PHP/SQL 1: Introduction to Database Programming

- **Lesson 1: Introduction**
  - Understanding the Learning Sandbox
  - Experimenting with Examples
  - MySQL Syntax
  - Databases and Permissions
    - Creating Databases
    - The Command Line

- **Lesson 2: Creating Tables**
  - Reconnecting to MySQL
  - Creating and Checking Tables
    - Creating a Table
    - Checking Tables
  - Inserting Values into Tables
    - Inserting Specific Column Values
  - Retrieving Data from Tables

- **Lesson 3: Searching Through Tables**
  - Exploring the 'where' Keyword
  - The 'like' Selector
  - Updating and Deleting Data
    - Updating Entries
    - Deleting Entries

- **Lesson 4: Managing Tables**
  - Altering Tables
    - Deleting and Reordering Columns
    - Changing Column Type
  - Renaming Tables
  - Deleting Tables

- **Lesson 5: Managing Table Entries**
  - Optimizing with Keys/Indexes
  - Auto_increment and Primary Keys
    - The Primary Key

- **Lesson 6: Relational Databasing**
  - What is a Relational Database?
  - Inner Joins
  - Outer Joins
  - Aliases

- **Lesson 7: Managing Query Results**
  - Group By
  - Using Distinct to Prevent Duplicate Results
  - Searching and Counting within Groups
    - Searching Within Groups
  - Renaming Results
  - Ordering Results
    - Descending Results
    - Nesting Groups and Orders
  - Limiting Results

- **Lesson 8: Database Programming with PHP**
  - The PHP/MySQL Relationship
  - Connecting to a Database in PHP
  - Executing SQL Commands
- **Delimiting Queries**
- **Lesson 9:** *Project Address/Phone Book, Part 1*
  - Project Description
  - Table Layout
  - Table Creation
- **Lesson 10:** *Project Address/Phone Book, Part 2*
  - Starting with the HTML Form
  - Storing the Data into SQL through PHP
  - Breaking Down addentry.php
- **Lesson 11:** *Project Address/Phone Book, Part 3*
  - Construction of viewbook.php
- **Lesson 12:** *Advanced SQL Syntax, Part 1*
  - Date and Time Functions
    - `DAYOFWEEK(date)`
    - `WEEKDAY(date)`
    - `DAYOFMONTH(date)`
    - `DAYOFYEAR(date)`
    - `YEAR(date)`
    - `MONTH(date)`
    - `DOWNAME(date)`
    - `MONTHNAME(date)`
    - `QUARTER(date)`
    - `WEEK(date)`
    - `WEEK(date, weekday)`
    - `TO_DAYS(date)`
    - `FROM_DAYS(x)`
    - `CURDATE(), CURRENT DATE`
    - `PERIOD_ADD(period, x)`
    - `PERIOD_DIFF(period1, period2)`
    - `DATEDIFF(date1, date2)`
    - `DATE_FORMAT(date, format)`
  - Date/Time Mathematical Functions
    - `DATE_ADD(date/time, INTERVAL expr type), ADDDATE(date/time, INTERVAL expr type)`
    - `DATE_SUB(date/time, INTERVAL expr type), SUBDATE(date/time, INTERVAL expr type)`
    - `EXTRACT(expr type FROM date/time)`
    - `CURRENT_TIME()`
    - `HOUR(time)`
    - `MINUTE(time)`
    - `SECOND(time)`
    - `TIME_TO_SEC(time)`
    - `SEC_TO_TIME(sec)`
    - `TIME_FORMAT(time, format)`
    - `NOW(), SYSDATE(), CURRENT_TIMESTAMP()`
    - `UNIX_TIMESTAMP(), UNIX_TIMESTAMP(date)`
    - `FROM_UNIXTIME(unix_timestamp), FROM_UNIXTIME(unix_timestamp, format)`
- **Lesson 13:** *Advanced SQL Syntax, Part 2*
  - Math Functions
    - `MOD(x, y)`
    - `ABS(x)`
    - `SIGN(x)`
    - `LEAST(x, y, ...)`
Lesson 14: Advanced SQL Syntax, Part 3

- **String and Character Functions**
  - ASCII(str)
  - CONV(x,y,z)
  - BIN(x)
  - OCT(x)
  - HEX(x)
  - CHAR(x,y,z,...)
  - LENGTH(str), CHAR_LENGTH(str), OCTET_LENGTH(str), CHARACTER_LENGTH(str)
  - LOCATE(str1,str2), POSITION(str1 IN str2)
  - LOCATE(str1,str2,x)
  - INSTR(str1, str2)
  - SOUNDEX(str)
  - CONCAT (str1,str2,...)
  - LPAD(str1,x,str2)
  - RPAD(str1,x,str2)
  - LEFT(str,x)
  - RIGHT(str,x)
  - LTRIM(str)
  - RTRIM(str)
  - LOAD_FILE(filen)
  - SPACE(x)
  - REPLACE(str1,str2,str3)
  - REPEAT (str,x)
  - REVERSE(str)
  - INSERT(str1,x,y,str2)
  - ELT(x,str1,str2,str3,...)
  - FIELD(str1,str2,str3,str4,...)
  - TRIM([BOTH | LEADING | TRAILING] [str2] FROM) str1
  - SUBSTRING (str,x), SUBSTRING(str FROM x)
  - SUBSTRING(str,x,y), SUBSTRING(str FROM x FOR y), MID(str,x,y)
- `SUBSTRING_INDEX(str1,str2,x)`
- `LCASE(str), LOWER(str)`
- `UCASE(str), UPPER(str)`
- `MAKE_SET(bit,str1,str2,...)`
- `EXPORT_SET(bit,str1,str2,[str3,[x]])`
- `FIND_IN_SET(str1,strlist)`

- Lesson 15: **Final Project**
  - Final Project, Part 1
  - Final Project, Part 2

Copyright © 1998-2014 O'Reilly Media, Inc.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License. See [http://creativecommons.org/licenses/by-sa/3.0/legalcode](http://creativecommons.org/licenses/by-sa/3.0/legalcode) for more information.
Welcome to the O'Reilly School of Technology's PHP/SQL 1: Introduction to Database Programming course!

Course Objectives

When you complete this course, you will be able to:

- create, search, and manage SQL tables and entries.
- perform basic SQL queries and joins, and manage the results.
- demonstrate understanding of advanced SQL syntax for date, time, math, and string functions.
- implement SQL database programming into a PHP-based web application.
- build a full-fledged online address/phone book using PHP and MySQL.

SQL (Structured Query Language) is the language for databases such as Oracle, MySQL, MS Access, etc. Knowing SQL is paramount to using these databases. In this course, you will learn basic SQL database creation and manipulation, as well as how to search databases and how to incorporate them into PHP-based programs and applications.

From beginning to end, you will learn by doing your own SQL-based projects using PHP, and then handing them in for instructor feedback. These projects, as well as the final project, will add to your portfolio and will contribute to certificate completion. Besides a browser and internet connection, all software is provided online by the O'Reilly School of Technology.

Learning with O'Reilly School of Technology Courses

As with every O'Reilly School of Technology course, we'll take a user-active approach to learning. This means that you (the user) will be active! You'll learn by doing, building live programs, testing them and experimenting with them—hands-on!

To learn a new skill or technology, you have to experiment. The more you experiment, the more you learn. Our system is designed to maximize experimentation and help you learn to learn a new skill.

We'll program as much as possible to be sure that the principles sink in and stay with you.

