(mysql_error());

// get the first (and hopefully only) entry from the result
$row = mysql_fetch_array( $result );
// Print out the contents of each row into a table
echo $row['name']." - ".$row['age'];
?>
```

Display:

```
Sandy Smith - 21
```

MySQL Wildcard Usage '%'

If you wanted to select every person in the table who was in their 20's, how could you go about doing it? With the tools you have now, you could make 10 different queries, one for each age 20, 21, 22...but that seems like more work than we need to do.

In MySQL there is a "wildcard" character '%' that can be used to search for partial matches in your database. The '%' tells MySQL to ignore the text that would normally appear in place of the wildcard.

For example '2%' would match the following: 20, 25, 2000000, 2avkldj3jklsaf, and 2!

On the other hand, '2%' would not match the following: 122, a20, and 32.

MySQL Query WHERE With Wildcard

To solve our problem from before, selecting everyone who is their 20's from or MySQL table, we can utilize wildcards to pick out all strings starting with a 2.

PHP & MySQL Code:

```
<?php
// Connect to MySQL
// Insert a row of information into the table "example"
$result = mysql_query("SELECT * FROM example WHERE age LIKE '2%' ")
or die(mysql_error());

// keeps getting the next row until there are no more to get
while($row = mysql_fetch_array( $result )) {
 // Print out the contents of each row
 echo $row['name']." - ".$row['age']. "<br />";
}
?>
```

Display:

```
Timmy Mellowman - 23
Sandy Smith - 21
```

You can use this wildcard at the beginning, middle, and end of the string. Experiment with it so you can see for yourself how powerful this little trick can be.

Note: The wildcard was used for example purposes only. If you really wanted to explicilty select people who are in their 20's you would use greater than 19 and less than 30 to define the 20's range. Using a wildcard in this example would select unwanted cases, like a 2 year old and your 200 year old great-great-great-grandparents.

MySQL Order By

It would be nice to be able to make MySQL results easier to read and understand. A common way to do this in the real world is to order a big list of items by name or amount. The way to order your result in MySQL is to use the ORDER BY statement.

What ORDER BY does is take the a column name that you specify and sort it in alphabetical order (or numeric order if you are using numbers). Then when you use *mysql_fetch_array* to print out the result, the values are already sorted and easy to read.

Ordering is also used quite frequently to add additional functionality to webpages that use any type of column layout. For example, some forums let you sort by date, thread title, post count, view count, and more.

Sorting a MySQL Query - ORDER BY

Let's use the same query we had in MySQL Select and modify it to ORDER BY the person's age. The code from MySQL Select looked like...

PHP & MySQL Code:

```
<?php
// Make a MySQL Connection
mysql_connect("localhost", "admin", "1admin") or die(mysql_error());
mysql_select_db("test") or die(mysql_error());

// Get all the data from the "example" table
$result = mysql_query("SELECT * FROM example")
or die(mysql_error());

echo "<table border='1'>";
echo "<tr> <th>Name</th> <th>Age</th>
</tr>";
// keeps getting the next row until there are no more to get
while($row = mysql_fetch_array( $result )) {
 // Print out the contents of each row into a table
 echo "<tr><td>";
 echo $row['name'];
 echo "</td><td>";
 echo $row['age'];
 echo "</td></tr>";
}

echo "</table>";
?>
```

Name	Age
Timmy Mellowman	23
Sandy Smith	21
Bobby Wallace	15

What we need to do is add on to the existing MySQL statement "SELECT * FROM example" to include our new ordering requirement. When you choose to order a column, be sure that your ORDER BY appears after the SELECT ... FROM part of the MySQL statement.

PHP & MySQL Code:

```
<?php
// Make a MySQL Connection
mysql_connect("localhost", "admin", "1admin") or die(mysql_error());
mysql_select_db("test") or die(mysql_error());

// Get all the data from the "example" table
$result = mysql_query("SELECT * FROM example ORDER BY age")
or die(mysql_error());

echo "<table border='1'>";
echo "<tr> <th>Name</th> <th>Age</th>
</tr>";
// keeps getting the next row until there are no more to get
while($row = mysql_fetch_array( $result )) {
 // Print out the contents of each row into a table
 echo "<tr><td>";
 echo $row['name'];
 echo "</td><td>";
 echo $row['age'];
 echo "</td></tr>";
}

echo "</table>";
?>
```

Name	Age
Bobby Wallace	15
Sandy Smith	21
Timmy Mellowman	23

Presto! We have an ordered MySQL result! Notice that we didn't have to change any of our PHP code. Remember this whenever you're editing a PHP script that uses MySQL. Sometimes it may be easier to just tweak your MySQL query instead of trying to mess around in PHP.

MySQL Joins

Thus far we have only been getting data from one table at a time. This is fine for simple takes, but in most real world MySQL usage you will often need to get data from multiple tables in a single query.

The act of *joining* in MySQL refers to smashing two or more tables into a **single** table. This means everything you have learned so far can be applied after you've created this new, *joined* table.

MySQL Join Table Setup

We like to show examples and code before we explain anything in detail, so here is how you would combine two tables into one using MySQL. The two tables we will be using relate to a families eating habits.

Position	Age
Dad	41
Mom	45
Daughter	17
Dog	

Meal	Position
Steak	Dad
Salad	Mom
Spinach Soup	
Tacos	Dad

The important thing to note here is that the column *Position* contains information that can tie these two tables together. In the "family" table, the *Position* column contains all the members of the family and their respective ages. In the "food" table the *Position* column contains the family member who enjoys that dish.

It's only through a shared column relationship such as this that tables can be *joined* together, so remember this when creating tables you wish to have interact with each other.

MySQL Join Simple Example

Let's imagine that we wanted to SELECT all the dishes that were liked by a family

member. If you remember from the previous lesson, this is a situation when we need to use the WHERE clause. We want to SELECT all the dishes WHERE a family member likes it.

We will be performing a generic *join* of these two tables using the *Position* column from each table as the connector.

Note: This example assumes you have created the MySQL tables "food" and "family". If you do not have either of them created, you can either create them using our MySQL Create Table lesson or do it manually yourself.

PHP and MySQL Code:

```
<?php
// Make a MySQL Connection
// Construct our join query
$query = "SELECT family.Position, food.Meal ".
 "FROM family, food ".
 "WHERE family.Position = food.Position";

$result = mysql_query($query) or die(mysql_error());


// Print out the contents of each row into a table
while($row = mysql_fetch_array($result)){
 echo $row['Position']. " - ". $row['Meal'];
 echo "<br />";
}
?>
```

The statement "WHERE family.Position = food.Position" will restrict the results to the rows where the *Position* exists in **both** the "family" and "food" tables.

Display:

```
Dad - Steak
Mom - Salad
Dad - Tacos
```

Those are the results of our PHP script. Let's analyze the tables to make sure we agree with these results.

Position	Age
Dad	41
Mom	45
Daughter	17
Dog	

Meal	Position
Steak	***Dad***
Salad	***Mom***
Spinach Soup	
Tacos	***Dad***

Our results show that there were three meals that were liked by family members. And by manually perusing the tables it looks like there were indeed three meals liked by family members.

Note: This is a very simple example of a join. If you do not understand it yet do not despair. Joins are a very hard concept to grasp for beginning MySQL developers.

MySQL LEFT JOIN

In the previous lesson, Mysql Joins we learned how to do a basic join of two tables. This lesson will teach you how to do a specialized join: *left join*.

MySQL LEFT JOIN Explanation

How is a LEFT JOIN different from a normal join? First of all, the syntax is quite different and somewhat more complex. Besides looking different, the LEFT JOIN gives extra consideration to the table that is on the left.

Being "on the left" simply refers to the table that appears before the LEFT JOIN in our SQL statement. Nothing tricky about that.

This extra consideration to the left table can be thought of as special kind of preservation. Each item in the left table will show up in a MySQL result, **even** if there isn't a match with the other table that it is being joined to.

MySQL Join and LEFT JOIN Differences

Here are the tables we used in the previous Mysql Joins lesson.

Position	Age
Dad	41
Mom	45
Daughter	17
Dog	

Meal	Position
Steak	Dad
Salad	Mom
Spinach Soup	
Tacos	Dad

We executed a simple query that selected all meals that were liked by a family member with this simple join query:

Simplified MySQL Query:

```
SELECT food.Meal, family.Position
FROM family, food
WHERE food.Position = family.Position
```

Result:

```
Dad - Steak
Mom - Salad
Dad - Tacos
```

When we decide to use a LEFT JOIN in the query instead, all the family members be listed, even if they do not have a favorite dish in our *food* table.

This is because a left join will preserve the records of the "left" table.

MySQL LEFT JOIN Example

The code below is the exact same as the code in the previous lesson, except the LEFT JOIN has now been added to the query. Let's see if the results are what we expected.

PHP and MySQL Code:

```
<?php
// Make a MySQL Connection
// Construct our join query
$query = "SELECT family.Position, food.Meal ".
 <indent>"FROM family LEFT JOIN food ".
 "ON family.Position = food.Position";</indent>

$result = mysql_query($query) or die(mysql_error());


// Print out the contents of each row into a table
while($row = mysql_fetch_array($result)){
 <indent>echo $row['Position']. " - ". $row['Meal'];
 echo "<br />";</indent>
}
?>
```

Display:

```
Dad - Steak
Dad - Tacos
Mom - Salad
Daughter -
Dog -
```

Success! The LEFT JOIN preserved every family member, including those who don't yet have a favorite meal in the *food* table! Please feel free to play around with LEFT JOIN until you feel like you have a solid grasp of it. This stuff isn't easy!

MySQL Update

Imagine that you have a MySQL table that holds the information of all the employees in your company. One of the columns in this table is called "Seniority" and it holds an integer value of how many months an employee has worked at your company. Unfortunately for you, your job is to update these numbers every month.

You may be thinking that you'll have to open up your MySQL administration tool and edit each entry by hand. That would take hours. On the other hand, you could master MySQL and have an automated script that you run each month to get the job done for you.

In this lesson you will learn how to replace the existing data of a MySQL table with freshly supplied up-to-date data using the UPDATE MySQL query.

MySQL Update Example

Once again we will be working with the data from a previous example. Sandy has just had a birthday and she now 22 years old. Our job now is to update her age using MySQL commands like UPDATE, SET, and WHERE.

- **UPDATE** - Performs an update MySQL query
- **SET** - The new values to be placed into the table follow SET
- **WHERE** - Limits which rows are affected

PHP & MySQL Code:

```
<?php
// Connect to MySQL
// Get Sandy's record from the "example" table
$result = mysql_query("UPDATE example SET age='22' WHERE age='21'")
or die(mysql_error());


$result = mysql_query("SELECT * FROM example WHERE age='22'")
or die(mysql_error());

// get the first (and hopefully only) entry from the result
$row = mysql_fetch_array( $result );
echo $row['name']." - ".$row['age']. "<br />";
?>
```

Display:

```
Sandy Smith - 22
```

Now it is important to note that this query would have updated ALL records that had an age of 21 to the new age of 22. In a table where Sandy is not the onlyl entry, this may become a problem, and a more sophisticated solution would be necessary.

MySQL Delete

Maintenance is a very common task that is necessary for keeping MySQL tables current. From time to time, you may even need to delete items from your database. Some potential reasons for deleting a record from MySQL include when: someone deletes a post from a forum, an employee leaves a company, or you're trying to destroy your records before the federalies come!

MySQL DELETE Example

The DELETE query is very similar to the UPDATE Query in the previous lesson We need to choose a table, tell MySQL to perform the deletion, and provide the requirements that a record must have for it to be deleted.

Say we want to delete the youngest employee from our previously created table because he has to go back to school. This is how we do it.

PHP & MySQL Code:

```
<?php
// Connect to MySQL
// Delete Bobby from the "example" MySQL table
mysql_query("DELETE FROM example WHERE age='15'")
or die(mysql_error());
?>
```

It is important to note that this query would have deleted ALL records that had an age of 15. Since Bobby was the only 15 year old this was not a problem.

MySQL DELETE Tips

Before performing a large delete on a database, be sure to back up the table/database in case your script takes off a little more than desired. Test your delete queries before even thinking about using them on your table. As long as you take caution when using this powerful query you should not run into any problems.

MySQL Database Backups

If you're storing anything in MySQL databases that you do not want to lose, chances are you should be doing weekly or even daily backups. Depending on what you're using your databases for -- be it to store forum messages, employee information, or your spending information -- you are going to need to choose a backup schedule that meets your needs.

You may or may not know that MySQL databases are just files that are stored on your web server. This fact makes the whole backup and restore process extremely simple and painless once you have figured out how to do it.

Different Ways to Get That Backup Done

There are many paths you can take to create a MySQL backup. However, no matter which application, control panel tool, or SSH script you use, all of the backups will fit into two types of backups: a dump or raw backup.

MySQL Dump

A MySQL dump is a bit slower than a raw backup because it creates all the SQL queries required to create the tables of that database, as well as all the insert queries required to place the information back into the database's tables.

If you want to perform the mysql dump manually, without the assistance of your hosts control panel, then run SSH to your web server and do the following (taken from MySql.com):

.:. mysqldump --tab=/path/to/some/dir --opt db_name

If you were to open up a MySQL dump file you would see a slew of SQL queries that you would probably be able to understand (if you've already read through this whole tutorial!).

MySQL Raw Backup

A MySQL Raw Backup is quicker because it does not translate the contents of the database into human readable SQL queries. However, not many control panels support this type of backup, so do not worry if your hosting provider doesn't have this option set up for you.

MySQL Backup in Control Panel cPanel

cPanel is the most widely used web host control panel at this time, so we thought it would make sense to provide a walkthrough specifically for cPanel.

From the application selection screen click "Backup". This will bring you to the backup application that allows you to generate and download complete backups for your site.

To back up a database individually, look for the title "Download a SQL Database Backup" or something similar. Below that title should be a listing of every database that you have created. Simply click on the name of the database you want to backup and save it to your computer.

That's it! Now just be sure that you have a regular backup schedule, just in case the unthinkable

happens and your web host loses all your database information!

MySQL GROUP BY - Aggregate Functions

After you have mastered the basics of MySQL, it's time to take the next step and take on Aggregate Functions. Before we talk about what they are, let's review the definition of aggregate, as it relates to MySQL:

.:. **Aggregate** - Constituting or amounting to a whole; total. ~American Heritage Dictionary

With this type of wording, we can assume that MySQL's aggregate functions are something that will be very top-level, or in other words, the opposite of detailed.

The most common types of aggregate functions let you find out things like the minimum, maximum and even the average of a "grouped" set of data. The trick to understanding aggregate functions is often understanding what kind of data is being grouped and analyzed.

MySQL GROUP BY - The Data

Before we can start throwing around these fancy functions, let's build an appropriate table that has enough data in it to be meaningful to us. Below is the SQL for our "products" table. You can either run this SQL statement in your MySQL administrator software or use MySQL to execute the queries (i.e. create table, then each of the records).

You can download the products.sql file from our website. If you are new to MySQL you will need to know how to Create a MySQL Table and Insert a MySQL Row.

Below is the MySQL table *products*.

id	name	type	price
123451	Park's Great Hits	Music	19.99
123452	Silly Puddy	Toy	3.99
123453	Playstation	Toy	89.95
123454	Men's T-Shirt	Clothing	32.50
123455	Blouse	Clothing	34.97
123456	Electronica 2002	Music	3.99
123457	Country Tunes	Music	21.55
123458	Watermelon	Food	8.73

GROUP BY - Creating Your First "Group"

Imagine that our store was running an advertisement in the newspaper and we wanted to have a "bargain basement" section that listed the lowest price of each product type. In this case we would be "grouping" by the product type and finding the minimum price of each group.

Our query needs to return two columns: product type and minimum price. Additionally, we want to use the *type* column as our group. The SELECT statement we are about to use will look different because it includes an aggregate function, MIN, and the GROUP BY statement, but otherwise it isn't any different than a normal SELECT statement.

PHP and MySQL Code:

```
<?php
// Make a MySQL Connection

$query = "SELECT type, MIN(price) FROM products GROUP BY type";

$result = mysql_query($query) or die(mysql_error());

// Print out result
while($row = mysql_fetch_array($result)){
 echo $row['type']. " - $". $row['MIN(price)'];
 echo "<br />";
}
?>
```

Our "products" table has four types of products: Music, Toy, Clothing and Food. When we GROUP BY *type* then we get one result for each of these types.

Display:

```
Clothing - $32.50
Food - $8.73
Music - $3.99
Toy - $3.99
```

MySQL GROUP BY - Review

Group BY is good for retrieving information about a group of data. If you only had one product of each type, then GROUP BY would not be all that useful.

GROUP BY only shines when you have many similar things. For example, if you have a number of products of the same type, and you want to find out some statistical information like the minimum, maximum, or other top-level info, you would use GROUP BY.

Some technical rules of GROUP BY:

.:. The column that you GROUP BY must also be in your SELECT statement.
.:. Remember to group by the column you want information about and not the one you are applying the aggregate function on. In our above example we wanted information on the *type* column and the aggregate function was applied to the *price* column.

The next few lessons will provide a walkthrough for using other popular MySQL aggregate functions in conjunction with the *GROUP BY* statement.

MySQL Aggregate Functions - COUNT()

This lesson will teach you how to use the aggregate function COUNT(). If you missed the Aggregate Introduction Lesson, please check it out now, as it explains many concepts used in this lesson!

We will be using the "products" table that we constructed to display the use of MySQL's COUNT function.

You can download the products.sql file from our website. If you are new to MySQL you will need to know how to Create a MySQL Table and Insert a MySQL Row first.

Below is the MySQL table "products".

id	name	type	price
123451	Park's Great Hits	Music	19.99
123452	Silly Puddy	Toy	3.99
123453	Playstation	Toy	89.95
123454	Men's T-Shirt	Clothing	32.50
123455	Blouse	Clothing	34.97
123456	Electronica 2002	Music	3.99
123457	Country Tunes	Music	21.55
123458	Watermelon	Food	8.73

MySQL COUNT - Counting Records

The COUNT function is an aggregate function that simply counts all the items that are in a group. The "products" table that is displayed above has several products of various types. One use of COUNT might be to find out how many items of each type there are in the table.

Just as we did in the aggregate introduction lesson, we are going to GROUP BY *type* to create four groups: Music, Toy, Clothing and Food. For a slight change of pace, let's count the *name* column to find how many products there are per type.

PHP and MySQL Code:

```
<?php
// Make a MySQL Connection

$query = "SELECT type, COUNT(name) FROM products GROUP BY type";

$result = mysql_query($query) or die(mysql_error());

// Print out result
while($row = mysql_fetch_array($result)){
 echo "There are ". $row['COUNT(name)'] ." ". $row['type'] ." items.";
 echo "<br />";
}
?>
```

Display:

```
There are 2 Clothing items.
There are 1 Food items.
There are 3 Music items.
There are 2 Toy items.
```

MySQL Aggregate Functions - SUM()

This lesson will teach you how to use the aggregate function SUM(). If you haven't already read through Tizag's Aggregate Introduction Lesson, please check it out now. It explains concepts used in this lesson.

We will be using the "products" table again -- this time to display the use of MySQL's SUM function.

You can download the products.sql file from our website. If you are new to MySQL you will need to know how to Create a MySQL Table and Insert a MySQL Row.

Here's a visual of the "products" table.

id	name	type	price
123451	Park's Great Hits	Music	19.99
123452	Silly Puddy	Toy	3.99
123453	Playstation	Toy	89.95
123454	Men's T-Shirt	Clothing	32.50
123455	Blouse	Clothing	34.97
123456	Electronica 2002	Music	3.99
123457	Country Tunes	Music	21.55
123458	Watermelon	Food	8.73

MySQL SUM - Totaling Groups

SUM is an aggregate function that totals a specific column for a group. The "products" table that is displayed above has several products of various types. One use of SUM might be to find the total of all the items' *price* for each product *type*.

Just as we did in the aggregate introduction lesson, we are going to apply the aggregate function to *price* and GROUP BY *type* to create four groups: Music, Toy, Clothing and Food.

PHP and MySQL Code:

```
<?php
// Make a MySQL Connection

$query = "SELECT type, SUM(price) FROM products GROUP BY type";

$result = mysql_query($query) or die(mysql_error());

// Print out result
while($row = mysql_fetch_array($result)){
 echo "Total ". $row['type']. " = $". $row['SUM(price)'];
 echo "<br />";
}
?>
```

Display:

```
Total Clothing = $67.47
Total Food = $8.73
Total Music = $45.53
Total Toy = $93.94
```

MySQL Aggregate Functions - AVG()

This lesson will teach you how to use the aggregate function AVG(). If you missed the Aggregate Introduction Lesson, please check it out now. It explains the meaning of aggregates and describes the GROUP BY statement.

The table we will be using is "products" and you can download the products.sql file so you can follow along. The table can be entered through you MySQL interface or through PHP.

If you are new to MySQL/PHP programming you will need to know how to Create a MySQL Table and Insert a MySQL Row

id	name	type	price
123451	Park's Great Hits	Music	19.99
123452	Silly Puddy	Toy	3.99
123453	Playstation	Toy	89.95
123454	Men's T-Shirt	Clothing	32.50
123455	Blouse	Clothing	34.97
123456	Electronica 2002	Music	3.99
123457	Country Tunes	Music	21.55
123458	Watermelon	Food	8.73

MySQL Average - Finding a Middle Ground

The AVG function returns the average value for the specified column of a group.

Our imaginary customers have been complaining recently that our prices are too high, so we would like to find out the average *price* of each product *type* to see if this is in fact the truth.

To find out this metric we are going to apply the aggregate function to the *price* and GROUP BY *type* to create four price groups: Music, Toy, Clothing and Food.

PHP and MySQL Code:

```
<?php
// Make a MySQL Connection

$query = "SELECT type, AVG(price) FROM products GROUP BY type";