Each time we discuss a new concept, you'll put it into code and see what YOU can do with it. On occasion we'll even give you code that doesn't work, so you can see common mistakes and how to recover from them. Making mistakes is actually another good way to learn.

Above all, we want to help you to learn to learn. We give you the tools to take control of your own learning experience.

When you complete an OST course, you know the subject matter, and you know how to expand your knowledge, so you can handle changes like software and operating system updates.

Here are some tips for using O'Reilly School of Technology courses effectively:

- **Type the code.** Resist the temptation to cut and paste the example code we give you. Typing the code actually gives you a feel for the programming task. Then play around with the examples to find out what else you can make them do, and to check your understanding. It's highly unlikely you'll break anything by experimentation. If you do break something, that's an indication to us that we need to improve our system!

- **Take your time.** Learning takes time. Rushing can have negative effects on your progress. Slow down and let your brain absorb the new information thoroughly. Taking your time helps to maintain a relaxed, positive approach. It also gives you the chance to try new things and learn more than you otherwise would if you blew through all of the coursework too quickly.

- **Experiment.** Wander from the path often and explore the possibilities. We can't anticipate all of your questions and ideas, so it's up to you to experiment and create on your own. Your instructor will help if you go completely off the rails.

- **Accept guidance, but don't depend on it.** Try to solve problems on your own. Going from misunderstanding to understanding is the best way to acquire a new skill. Part of what you're learning is problem solving. Of course, you can always contact your instructor for hints when you need them.

- **Use all available resources!** In real-life problem-solving, you aren't bound by false limitations; in OST courses, you are free to use any resources at your disposal to solve problems you encounter: the Internet, reference books, and online help are all fair game.

- **Have fun!** Relax, keep practicing, and don't be afraid to make mistakes! Your instructor will keep you at it until you've mastered the skill. We want you to get that satisfied, "I'm so cool! I did it!" feeling. And you'll have some projects to show off when you're done.
Lesson Format

We'll try out lots of examples in each lesson. We'll have you write code, look at code, and edit existing code. The code will be presented in boxes that will indicate what needs to be done to the code inside.

Whenever you see white boxes like the one below, you'll type the contents into the editor window to try the example yourself. The CODE TO TYPE bar on top of the white box contains directions for you to follow:

<table>
<thead>
<tr>
<th>CODE TO TYPE:</th>
</tr>
</thead>
<tbody>
<tr>
<td>White boxes like this contain code for you to try out (type into a file to run).</td>
</tr>
<tr>
<td>If you have already written some of the code, new code for you to add looks like this.</td>
</tr>
<tr>
<td>If we want you to remove existing code, the code to remove will look like this.</td>
</tr>
<tr>
<td>We may also include instructive comments that you don’t need to type.</td>
</tr>
</tbody>
</table>

We may run programs and do some other activities in a terminal session in the operating system or other command-line environment. These will be shown like this:

<table>
<thead>
<tr>
<th>INTERACTIVE SESSION:</th>
</tr>
</thead>
<tbody>
<tr>
<td>The plain black text that we present in these INTERACTIVE boxes is provided by the system (not for you to type). The commands we want you to type look like this.</td>
</tr>
</tbody>
</table>

Code and information presented in a gray OBSERVE box is for you to inspect and absorb. This information is often color-coded, and followed by text explaining the code in detail:

<table>
<thead>
<tr>
<th>OBSERVE:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gray &quot;Observe&quot; boxes like this contain information (usually code specifics) for you to observe.</td>
</tr>
</tbody>
</table>

The paragraph(s) that follow may provide addition details on information that was highlighted in the Observe box.

We'll also set especially pertinent information apart in "Note" boxes:

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notes provide information that is useful, but not absolutely necessary for performing the tasks at hand.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tip</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tips provide information that might help make the tools easier for you to use, such as shortcut keys.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WARNING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warnings provide information that can help prevent program crashes and data loss.</td>
</tr>
</tbody>
</table>

The CodeRunner Screen

This course is presented in CodeRunner, OST's self-contained environment. We'll discuss the details later, but here's a quick overview of the various areas of the screen:
These videos explain how to use CodeRunner:

File Management Demo

Code Editor Demo

Coursework Demo

Understanding the Learning Sandbox

CodeRunner is a multi-purpose editor that allows you to create applications in many technologies. One of these technologies is MySQL.

Experimenting with Examples

All of the communication tools and learning content will be in the upper part of the screen. We'll ask you frequently to type code into the MySQL shell that we've set up for you, and to experiment by making your own changes. To access your MySQL shell, click the New Terminal button to connect to Unix and log in from there, or click the Connect to MySQL button to connect directly to the MySQL shell. For this course, you can use either method to get to your MySQL shell, although you should be familiar with both methods.

Click the New Terminal button to connect to Unix now. You will see prompts for the login and password you were given when you registered with OST, although the system may fill one or both of these in for you. If the system doesn't automatically log you in, type your login and password when prompted. (You may need to click inside the Unix Terminal window to be able to type. When typing your password, you will not see any characters reflected back to you as you type.) You will then be logged in to one of the OST servers.

You should see something like this:

```
Login: certjosh
Password:
Last login: Thu Aug  5 20:47:25 from 63.171.219.116
cold:~$
```

Then, to start your MySQL shell, type the following at the Unix prompt. (Don’t worry if you’re not familiar with Linux, UNIX, or SSH.)
Type the following command at your Unix prompt:

cold:~$ mysql -h sql.oreillyschool.com -p -u username username

Note  Make sure that you type your username twice, just as it appears above! Of course you should use your actual Sandbox login, which is listed for you on your Student Start Page.

You're prompted for your password. Enter it and you're set to start MySQL. Now you should see a MySQL prompt that looks like this:

OBSERVE:

Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 16168
Server version: 5.0.62-log Source distribution
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>

Since your SQL account is stored on our servers, you can access it from anywhere, through any SSH program you may have - not just the one we provided for you in CodeRunner. If you like, just SSH into your domain (i.e. yourdomain.oreillystudent.com) that we set up for you (ask your instructor if you’re not sure what yours is), then follow the steps above to log in to your MySQL shell.

MySQL Syntax

Now that we're in an interface environment, let's dive into MySQL and start learning a few keywords and syntax. First up is the select keyword. You use select to retrieve data from your database, query information (such as the date in the example below), inquire about the MySQL version, and view other database info. Try this:

Type the following command at your MySQL prompt:

mysql> select current_date(), current_time();

+----------------+----------------+
| current_date() | current_time() |
+----------------+----------------+
| 2010-09-21     | 15:53:59       |
+----------------+----------------+
1 row in set (0.00 sec)

You may have noticed that MySQL tried to put the information you requested into a nice, neat little table. Well, as nice and neat as tables get when they're built out of plus and minus signs, anyway.

See how fast the retrieval time was? 0.00 seconds! With a larger database or a more complicated query, you’d have a slightly longer response time.

In MySQL, commands such as select are case insensitive.

Note  This applies to keywords only. The following lines will all return the same result:

Type the following command at your MySQL prompt:

mysql> SELECT CURRENT_TIME(), CURRENT_DATE();
mysql> select current_time(), current_date();
mysql> SeLeCt CuRrEnT_tImE(), cUrReNt_DaTe();
Did all three lines return identical results? If not, spell check your command lines and make sure the only discrepancies are in the cases (upper or lower) of the letters. Now type in the following command and hit `enter`:

**Type the following command at your MySQL prompt:**

```
mysql> select current_time()
```

What happened? Nothing showed up except for this weird -> prompt.

Most MySQL keywords require a semicolon (;) at the end in order for the command to be executed. If MySQL doesn’t find a semicolon, it’ll simply move on to the next line with a new prompt ->. The MySQL program will repeat this until it finds the next semicolon entered. This feature is useful when your command line is really, really long. It allows you to break it up over several different prompt lines. Programmers in Java, C/C++, PERL, and other languages should have no problem understanding this concept, since it holds true in those languages as well. So how do you execute that command you entered on the last line? You can either enter the semicolon and press Enter to execute your command and retrieve the date, or you can enter `c` at the prompt to cancel the previous command and start with a fresh prompt.

To retrieve the version of MySQL you’re using, simply use the `version()` function. Let’s find out about your MySQL version by trying this:

**Type the following command at your MySQL prompt:**

```
mysql> select current_date(), version();
```

+----------------+------------+
| current_date() | version()  |
+----------------+------------+
| 2010-09-21     | 5.0.62-log |
+----------------+------------+
1 row in set (0.00 sec)

To speed up your MySQL experience, here’s a quick tip you may have figured out already. You can use the Up arrow key to recall the previous command lines you’ve typed. The more times you press the up key, the deeper into history of command lines it goes.

You can use semicolons to combine multiple command lines together. For example:

**Type the following command at your MySQL prompt:**

```
mysql> select current_time(); select version();
```

+----------------+
| current_time() |
+----------------+
| 15:55:41       |
+----------------+
1 row in set (0.00 sec)

+------------+
| version()  |
+------------+
| 5.0.62-log |
+------------+
1 row in set (0.00 sec)

---

**Databases and Permissions**

When working with SQL, you’re always operating within a database environment. In your case, at OST, the name of your database will be identical to your login. Let’s go ahead and pull it up.
Type the following command at your MySQL prompt:

```
mysql> show databases;
```

Do you see a list of databases? These are the names of the databases on the OST server to which you have access. Do you see your database? Let's access it.

Type the following blue and green text at your MySQL prompt:

```
mysql> use username;
Database changed
mysql>
```

What would happen if you tried to access a database to which you didn't have permission? Try it:

Type the following blue and green text at your MySQL prompt:

```
mysql> use zyla;
ERROR 1044:  Access denied for user: 'certjosh@cold.local.oreillyschool..com' to database 'zyla'
mysql>
```

Turns out you can't access any of them except yours—by the same token, nobody but you can look at your database.

Why is this? When we created a database for you on our OST servers, we were able to grant and deny privileges on your database, so that only your login (and your instructor's) could access it. Permissions and privileges are covered in more depth in the PHP/SQL 2 course.

**Creating Databases**

Who could have guessed that in order to show a list of databases, you had to type `show databases`? Can you predict which command we use to `create` databases? Well, if you said `create`, you're right. To create a database we do the following:

Type the following command at your MySQL prompt:

```
mysql> create database yourdatabasename;
```

Did you get an error message? Just like before, that's because on the OST servers, you only have access to your own database; you don't have permission to create new ones on our servers. But, on your own SQL server (for instance, in PHP/SQL 2) you would be able to use the `create` command to build new databases.

When you're finished with your MySQL session, type in `quit` to exit out of the MySQL session. This is one of the few keywords that don't require a semicolon. You should see something like this:

```
INTERACTIVE SESSION:
mysql> quit
Bye
cold:~$
```

You can also use the keyword `exit`.

**The Command Line**
Since you’ve exited the MySQL shell, it’s a good time to revisit the command you typed to enter MySQL in the first place:

```
cold:~$ mysql -h sql.oreillyschool.com -p -u username
```

Most of this course will be spent within the MySQL shell itself; however, because the command line is used often in advanced SQL topics, it’s good to know a bit about that as well.

The command line is a way to enter commands into a computer through the keyboard. The command line is used mostly in open source operating systems like Linux or UNIX, but you can find ways to open a command line terminal in Windows, Mac, or any other operating system. mysql is the command name used to launch the MySQL program that's been installed by the system administrator. Everything written after that command are parameters that are passed into the command to give the application instructions to execute upon launch.

- The `-h` flag tells MySQL to read all the databases off a separate server—in your case, instead of `cold.oreillyschool.com`, you use the OST database server, `sql.oreillyschool.com`, because that's where your database resides.
- The `-p` flag tells mysql to prompt for a password, so you can log in safely.
- The `-u` flag tells mysql to log in the user specified by `username`.
- The last `username` parameter specifies the default database to use—in your case, we created your database name to be the same as your Sandbox login. In the later course, you might create a database with any name you wish.

Just for kicks, let's try to access that pesky 'zyla' database again, using the command line:

```
cold:~$ mysql -h sql.oreillyschool.com -p -u username zyla
Password:
ERROR 1044:  Access denied for user: 'certjosh@cold.local.oreillyschool.com' to database 'zyla'
mysql>
```

Even though it still asked for your password, MySQL did not allow you to log in, because you attempted to default to a database you couldn’t access.

We’ve covered a lot of ground already, but we’ve barely started the fun. See you in the next lesson!

Copyright © 1998-2014 O'Reilly Media, Inc.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License. See http://creativecommons.org/licenses/by-sa/3.0/legalcode for more information.
Creating Tables

Reconnecting to MySQL

Welcome to the second lesson. Do you remember how to log into your MySQL shell? Just to refresh your memory, click the New Terminal button to open a new terminal session and type in your Sandbox username and password to log in. Once you’re in, type the following to launch MySQL:

```
cold:~$ mysql -h sql.useractive.com -p -u username username
Password:
```

Welcome to the MySQL monitor. Commands end with ; or \g. Your MySQL connection id is 16168
Server version: 5.0.62-log Source distribution
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

```
```

Creating and Checking Tables

By now, you know that you’re using a database named after your Sandbox login, that we created especially for you. But what’s in that database?

```
mysql> show tables;
Empty set (0.00 sec)
mysql>
```

Creating a Table

OK, you’re sitting in an empty database. It’s time to add a table. Now, we need to think of a purpose for this table. For now, let’s make a table that will hold a list of your friends and some of their statistics. Let’s think of some information about your friends that you might want in a database.

Let’s try first name, last name, birthday, gender, height, and weight.

**Note** When you create tables, it’s best to be careful from the start. Plan everything in advance before you actually begin creating anything.

Any idea what the command would be to create a table? If you thought of `create`, you’re catching on quickly.

```
mysql> create table friends (first_name varchar(20), last_name varchar(20), birthday date, gender char(1), height int(3) unsigned, weight int(4) unsigned);
```

Check out the table that you just created by using the `show tables` command. The table `friends` should be in the list.
Phew! That was a lot of text just to create a table! Let's go over what we just did. First, we created a table named `friends`. Then, in a set of parentheses, we put in all the data and the corresponding data types.

We have `first_name` as a `varchar` data type, which stands for varying characters. The number 20 (in parentheses) following `varchar` indicates that the `first_name` data entries can be up to 20 characters long.

The `char` is very similar to `varchar` in that it also takes ASCII characters such as "a", "b", "c", "1", "2", ",", ">", and so on. The `char` data type can be set to take in a maximum of 255 characters, while `varchar` can take up to 65,535 characters. The key difference between the two lies in the fact that they allocate a different amount of memory space.

To make a memory-efficient table, try to follow these guidelines:

- If you don't know exactly how many characters the data will be, or if it might be more than 255 characters, use `varchar`.
- If you know the data's exact size and it's less than 255 characters, use `char`.

To illustrate, it's wise to use `varchar` for names since you don't know exactly how many characters will be involved from entry to entry. For gender (gender) (m/f), yes/no fields, and other short data fields, it's good to use `char`.