$result = mysql_query($query) or die(mysql_error());

// Print out result
while($row = mysql_fetch_array($result)){
 echo "The average price of ". $row['type']. " is $".$row['AVG(price)'];
 echo "<br />";
}
?>
```

Display:

```
The average price of Clothing is $33.735000
The average price of Food is $8.730000
The average price of Music is $15.176667
The average price of Toy is $46.970000
```

Those prices seem very reasonable, in my opinion. I think our imaginary customers should change their view and keep buying products from us.

MySQL Aggregate Functions - MIN()

This lesson will teach you how to use the aggregate function MIN(). If you missed the Aggregate Introduction Lesson, you might want to check it out to learn about the GROUP BY statement and its use with MySQL aggregate functions.

You can download the table used in this example, products.sql, from our website. A SQL file can be run through your MySQL administrator interface to create the table.

However, if you would like to create the table with PHP/MySQL, check out our Create a MySQL Table and Insert a MySQL Row lessons.

id	name	type	price
123451	Park's Great Hits	Music	19.99
123452	Silly Puddy	Toy	3.99
123453	Playstation	Toy	89.95
123454	Men's T-Shirt	Clothing	32.50
123455	Blouse	Clothing	34.97
123456	Electronica 2002	Music	3.99
123457	Country Tunes	Music	21.55
123458	Watermelon	Food	8.73

MySQL MIN

The MIN function is an aggregate function that finds the smallest value in a group. The products table that is displayed above has several products of various types. One use of MIN might be to find out the cheapest item in each group.

Just as we did in theAggregate Introduction Lesson, we are going to GROUP BY *type* to create four groups: Music, Toy, Clothing and Food. The column that will have the MIN function applied to it is, of course, *price*.

PHP and MySQL Code:

```
<?php
// Make a MySQL Connection

$query = "SELECT type, MIN(price) FROM products GROUP BY type";

$result = mysql_query($query) or die(mysql_error());

// Print out result
while($row = mysql_fetch_array($result)){
 echo "The cheapest  ". $row['type']. " is $" .$row['MIN(price)'];
 echo "<br />";
}
?>
```

Display:

```
The cheapest Clothing is $32.50
The cheapest Food is $8.73
The cheapest Music is $3.99
The cheapest Toy is $3.99
```

MySQL Aggregate Functions - MAX()

This lesson will teach you how to use the MAX() aggregate function . If you missed the Aggregate Introduction Lesson, please check it out now, as it explains many concepts used in this lesson!

You can download the example the products.sql file from our website, which contains the SQL for the table below.

If you would like to use PHP/MySQL to create the table, then you will need to know how to Create a MySQL Table and Insert a MySQL Row.

Below is the MySQL table "products".

id	name	type	price
123451	Park's Great Hits	Music	19.99
123452	Silly Puddy	Toy	3.99
123453	Playstation	Toy	89.95
123454	Men's T-Shirt	Clothing	32.50
123455	Blouse	Clothing	34.97
123456	Electronica 2002	Music	3.99
123457	Country Tunes	Music	21.55
123458	Watermelon	Food	8.73

MySQL MAX - Finding the Big One

MySQL's MAX aggregate function will find the largest value in a group. The "products" table that is displayed above has several products of various types. We could use the MAX function to find the most expensive item for each *type* of product.

Just as we did in the aggregate introduction lesson, we are going to GROUP BY *type* to create four groups: Music, Toy, Clothing and Food. We will also be applying the aggregate function to the *price* column.

PHP and MySQL Code:

```
<?php
// Make a MySQL Connection

$query = "SELECT type, MAX(price) FROM products GROUP BY type";

$result = mysql_query($query) or die(mysql_error());

// Print out result
while($row = mysql_fetch_array($result)){
 echo "The most expensive  ". $row['type']. " is $" .$row['MAX(price)'];
 echo "<br />";
}
?>
```

Display:

```
The most expensive Clothing is $34.97
The most expensive Food is $8.73
The most expensive Music is $21.55
The most expensive Toy is $89.95
```

MySQL - SQL Injection Prevention

If you have ever taken raw user input and inserted it into a MySQL database there's a chance that you have left yourself wide open for a security issue known as *SQL Injection*. This lesson will teach you how to help prevent this from happening and help you secure your scripts and MySQL statements.

What is SQL Injection

SQL injection refers to the act of someone inserting a MySQL statement to be run on your database without your knowledge. Injection usually occurs when you ask a user for input, like their name, and instead of a name they give you a MySQL statement that you will unknowingly run on your database.

SQL Injection Example

Below is a sample string that has been gathered from a normal user and a bad user trying to use SQL Injection. We asked the users for their login, which will be used to run a SELECT statement to get their information.

MySQL & PHP Code:

```
// a good user's name
$name = "timmy";
$query = "SELECT * FROM customers WHERE username = '$name'";
echo "Normal: " . $query . "<br />";

// user input that uses SQL Injection
$name_bad = "' OR 1'";

// our MySQL query builder, however, not a very safe one
$query_bad = "SELECT * FROM customers WHERE username = '$name_bad'";

// display what the new query will look like, with injection
echo "Injection: " . $query_bad;
```

Display:

```
Normal: SELECT * FROM customers WHERE username = 'timmy'
Injection: SELECT * FROM customers WHERE username = '' OR 1''
```

The normal query is no problem, as our MySQL statement will just select everything from customers that has a username equal to *timmy*.

However, the injection attack has actually made our query behave differently than we intended. By using a single quote (') they have ended the string part of our MySQL query

∴ username = ''

and then added on to our WHERE statement with an OR clause of 1 (always true).

.:. username = ' ' **OR 1**

This OR clause of 1 will always be *true* and so **every single entry** in the "customers" table would be selected by this statement!

More Serious SQL Injection Attacks

Although the above example displayed a situation where an attacker could possibly get access to a lot of information they shouldn't have, the attacks can be a lot worse. For example an attacker could empty out a table by executing a *DELETE* statement.

MySQL & PHP Code:

```
$name_evil = "'; DELETE FROM customers WHERE 1 or username = '";

// our MySQL query builder really should check for injection
$query_evil = "SELECT * FROM customers WHERE username = '$name_evil'";

// the new evil injection query would include a DELETE statement
echo "Injection: " . $query_evil;
```

Display:

```
SELECT * FROM customers WHERE username = ' '; DELETE FROM customers WHERE 1 or
username = ' '
```

If you were run this query, then the injected DELETE statement would completely empty your "customers" table. Now that you know this is a problem, how can you prevent it?

Injection Prevention - mysql_real_escape_string()

Lucky for you, this problem has been known for a while and PHP has a specially-made function to prevent these attacks. All you need to do is use the mouthful of a function *mysql_real_escape_string*.

What *mysql_real_escape_string* does is take a string that is going to be used in a MySQL query and return the same string with all SQL Injection attempts safely escaped. Basically, it will replace those troublesome quotes(') a user might enter with a MySQL-safe substitute, an escaped quote \'.

Lets try out this function on our two previous injection attacks and see how it works.

MySQL & PHP Code:

```
//NOTE: you must be connected to the database to use this function!
// connect to MySQL

$name_bad = "' OR 1'";

$name_bad = mysql_real_escape_string($name_bad);

$query_bad = "SELECT * FROM customers WHERE username = '$name_bad'";
echo "Escaped Bad Injection: <br />" . $query_bad . "<br />";


$name_evil = "'; DELETE FROM customers WHERE 1 or username = '";

$name_evil = mysql_real_escape_string($name_evil);

$query_evil = "SELECT * FROM customers WHERE username = '$name_evil'";
echo "Escaped Evil Injection: <br />" . $query_evil;
```

Display:

```
Escaped Bad Injection:
SELECT * FROM customers WHERE username = '\' OR 1\''
Escaped Evil Injection:
SELECT * FROM customers WHERE username = '\'; DELETE FROM customers WHERE 1 or
username = \''
```

Notice that those evil quotes have been escaped with a backslash \, preventing the injection attack. Now all these queries will do is try to find a username that is just completely ridiculous:

- **.:.** Bad: \' OR 1\'
- **.:.** Evil: \'; DELETE FROM customers WHERE 1 or username = \'

And I don't think we have to worry about those silly usernames getting access to our MySQL database. So please do use the handy *mysql_real_escape_string()* function to help prevent SQL Injection attacks on your websites. You have no excuse not to use it after reading this lesson!

MySQL Date - Formats

MySQL comes with several data types for storing a date in its database system: DATETIME, DATE, TIMESTAMP, and YEAR. This lesson will show you the proper formats of each type, show their related MySQL functions, and give an INSERT exampleof each.

These date types are chosen for a column when you create a new table in MySQL. Often the most difficult part of using dates in MySQL is to be sure the format of the date you are trying to store matches the format of your table's date column. If you haven't already, try to create a new MySQL table with the date types we mentioned above.

We have assembled a "date playground" MySQL table that can be used to follow along with this lesson. dateplayground.sql. Also, the following acronyms are used in this lesson:

- Y - year segment
- M - month segment
- D - day segment
- H - hour segment
- m - minute segment, note the lower case
- S - sec segment

MySQL Date - DATE

The default way to store a date in MySQL is with the type DATE. Below is the proper format of a DATE.

- **YYYY-MM-DD**
- **Date Range**: 1000-01-01 to 9999-12-31

If you try to enter a date in a format other than the Year-Month-Day format then it might work, but it won't be storing them as you expect.

To insert the current date into your table you can use MySQL's built-in function CURDATE() in your query. Below we have created 2 dates, one manually and one using CURDATE().

PHP & MySQL Code:

```
<?php
//This assumes you have already created the 'dateplayground' table
//Connect to DB
$query_manual = "INSERT INTO dateplayground (dp_name, dp_date)
 VALUES ('DATE: Manual Date', '2020-2-14')";
$query_auto = "INSERT INTO dateplayground (dp_name, dp_date)
 VALUE ('DATE: Auto CURDATE()', CURDATE() )";

mysql_query($query_manual) or die(mysql_error());
mysql_query($query_auto) or die(mysql_error());
?>
```

MySQL Date - YEAR

If you just need to store the year of an event, MySQL also has a date type just for that. YEAR's

format is simply:

- **YYYY**
- **Date Range**: 1901 to 2155

It should be noted that the range of years that can be stored are from 1901 to 2155. If you need to store years outside that range then use DATE instead of YEAR.

Below we have created another manual and automatic example to show off YEAR's use. We have used CURDATE() again, even though it provides a lot more information than YEAR requires. All the date information, besides the year, is just ignored by YEAR.

PHP & MySQL Code:

```
<?php
$query_manual = "INSERT INTO dateplayground (dp_name, dp_year)
 VALUES ('YEAR: Manual Year', '2011')";
$query_auto = "INSERT INTO dateplayground (dp_name, dp_year)
 VALUE ('YEAR: Auto CURDATE()', CURDATE() )";

mysql_query($query_manual) or die(mysql_error());
mysql_query($query_auto) or die(mysql_error());
?>
```

MySQL Date - DATETIME

DATETIME actually stores both the current date and time, meaning it has the ability to store the year, month, day, hour, minute, and second inside it. DATETIME's format is:

- **YYYY-MM-DD HH:mm:SS**
- **Date Range**: 1000-01-01 00:00:00 to 9999-12-31 23:59:59

The hyphen and the colon are the standard character to separate a date and time respectively, but MySQL allows for you to choose your own delimiters if you wish.

With DATETIME you can choose to store the date or the time and date together, **but** you cannot store just the time.

In our example below we have manually stored a complete DATETIME and also used three different MySQL functions: CURDATE(), CURTIME(), and NOW().

PHP & MySQL Code:

```
<?php
$query_manual = "INSERT INTO dateplayground (dp_name, dp_datetime)
 VALUES ('DATETIME: Manual DateTime', '1776-7-4 04:13:54')";
$query_autodate = "INSERT INTO dateplayground (dp_name, dp_datetime)
 VALUE ('DATETIME: Auto CURDATE()', CURDATE() )";
$query_autotime = "INSERT INTO dateplayground (dp_name, dp_datetime)
 VALUE ('DATETIME: Auto CURTIME()', CURTIME() )";  //This will fail
$query_autonow = "INSERT INTO dateplayground (dp_name, dp_datetime)
 VALUE ('DATETIME: Auto NOW()', NOW() )";

mysql_query($query_manual) or die(mysql_error());
mysql_query($query_autodate) or die(mysql_error());
mysql_query($query_autotime) or die(mysql_error());
mysql_query($query_autonow) or die(mysql_error());
?>
```

MySQL Date - TIMESTAMP

TIMESTAMP is a format popularized by the *NIX operating systems that stores the amount of time that has passed since January 1, 1970. If you want more versatility than this date type provides, remember to try DATETIME.

- **.:. YYYY-MM-DD HH:mm:SS**
- **.:. Date Range**: 1970-01-01 00:00:00 to 2037-12-31 23:59:59

The big difference between DATETIME and TIMESTAMP is the date ranges that can be stored. Below we have purposely entered an erroneous date, manually, so you can see what happens when you enter a date that is outside the boundaries of a this type.

PHP & MySQL Code:

```
<?php
//This will fail
$query_manual = "INSERT INTO dateplayground (dp_name, dp_timestamp)
 VALUES ('TIMESTAMP: Manual Timestamp', '1776-7-4 04:13:54')";
$query_autodate = "INSERT INTO dateplayground (dp_name, dp_timestamp)
 VALUE ('TIMESTAMP: Auto CURDATE()', CURDATE() )";
//This will fail
$query_autotime = "INSERT INTO dateplayground (dp_name, dp_timestamp)
 VALUE ('TIMESTAMP: Auto CURTIME()', CURTIME() )";
$query_autonow = "INSERT INTO dateplayground (dp_name, dp_timestamp)
 VALUE ('TIMESTAMP: Auto NOW()', NOW() )";

mysql_query($query_manual) or die(mysql_error());
mysql_query($query_autodate) or die(mysql_error());
mysql_query($query_autotime) or die(mysql_error());
mysql_query($query_autonow) or die(mysql_error());
?>
```

Viewing dateplayground in PHP

Below is a quick script that will spit out the MySQL table dateplayground in HTML.

PHP & MySQL Code:

```
<?php
$query = "SELECT * FROM dateplayground";
$result = mysql_query($query) or die(mysql_error());

echo "<table border='1'><tr>";
for($i = 0; $i < mysql_num_fields($result); $i++){
 echo "<th>".mysql_field_name($result, $i)."</th>";
}
echo "</tr>";
while($row = mysql_fetch_array($result)){
 echo "<tr>";
 for($i = 0; $i < mysql_num_fields($result); $i++){
  echo "<td>". $row[$i] ."</td>";
 }
 echo "</tr>";
}

echo "</table>";

?>
```

dp_name	dp_year	dp_date	dp_datetime	dp_timestamp
DATE: Manual Date	0000	2020-02-14	0000-00-00 00:00:00	0000-00-00 00:00:00
DATE: Auto CURDATE()	0000	2006-09-19	0000-00-00 00:00:00	0000-00-00 00:00:00
YEAR: Manual Year	2011	0000-00-00	0000-00-00 00:00:00	0000-00-00 00:00:00
YEAR: Auto CURDATE()	2006	0000-00-00	0000-00-00 00:00:00	0000-00-00 00:00:00
DATETIME: Manual DateTime	0000	0000-00-00	1776-07-04 04:13:54	0000-00-00 00:00:00
DATETIME: Auto CURDATE()	0000	0000-00-00	2006-09-19 00:00:00	0000-00-00 00:00:00
DATETIME: Auto CURTIME()	0000	0000-00-00	0000-00-00 00:00:00	0000-00-00 00:00:00
DATETIME: Auto NOW()	0000	0000-00-00	2006-09-19 16:56:56	0000-00-00 00:00:00
TIMESTAMP: Manual Timestamp	0000	0000-00-00	0000-00-00 00:00:00	0000-00-00 00:00:00
TIMESTAMP: Auto CURDATE()	0000	0000-00-00	0000-00-00 00:00:00	2006-09-19 00:00:00
TIMESTAMP: Auto CURTIME()	0000	0000-00-00	0000-00-00 00:00:00	0000-00-00 00:00:00
TIMESTAMP: Auto NOW()	0000	0000-00-00	0000-00-00 00:00:00	2006-09-19 16:56:56

Notice that the rows DATETIME: Auto CURTIME(), TIMESTAMP: Manual Timestamp, and TIMESTAMP: Auto CURTIME() have all zeros. This is because they were the INSERTs that were erroneous. When you enter dates that are out of the range or in the wrong format for a given date type, MySQL will often just enter in the default value of all zeros.

MySQL Time - Formats

There are three different types of time data types in MySQL: TIME, DATETIME, and TIMESTAMP. If you would like to learn more about DATETIME and TIMESTAMP, then check out our MySQL Date section, as we've covered them there. This lesson will just be covering the basics of using TIME.

MySQL Time - TIME

First you need to create a MySQL table with a TIME type. We have one already created if you want to use it: timeplayground.sql.

The TIME data type can be used to store actual times as well as the amount of time between two points in time (like the time between now and the weekend) that may sometimes be larger than 23 hours. H - Hour; M - Minute; S - Second.

- **Standard format**: HH:MM:SS
- **Extended hour format**: HHH:MM:SS
- **Time Range**: -838:59:50 to 838:59:59

When manually entering a time into MySQL it is highly recommended that you use the exact format show above. MySQL allows for many different ways to enter a time, but they don't always behave as you would expect. Using the standard/extended format we have shown above will help you avoid annoying problems.

Below we have entered 3 manual times into MySQL. The first is done in the recommended format, the second is a shorthand version of the first and the final example is outside the allowed time range.

PHP & MySQL Code:

```
<?php
//This assumes you have already created the 'dateplayground' table
//Connect to DB
$query_manual1 = "INSERT INTO timeplayground (dp_name, dp_time)
 VALUES ('TIME: Manual Time', '12:10:00')"; //perfectly done
$query_manual2 = "INSERT INTO timeplayground (dp_name, dp_time)
 VALUES ('TIME: Manual Time', '1210')"; // will this shorthand work?
$query_manual3 = "INSERT INTO timeplayground (dp_name, dp_time)
 VALUES ('TIME: Manual Time', '978:31:12')"; //how about this?

mysql_query($query_manual1) or die(mysql_error());
mysql_query($query_manual2) or die(mysql_error());
mysql_query($query_manual3) or die(mysql_error());
?>
```

MySQL Time - NOW()

To get the current time, use MySQL's built in function NOW(). NOW() contains both the date and time information, but MySQL is smart enough to just use the data needed for TIME.

PHP & MySQL Code:

```
<?php
$query_auto = "INSERT INTO timeplayground (dp_name, dp_time)
 VALUE ('TIME: Auto NOW()', NOW() )";

mysql_query($query_auto) or die(mysql_error());
?>
```

MySQL timeplayground.sql Displayed

Below is a small PHP script to spit out a rough version of our timeplayground.sql table.

PHP & MySQL Code:

```
<?php
$query = "SELECT * FROM timeplayground";
$result = mysql_query($query) or die(mysql_error());

echo "<table border='1'><tr>";
for($i = 0; $i < mysql_num_fields($result); $i++){
 echo "<th>".mysql_field_name($result, $i)."</th>";
}
echo "</tr>";
while($row = mysql_fetch_array($result)){
 echo "<tr>";
 for($i = 0; $i < mysql_num_fields($result); $i++){
  echo "<td>". $row[$i] ."</td>";
 }
 echo "</tr>";
}

echo "</table>";
?>
```

dp_name	dp_time
TIME: Manual Time	12:10:00
TIME: Manual Time	00:12:10
TIME: Manual Time	838:59:59
TIME: Auto NOW()	14:30:36

Our first manual time was handled just fine, but our second one did not. MySQL interpreted 1210 as MM:SS instead of HH:MM as we assumed. This is why it's best to use the formats we've described at the beginning.

The third manual entry was changed from 978:31:12 to 838:59:59, so that it would be within TIME's range.

MySQL Index - Overclock Your Tables

Hardware enthusiasts have been overclocking their PCs for years now, trying to push the limits of their hardware for maximum performance. Sometimes they are successful and their applications run speedy fast, while other times they push a little too hard and end up damaging the computer!

Although it isn't quite as extreme, **indexes** in MySQL can increase the speed of your MySQL queries to squeeze a bit more performance out of your database.

MySQL Index - Speed and Extra Overhead

Indexes are created on a per column basis. If you have a table with the columns: name, age, birthday and employeeID and want to create an index to speed up how long it takes to find *employeeID* values in your queries, then you would need to create an index for *employeeID*. When you create this index, MySQL will build a lookup index where *employeeID* specific queries can be run quickly. However, the name, age and birthday queries would not be any faster.

Indexes are something extra that you can enable on your MySQL tables to increase performance,cbut they do have some downsides. When you create a new index MySQL builds a separate block of information that needs to be updated every time there are changes made to the table. This means that if you are constantly updating, inserting and removing entries in your table this could have a negative impact on performance.

Creating a MySQL Index - New Table

If you are creating a new MySQL table you can specify a column to index by using the INDEX term as we have below. We have created two fields: name and employeeID (index).

MySQL Code:

```
CREATE TABLE employee_records (
 name VARCHAR(50),
 employeeID INT, INDEX (employeeID)
)
```

Creating a MySQL Index - Existing Table

You can also add an index to an older table that you think would benefit from some indexing. The syntax is very similar to creating an index in a new table. First, let's create the table.

MySQL Code:

```
CREATE TABLE employee_records2 (name VARCHAR(50), employeeID INT)
```

With our newly created table we are going to update the "employee_records2" table to include an index.

MySQL Code:

```
CREATE INDEX id_index ON employee_records2(employeeID)
```

We keep our existing *employeeID* field and create a new index *id_index* that is made up of *employeeID* data.