Moving right along, the birthday is of the `date` data type. A date in MySQL is in the format year-month-day, such as `2015-09-21`. It's more advantageous to use the `date` data type for dates than to use the `char` type—we'll see why in a bit.

We can represent gender using a single character (m or f—and you can use other codes like "d" for decline to answer), so we'll use the `char` data type. Height and weight are integers. Integers refer to all whole numbers, including negative whole numbers (52, 5, 0, -1, -40). However, notice the word `unsigned` that is within the code. `Unsigned` disables negative numbers. This is useful here since we won't need negative numbers for height and weight. Choosing the right type of variables will save a lot of disk space (especially when your database is huge) and minimize the data access time.

Oops, I almost forgot one of the data types!

The `text` data type is very similar to the `varchar` and `char` data types, except that we do not specify the maximum length in parentheses after the word `text`. The text data type might be useful if you need a data column to store a lengthy description of an item or some other long piece of text. Be careful with long text entries; the whole structure of a database is such that entries are stable and somewhat permanent. So, say you make a spelling error in that long text variable that you stored and now you want to change it. It would be do-able, but correcting such errors is difficult. (We could use the `alter` command to do so, but let's worry about that later.)

### Checking Tables

Now that you've created the table named `friends`, let's make sure everything is in order. Try this:
For now, don’t worry about the Null, Key, and Extra columns. The Default column simply states that if no data is entered in that particular field, a NULL value is assigned (basically, it’s left empty). The useful columns here are Field and Type. Sometimes database programmers can get lost and forget the types of data and what names they’ve created. To find out the names of the fields and the data types, we simply use `describe`. Pretty handy, eh?

Okay, now that the table is set up, we’ll need to enter some data. But before we do that, let me just give you a quick reference table of all data types out there that you can use. We won’t cover all of them here, but the data types we will cover can handle just about any normal database.

<table>
<thead>
<tr>
<th>Numeric Data Types</th>
<th>Space Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>TINYINT</td>
<td>1 byte</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>2 bytes</td>
</tr>
<tr>
<td>MEDIUMINT</td>
<td>3 bytes</td>
</tr>
<tr>
<td>INT</td>
<td>4 bytes</td>
</tr>
<tr>
<td>INTEGER</td>
<td>4 bytes</td>
</tr>
<tr>
<td>BIGINT</td>
<td>8 bytes</td>
</tr>
<tr>
<td>FLOAT(4)</td>
<td>4 bytes</td>
</tr>
<tr>
<td>FLOAT(8)</td>
<td>8 bytes</td>
</tr>
<tr>
<td>FLOAT</td>
<td>4 bytes</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>8 bytes</td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>8 bytes</td>
</tr>
<tr>
<td>REAL</td>
<td>8 bytes</td>
</tr>
<tr>
<td>DECIMAL(N,D)</td>
<td>N bytes (D+2, if N&lt;D)</td>
</tr>
<tr>
<td>NUMERIC(N,D)</td>
<td>N bytes (D+2, if N&lt;D)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Date and Time Data Types</th>
<th>Space Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>3 bytes</td>
</tr>
<tr>
<td>DATETIME</td>
<td>8 bytes</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>4 bytes</td>
</tr>
<tr>
<td>TIME</td>
<td>3 bytes</td>
</tr>
<tr>
<td>YEAR</td>
<td>1 byte</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>String Data Types</th>
<th>Space Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR(X)</td>
<td>X bytes, where X &lt;= 2^8 (255)</td>
</tr>
<tr>
<td>VARCHAR(X)</td>
<td>L+1 or 2 bytes, where X &lt;= 2^16 (65,535)</td>
</tr>
<tr>
<td>TINYBLOB, TINYTEXT</td>
<td>L+1 bytes, where L &lt; 2^8 (255)</td>
</tr>
</tbody>
</table>
Inserting Values into Tables

Now, let's enter some values into the table "friends" that we've already set up.

Type the following command at your MySQL prompt:

```
mysql> insert into friends values ('Hoyoul', 'Kang', -> '1979-08-08', 'm', 165, 145);
Query OK, 1 row affected (0.00 sec)
```

If something went wrong (for example, if you forgot to type a character), often MySQL will tell you the approximate location of the error.

We've successfully inserted one entry into the friends table, using the command `insert into`. The table name follows the command, followed by the column values within a set of parentheses.

Note

Within the parentheses, you must enter the values in the exact order you specified when you first created the table format. In other words, "Hoyoul" and "Kang" are the first_name and the last_name columns, respectively, that you created previously.

Values that fit into the `varchar`, `char`, `text`, and `date` data types need to be within either single or double quotes (if the values contain quotes, there are other rules for entry; for example, using backticks (`) instead of quotes). When we get into PHP interfacing, we'll need to use single quotes due to **interpolation**, so you might want to make a habit of using single quotes to define values from now on.

Inserting Specific Column Values

You can also choose which specific fields you want to enter using the **insert into** statement. Let's say we want to enter another row into our friends table with first name = 'Joe', birthday = '1955-12-12', gender = 'm', weight = '160'. Here's how you would go about doing it.

Type the following command at your MySQL prompt:

```
mysql> insert into friends (first_name, birthday, gender, weight) -> values ('Joe', '1955-12-12', 'm', '160');
```

Right after the table name, you type the names of the fields you want to insert values into and then enter the corresponding values, in the same order, into the table. So 'Joe' corresponds to **first_name**, '1955-12-12' corresponds to **birthday**, and so on. The fields for which you didn't supply a value would simply contain the default **NULL** (an empty value).

Retrieving Data from Tables

Let's retrieve the data we've just entered.
Here we have the keyword `select`, which retrieves the columns and their values in a certain table—in our case, `friends`. The star (*) is a wildcard—a shortcut—that in our case means all column values.

Take some time to insert some more of your friends and their statistics into your table. Try to add at least five more. That should get you really comfortable with using the Insert syntax. Try to enter different values for all the different fields so you have a good amount of variety in your data—the more friends you enter at this point, the more fun you can have later when we start to play around with querying.

Be sure to play around with `create`, `select`, and `insert`, and to hand in your assignments from the syllabus. Please make sure to read all the comments on the Graded links once your coursework has been returned. See you in the next lesson!
Exploring the 'where' Keyword

We took a quick look at the select keyword in our previous lab. Do you remember the syntax of that command? Let's look at it again.

Type the following command at your MySQL prompt:

```
mysql> select * from friends;
```

Here we're selecting to view all the columns (the asterisk * symbol or "wildcard" represents "all") from the table friends.

Let's try a more complicated query. Say we want to get a list of the first names of all your female friends. We can do this in one command:

Type the following command at your MySQL prompt:

```
mysql> select first_name from friends where gender = 'f';
```

```
+------------+
| first_name |
+------------+
| Trish      |
| Kerry      |
| Dessie     |
+------------+
3 rows in set (0.00 sec)
```

mysql>

This statement is very similar to an English sentence, isn’t it? The previous command line could be read, "select the first_name column from table friends where gender is equal to 'f'." The part after the keyword where is called the conditional statement, and it's used just like you would use it in any programming language—for instance, a PHP if/else statement.

Type the following command at your MySQL prompt:

```
mysql> select first_name from friends where gender = 'f' or gender = 'm';
```

The operator or is used here to expand our options when retrieving data. Basically, this means that if the gender matches 'f' or 'm', then MySQL returns the first_name value for the entry to be displayed.

Type the following command at your MySQL prompt:

```
mysql> select first_name, last_name, birthday -> from friends where height > 175 and weight < 160;
```

The three names of the fields (first_name, last_name, and birthday) are given (separated by commas) to specify that you want those three fields to be displayed. Since we have two conditions, and we need to satisfy both of them, we use the and operator. The first condition was that the height must be greater than 175 centimeters, which was indicated with height > 175. The second condition was that weight was under 160 pounds, represented by weight < 160.

Observe the following comparison operators:
### The 'like' Selector

If you've worked with programming languages, you probably recognized the conditional statement operators. However, you may not have seen the special SQL `like` selector yet.

Type the following command at your MySQL prompt:

```sql
mysql> select first_name,birthday from friends where first_name like ('Des%');
```

+------------+------------+
<table>
<thead>
<tr>
<th>first_name</th>
<th>birthday</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dessie</td>
<td>1975-08-01</td>
</tr>
</tbody>
</table>
+------------+------------+
1 row in set (0.00 sec)

In MySQL, the keyword `like` is used as an **operator**, to search for column values that match a certain string pattern—in our case, we wanted to find Dessie's birthday, but couldn't remember how her name was spelled. Because we know it starts with "Des," we use the percent sign (%) **wildcard** to represent the rest of her name, however it may be spelled. Of course, if you have friends named "Destiny" or "Deshawn," their birthdays will be listed as well.

### Updating and Deleting Data

Being able to search a table is great, but tables aren't much good if you can't update or delete entries. Luckily, doing these tasks is just as simple as searching.

#### Updating Entries

When you retrieved your friends' data using the `select` keyword, did you notice something strange about Hoyoul's height? His weight looks normal in pounds—145 pounds is a reasonable human weight—but if we take his height in inches, 165 inches would make him over 13 feet tall!

To make our data consistent, we should pick either the Imperial or the Metric system for both columns. Let's go with Imperial, which would make his height 69 inches. Update it as shown:

```sql
mysql> update friends set height=69 where first_name='Hoyoul';
Query OK, 1 rows affected (0.00 sec)
```

Now let's use `select` again to check our results:
Type the following into your MySQL prompt:

```sql
mysql> select * from friends where first_name='hoyoul';
+------------+-----------+------------+---------+--------+--------+
| first_name | last_name | birthday   | gender  | height | weight |
|------------|-----------|------------|---------|--------|--------+
| Hoyoul     | Kang      | 1979-08-08 | m       | 69     | 145    |
+------------+-----------+------------+---------+--------+--------+
1 row in set (0.00 sec)
```

And if you want to update multiple column values:

Type the following into your MySQL prompt:

```sql
mysql> update friends set birthday='1975-09-30',gender='f' where last_name='Parmelee';
+------------+-----------+------------+--------+--------+--------+
| first_name | last_name | birthday   | gender | height | weight |
|------------|-----------|------------|--------|--------|--------+
| Kerry      | Parmelee  | 1975-09-30 | f      | NULL   | NULL   |
+------------+-----------+------------+--------+--------+--------+
1 row in set (0.00 sec)
```

Deleting Entries

We’ve decided that since we don’t know Joe’s last name, we can’t really consider him a friend—more of an acquaintance, really. So we need to remove him from our table, along with anyone whose last name was never mentioned:

Type the following into your MySQL prompt:

```sql
mysql> delete from friends where last_name is null;
Query OK, 1 row affected (0.00 sec)
mysql>
```

And now check our table, using the `select` keyword:

Type the following into your MySQL prompt:

```sql
mysql> select * from friends;
+------------+-----------+------------+--------+--------+--------+
| first_name | last_name | birthday   | gender | height | weight |
|------------|-----------|------------|--------|--------|--------+
| Hoyoul     | Kang      | 1979-08-08 | m      | 69     | 145    |
| Trish      | Gray      | 0000-00-00 | f      | 63     | 116    |
| Kerry      | Beck      | 0000-00-00 | f      | 72     | 135    |
| Dessie     | Coale     | 1975-08-01 | f      | 68     | 125    |
| Kerry      | Parmelee  | 1975-09-30 | f      | 0      | 0      |
+------------+-----------+------------+--------+--------+--------+
5 rows in set (0.00 sec)
```

Here, the `delete` keyword was used to delete rows of data, chosen by the conditional statement `last_name is null`. There’s no need to specify a wildcard (*) or any column names, since the entire row will always be deleted.

But why was the word `is` used instead of the equality operator (=)? Since `null` is a special keyword indicating an empty cell, MySQL requires the operator `is` in order to prevent `null` from being confused with a string, such
as 'hello' or, incidentally, 'null.'

Get in your fill of practice, and be sure to hand in your assignments. See you in the next lesson!
Managing Tables

Altering Tables

The overall idea of having a database is to allow many entries over a long period of time. The database columns should therefore be well thought-out beforehand, and we should try to anticipate possible future changes. However, once in a while, something completely unexpected happens and you have to alter the table columns to add new categories or change existing ones.

Before we discuss the alter command, let's create another table to play with:

```
Type the following command at your MySQL prompt:

mysql> create table officeproducts (product_name varchar(20), -> price decimal(6,2));
Query OK, 0 rows affected (0.02 sec)
mysql>
```

You now have a table called officeproducts with two columns of information—product_name and price.

As it turns out, we forgot to add a description column. Luckily, SQL contains many commands to deal with human error, and one of those is alter table. Let's alter our officeproducts table to add a description column.

```
Type the following command at your MySQL prompt:

mysql> alter table officeproducts add description text;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0
mysql>
```

Now let's check the table to make sure it was changed correctly.

```
Type the following command at your MySQL prompt:

mysql> describe officeproducts;
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| product_name | varchar(20)  | YES  |     | NULL    |       |
| price        | decimal(6,2) | YES  |     | NULL    |       |
| description  | text         | YES  |     | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
mysql>
```

We simply used the alter table and add commands to add a description column with type text.

Deleting and Reordering Columns

If we decide that we no longer want a column in our table, we can drop it. However, if we have data in that column, we will lose it, so we must be very careful when we use drop. Suppose we want to drop the description column.
Type the following command at your MySQL prompt:

```sql
mysql> alter table officeproducts drop description;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

**Note** Be careful using the `drop` command! When you drop the column, you drop all of its data.

We used the `drop` command to delete the column called `description` from the table `officeproducts`. Notice that, unlike adding columns, when dropping columns we don’t need to specify the column type.

Now, suppose we really do want that description column, but we want it to come after `product_name`. Try this:

Type the following commands at your MySQL prompt:

```sql
mysql> alter table officeproducts add description text after product_name;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```sql
mysql> describe officeproducts;
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
|--------------+--------------+------+-----+---------+-------+
| product_name | varchar(20)  | YES  |     | NULL    |       |
| description  | text         | YES  |     | NULL    |       |
| price        | decimal(6,2) | YES  |     | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
3 rows in set (0.01 sec)
```

This time, we added `after column_name` to our command line. When you insert a column using the `alter` command, the new column is appended after the last column by default. Using `after` enables you to specify where the new column goes. To put your new column up front as the first column of the table, you can use `first` instead of `after`:

```sql
OBSERVE:
alter table officeproducts add description text first;
```

To reorder a table without losing any data, use `alter table` with `modify`. Suppose we decide after all this that we’d like to move `description` after `price` again. Try this:

Type the following command at your MySQL prompt:

```sql
mysql> alter table officeproducts modify description text after price;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Now let’s check the table to make sure it was changed correctly.
Changing Column Type

What if you want to change the data type of an existing column? Suppose we want to change the data type of column description from text to varchar(255). Here's how we would do it:

Type the following commands at your MySQL prompt:

```
mysql> alter table officeproducts modify description varchar(255);
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```
mysql> describe officeproducts;
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| product_name | varchar(20)  | YES  |     | NULL    |       |
| price        | decimal(6,2) | YES  |     | NULL    |       |
| description  | varchar(255) | YES  |     | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Syntax-wise, the modify command above is almost exactly like the add command. However, rather than adding a new column, it takes an existing column and alters only its data type, to whatever you specify. In our case, it takes the description column and changing its data type from text to varchar(255).

Renaming Tables

Sometimes, you might need to change the name of a table after creating it. Instead of recreating the table and typing all of the column names and column types all over again, you can just rename the table.

Type the following command at your MySQL prompt:

```
mysql> alter table officeproducts rename as office_prod;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> describe officeproducts;
ERROR 1146: Table 'certjosh.officeproducts' doesn't exist
```

What just happened? By using the command rename as, we've taken the table officeproducts and renamed it office_prod. As a result, the table officeproducts no longer exists. Let's try this:
Type the following command at your MySQL prompt:

```
mysql> describe office_prod;
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| product_name | varchar(20)  | YES  |     | NULL    |       |
| description  | varchar(255) | YES  |     | NULL    |       |
| price        | decimal(6,2) | YES  |     | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Now that looks better.

**Deleting Tables**

We know how to create, alter, and rename tables. But what if we had an old table we no longer needed and wanted to remove? We’ll need to drop the table, meaning we’ll remove the table and all the entries inside of it. Let’s go ahead and drop `office_prod` altogether.

Type the following command at your MySQL prompt:

```
mysql> drop table office_prod;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> describe office_prod;
ERROR 1146: Table 'certjosh.office_prod' doesn't exist
```

By simply typing `drop table`, then the table name `office_prod`, we have deleted the entire table from our SQL database.

**WARNING**  When you drop a table, it is gone forever! If you have an essential table with lots of entries in it, all will be lost. Use extreme caution with the `drop` command!

Before moving on, be sure to hand in your assignments for this lesson. See you in the next one!
Managing Table Entries

You've learned how to create, alter, search, update, and delete tables in SQL. Theoretically, you could get by in SQL with these skills alone—especially with the limited data we've been using so far.

However, the reality is that an SQL table will probably end up with tons and tons of data. That's really the reason for creating an automated database system, isn't it? And since that's the case, we need to take more care with our tables to ensure their integrity and efficiency.

Optimizing with Keys/Indexes

Are you connected to your SQL database? Good. Let's create a new table, a **table_of_contents**, that contains the chapter names, descriptions, and topics of a book we want to post online:

<table>
<thead>
<tr>
<th>Type the following command at your MySQL prompt:</th>
</tr>
</thead>
<tbody>
<tr>
<td>mysql&gt; create table table_of_contents (chapter_name varchar(250), chapter_description text, chapter_topic varchar(25));</td>
</tr>
<tr>
<td>Query OK, 0 rows affected (0.01 sec)</td>
</tr>
<tr>
<td>mysql&gt;</td>
</tr>
</tbody>
</table>

Now let's fill the table with some book chapters.

<table>
<thead>
<tr>
<th>Type the following command at your MySQL prompt:</th>
</tr>
</thead>
<tbody>
<tr>
<td>mysql&gt; insert into table_of_contents values ('The Client Program', 'Recipes for the mysql interface itself.', 'queries');</td>
</tr>
<tr>
<td>Query OK, 0 rows affected (0.01 sec)</td>
</tr>
<tr>
<td>mysql&gt; insert into table_of_contents values ('Writing External Programs', 'Recipes for MySQL-based programs.', 'DBI');</td>
</tr>
<tr>
<td>Query OK, 0 rows affected (0.01 sec)</td>
</tr>
<tr>
<td>mysql&gt; insert into table_of_contents values ('Record Selection', 'Recipes for formatting MySQL output.', 'SELECT');</td>
</tr>
<tr>
<td>Query OK, 0 rows affected (0.01 sec)</td>
</tr>
<tr>
<td>mysql&gt;</td>
</tr>
</tbody>
</table>

Fill this table with a few more chapters, and you'll have something that looks like this:
**INTERACTIVE SESSION:**

```sql
mysql> select * from table_of_contents;
+----------------------------------|---------------------------------------------------+
<table>
<thead>
<tr>
<th>chapter_name</th>
<th>chapter_description</th>
</tr>
</thead>
</table>
+----------------------------------|---------------------------------------------------+
| The Client Program               | Recipes for the mysql interface itself.          |
| queries                          |                                                   |
| Writing External Programs        | Recipes for MySQL-based programs.                |
| DBI                              |                                                   |
| Record Selection                 | Recipes for formatting MySQL output.             |
| SELECT                           |                                                   |
| Working with Strings             | Recipes for controlling strings in output.       |
| pattern matching                 |                                                   |
| Working with Dates and Times     | Recipes for controlling date/time in output.     |
| TIMESTAMP                        |                                                   |
| Sorting Query Results            | Recipes for sorting SELECT results.              |
| ORDER BY                         |                                                   |
| Generating Summaries             | Recipes for using groupings, count(), min and max.|
| GROUP BY                         |                                                   |
| Introduction to MySQL on the Web | Formatting MySQL output into web pages.           |
| DBI                              |                                                   |
+----------------------------------|---------------------------------------------------+
8 rows in set (0.00 sec)
```

This table looks pretty organized and intuitive for a book, right? In fact, you can use the `chapter_topic` column to find the chapter that contains it.

**Type the following command at your MySQL prompt:**

```sql
mysql> select chapter_name,chapter_description from table_of_contents where chapter_topic='GROUP BY';
+----------------------+----------------------------------------------------+
<table>
<thead>
<tr>
<th>chapter_name</th>
<th>chapter_description</th>
</tr>
</thead>
</table>
+-----------------------|----------------------------------------------------+
<table>
<thead>
<tr>
<th>Generating Summaries</th>
<th>Recipes for using groupings, count(), min and max.</th>
</tr>
</thead>
</table>
+-----------------------|----------------------------------------------------+
1 row in set (0.00 sec)
```

Pretty handy, huh? Think about it. If you wanted to find a specific topic in a physical book, unless it had the rare, really great index, you’d probably have to flip through many pages in that book in order to find what you’re looking for. And, once you found your topic, there’d be no guarantee that the topic didn’t show up again in other chapters. If that book happened to be several hundred pages long, it would take way too long to find all the instances of your topic.

But wait—is SQL searching any differently from how you would search?
Type the following command at your MySQL prompt:

```
mysql> EXPLAIN select chapter_name, chapter_description from table_of_contents where chapter_topic='GROUP BY';
```

```
+----+-------------+-------------------+------+---------------+------+---------+------+
| id | select_type | table             | type | possible_keys | key  | key_len | ref  |
| rows | Extra       |
+----+-------------+-------------------+------+---------------+------+---------+------+
| 1  | SIMPLE      | table_of_contents | NULL | NULL          | NULL | NULL    | NULL |
| 8   | Using where |
+----+-------------+-------------------+------+---------------+------+---------+------+
1 row in set (0.00 sec)
```

Note  
The `EXPLAIN` keyword can be used in front of any SELECT statement, to analyze the optimization of that search.

Wow. Turns out SQL searches exactly like you do—looking at all eight rows in a table to find the topic in question. No big deal, though, right? I mean, it's only eight rows, after all.

But what if we were looking for "GROUP BY" in an entire online library, like Safari Online Books? What if we were looking for "GROUP BY" in 550 BILLION web pages worth of data, like Google? Imagine waiting hours and hours for your Google search to complete—without optimization, even super-fast SQL would get unacceptably bogged down.

Let's get real here. When you are looking up a topic in a physical book, you don't flip through indiscriminately. You look in the index in the back of the book to find it:

```
get-upandmove (diff, 665)
GRANT statement, 2
GROUP BY clause
    duplicates, preventing in query results with, 710
    per-group descriptive statistics, application to, 679
GROUP BY clauses, 350, 365
    errors associated with, 352
    expressions, using with, 359
    range values, 361
    summaries for temporal values and, 368
groups, 8
guess_table.pl, 516–319
H
Harness.java, 84
harness.php, 79
keynote.pl, 77
```

As it turns out, SQL can also create an index or key to help with searches.

Type the following command at your MySQL prompt:

```
mysql> alter table table_of_contents add key(chapter_topic);
```

```
Query OK, 8 rows affected (0.01 sec)
Records: 8  Duplicates: 0  Warnings: 0
```

In the statement above, we've added a **key** for the column **chapter_topic** to let SQL know that we plan to search on that column quite a bit. SQL responded by creating a new, invisible table, specially designed to speed up any search we do on **chapter_topic**.
Type the following command at your MySQL prompt:

```
mysql>   describe table_of_contents;
+---------------------+--------------+------+-----+---------+-------+
| Field               | Type         | Null | Key | Default | Extra |
+---------------------+--------------+------+-----+---------+-------+
| chapter_name        | varchar(250) | YES  |     | NULL    |       |
| chapter_description | text         | YES  |     | NULL    |       |
| chapter_topic       | varchar(25)  | YES  | MUL | NULL    |       |
+---------------------+--------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

As you can see, the `chapter_topic` column is now flagged as a **multiple key**, meaning that a value can be allowed more than once in this column (like the SELECT topic in our table).

So how does this help our search? Let's find out:

Type the following command at your MySQL prompt:

```
mysql>   EXPLAIN select chapter_name,chapter_description from table_of_contents where chapter_topic='GROUP BY';
```

```
+----+-------------+-------------------+------+---------------+---------------+--------
| id | select_type | table             | type | possible_keys | key           | key_len |
|    | ref         | rows | Extra       |
| ref | 1 | const | 1 | Using where |               |
+----+-------------+-------------------+------+---------------+---------------+--------
1 row in set (0.00 sec)
```

And there you have it—suddenly, SQL goes from having to look at every single entry in the table, to finding “GROUP BY” in one shot. That's a powerful thing, especially as the size and complexity of your tables increases.

### Auto_increment and Primary Keys

You may have already noticed that we have left out a very important column in our `table_of_contents` table: the chapter number itself! The order of chapters is incredibly important in a book, so there has to be a way to specify which goes where.

Let's add the chapter number:

```
mysql>   alter table table_of_contents add chapter int unsigned NOT NULL first;
```
Type the following command at your MySQL prompt:

```sql
mysql> select * from table_of_contents;
+---------+----------------------------------+-----------------------------------------+------------------+
| chapter | chapter_name                      | chapter_description                   | chapter_topic |
+---------+----------------------------------+-----------------------------------------+------------------+
|       0 | The Client Program               | Recipes for the mysql interface itself. | queries          |
|       0 | Writing External Programs        | Recipes for MySQL-based programs.     | DBI             |
|       0 | Record Selection                 | Recipes for formatting MySQL output.  | SELECT          |
|       0 | Working with Strings             | Recipes for controlling strings in output. | pattern matching | |
|       0 | Working with Dates and Times     | Recipes for controlling date/time in output. | TIMESTAMP       | |
|       0 | Sorting Query Results            | Recipes for sorting SELECT results.   | ORDER BY        |
|       0 | Generating Summaries             | Recipes for using groupings, count(), min and max. | GROUP BY        |
|       0 | Introduction to MySQL on the Web | Formatting MySQL output into web pages. | DBI             |
+---------+----------------------------------+-----------------------------------------+------------------+
8 rows in set (0.01 sec)
```

At this point, we could simply `update` the table with all the chapter numbers by hand. However, because we know that each chapter will have one unique, incremental number assigned to it, it makes sense to automate this process to prevent human error.

In order to do that, first we need to tell MySQL (like we did before with `chapter_topic`) that `chapter` will be a key column.

Type the following command at your MySQL prompt:

```sql
mysql> alter table table_of_contents add key(chapter);
Query OK, 8 rows affected (0.01 sec)
Records: 8  Duplicates: 0  Warnings: 0
```

Now, use `alter table` to modify the type of `chapter`:

Type the following command at your MySQL prompt:

```sql
mysql> alter table table_of_contents modify chapter int unsigned NOT NULL auto_increment;
Query OK, 8 rows affected (0.01 sec)
Records: 8  Duplicates: 0  Warnings: 0
```

When we modified the type of `chapter`, we left everything intact and simply added `auto_increment` to the end. Here's what SQL did:
Voila! SQL has filled in the chapter order, just how we wanted. Now, when we want to add a chapter, we can simply leave `chapter` blank, and it will be filled in for us. Like this:
Type the following command at your MySQL prompt:

```
mysql> insert into table_of_contents values ('','Modifying Tables',
    -> 'Dropping, adding, and changing columns','ALTER TABLE');
Query OK, 1 row affected (0.00 sec)
mysql> select * from table_of_contents;
```

```
+---------+----------------------------------+-----------------------------------------+-----------+------------------+
| chapter | chapter_name                     | chapter_description                    | chapter_topic |
+---------+----------------------------------+-----------------------------------------+-----------+------------------+
|       1 | The Client Program               | Recipes for the mysql interface itself. | queries   |
|       2 | Writing External Programs        | Recipes for MySQL-based programs.      | DBI       |
|       3 | Record Selection                 | Recipes for formatting MySQL output.   | SELECT    |
|       4 | Working with Strings             | Recipes for controlling strings in output. | pattern matching |
|       5 | Working with Dates and Times     | Recipes for controlling date/time in output. | TIMESTAMP |
|       6 | Sorting Query Results            | Recipes for sorting SELECT results.    | ORDER BY  |
|       7 | Generating Summaries             | Recipes for using groupings, count(), min and max. | GROUP BY |
|       8 | Introduction to MySQL on the Web | Formatting MySQL output into web pages. | DBI       |
|       9 | Modifying Tables                 | Dropping, adding, and changing columns  | ALTER TABLE |
+---------+----------------------------------+-----------------------------------------+-----------+------------------+
9 rows in set (0.00 sec)
```

```
mysql>
```

**The Primary Key**

Because our `chapter` column is considered a unique identifier, we can go one step further than simply naming it as a key. In this case, the optimal thing to do would be to name `chapter` as the table's primary key.
Type this command at your MySQL prompt:

```sql
mysql> alter table table_of_contents add primary key(chapter);
Query OK, 8 rows affected (0.01 sec)
Records: 8  Duplicates: 0  Warnings: 0

mysql> describe table_of_contents;
+---------------------+------------------+------+-----+---------+---------------+
| Field               | Type             | Null | Key | Default | Extra         |
+---------------------+------------------+------+-----+---------+---------------+
| chapter             | int(10) unsigned |      | PRI | NULL    | auto_increment|
| chapter_name        | varchar(250)     | YES  |     | NULL    |               |
| chapter_description | text             | YES  |     | NULL    |               |
| chapter_topic       | varchar(25)      | YES  | MUL | NULL    |               |
+---------------------+------------------+------+-----+---------+---------------+
4 rows in set (0.00 sec)

mysql>
```

Only one primary key can be named for a table, because this indicates to SQL that this column is indeed the table’s unique identifier. For this reason, of course, every row in that column absolutely must be unique.

Now that we’ve learned how to keep our tables optimized, we’re ready to increase the complexity of our database. Be sure to hand in your assignments, and see you in the next lesson!

*Copyright © 1998-2014 O’Reilly Media, Inc.*

*This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License. See [http://creativecommons.org/licenses/by-sa/3.0/legalcode](http://creativecommons.org/licenses/by-sa/3.0/legalcode) for more information.*
So far, we've worked with simple tables in SQL, and they have worked well for our simple examples. However, databases can quickly become enormous and incredibly complex. How do we deal with information when it outgrows one table?

What is a Relational Database?

Let's go back to our online book example. Are you connected to your SQL Database? Good.

<table>
<thead>
<tr>
<th>chapter</th>
<th>chapter_name</th>
<th>chapter_description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chapter_topic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>The Client Program</td>
<td>Recipes for the mysql interface itself.</td>
</tr>
<tr>
<td></td>
<td>queries</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Writing External Programs</td>
<td>Recipes for MySQL-based programs.</td>
</tr>
<tr>
<td></td>
<td>DBI</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Record Selection</td>
<td>Recipes for formatting MySQL output.</td>
</tr>
<tr>
<td></td>
<td>SELECT</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Working with Strings</td>
<td>Recipes for controlling strings in output.</td>
</tr>
<tr>
<td></td>
<td>pattern matching</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Working with Dates and Times</td>
<td>Recipes for controlling date/time in output.</td>
</tr>
<tr>
<td></td>
<td>pattern matching</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Sorting Query Results</td>
<td>Recipes for sorting SELECT results.</td>
</tr>
<tr>
<td></td>
<td>ORDER BY</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Generating Summaries</td>
<td>Recipes for using groupings, count(), min and max.</td>
</tr>
<tr>
<td></td>
<td>GROUP BY</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Introduction to MySQL on the Web</td>
<td>Formatting MySQL output into web pages.</td>
</tr>
<tr>
<td></td>
<td>DBI</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Modifying Tables</td>
<td>Dropping, adding, and changing columns</td>
</tr>
<tr>
<td></td>
<td>ALTER TABLE</td>
<td></td>
</tr>
</tbody>
</table>

There's a big problem with this table: the topic column. If you think about it, a book chapter's almost always going to cover lots of topics, not just one. But how do we include all the chapter topics without inserting the chapter name and description over and over again? The chapter column is auto-increment, so it won't even allow duplicate chapters anyway.

There's a simple solution to this dilemma: Let's create a new table, just for the chapter topics.

First, we need to drop the chapter_topic column from table_of_contents:

```
mysql> alter table table_of_contents drop chapter_topic;
Query OK, 9 rows affected (0.02 sec)
Records: 9  Duplicates: 0  Warnings: 0
```
Now, create the new table, called chapter_topics:

```sql
mysql> create table chapter_topics (topic_id int unsigned NOT NULL auto_increment, chapter int unsigned NOT NULL, topic varchar(25) NOT NULL, primary key(topic_id), key(chapter), key(topic));
Query OK, 0 rows affected (0.00 sec)

mysql> describe chapter_topics;
+----------+------------------+------+-----+---------+----------------+
| Field    | Type             | Null | Key | Default | Extra          |
+----------+------------------+------+-----+---------+----------------+
| topic_id | int(10) unsigned |      | PRI | NULL    | auto_increment |
| chapter  | int(10) unsigned |      | MUL | 0       |                |
| topic    | varchar(25)      |      | MUL |         |                |
+----------+------------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)
```

**Note** Did you see how we were able to do all the optimization from the last lesson within one `create table` statement? Pretty handy!

By splitting our data between two tables, we've created a perfect example of a relational database. The two tables are related because of the `chapter` column, mirrored in both tables as a touch point. This can be done with any number of tables. For some definitions of relational databases, see this answers.com article.

In fact, MySQL is considered a relational database management system (RDBMS), and SQL as a language was built specifically for many RDBMS, not just MySQL. Here's what Wikipedia has to say.

Before moving on, we need to fill the `chapter_topics` table with some values. Use the `insert into` statement to get something like the following:

**OBSERVE:**

```sql
mysql> select * from chapter_topics;
+----------+---------+------------------+
<table>
<thead>
<tr>
<th>topic_id</th>
<th>chapter</th>
<th>topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>queries</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>DBI</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>PHP</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>SELECT</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>WHERE</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>LIMIT</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>temporary tables</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>pattern matching</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>date_format</td>
</tr>
<tr>
<td>11</td>
<td>6</td>
<td>ORDER BY</td>
</tr>
<tr>
<td>12</td>
<td>7</td>
<td>GROUP BY</td>
</tr>
<tr>
<td>13</td>
<td>8</td>
<td>DBI</td>
</tr>
<tr>
<td>14</td>
<td>8</td>
<td>HTML</td>
</tr>
</tbody>
</table>
+----------+---------+------------------+
14 rows in set (0.00 sec)
```

**Inner Joins**

Now that we have our tables ready, let's get into some basic relational databasing. To combine the two tables we just created above, we would perform an `inner join` on the two tables, using one of the following commands:
Type any of the following commands at your MySQL prompt:

```sql
mysql> select * from table_of_contents, chapter_topics;
mysql> select * from table_of_contents join chapter_topics;
mysql> select * from table_of_contents cross join chapter_topics;
```

**Note** Although MySQL supports all three of these statements, the first one is preferred over the others due to some limitations on the other two.
<table>
<thead>
<tr>
<th>chapter</th>
<th>chapter_name</th>
<th>chapter_description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topic_id</td>
<td>chapter</td>
<td>topic</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>queries</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>queries</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>queries</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>queries</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>queries</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>queries</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>queries</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>queries</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>queries</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>queries</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>DBI</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>DBI</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>DBI</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>DBI</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>DBI</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>DBI</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>DBI</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>DBI</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>PHP</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>PHP</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>PHP</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>PHP</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>PHP</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>PHP</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>PHP</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>PHP</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>PHP</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>SELECT</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>SELECT</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>SELECT</td>
</tr>
<tr>
<td>4</td>
<td>Working with Strings</td>
<td>Recipes for controlling strings in output.</td>
</tr>
<tr>
<td>4</td>
<td>Working with Dates and Times</td>
<td>Recipes for controlling date/time in output.</td>
</tr>
<tr>
<td>4</td>
<td>Sorting Query Results</td>
<td>Recipes for sorting SELECT results.</td>
</tr>
<tr>
<td>4</td>
<td>Generating Summaries</td>
<td>Recipes for using groupings, count(), min and max.</td>
</tr>
<tr>
<td>8</td>
<td>Introduction to MySQL on the Web</td>
<td>Formatting MySQL output into web pages.</td>
</tr>
<tr>
<td>9</td>
<td>Modifying Tables</td>
<td>Dropping, adding, and changing columns</td>
</tr>
<tr>
<td>5</td>
<td>The Client Program</td>
<td>Recipes for the mysql interface itself.</td>
</tr>
<tr>
<td>2</td>
<td>Writing External Programs</td>
<td>Recipes for MySQL-based programs.</td>
</tr>
<tr>
<td>6</td>
<td>Sorting Query Results</td>
<td>Recipes for sorting SELECT results.</td>
</tr>
<tr>
<td>7</td>
<td>Generating Summaries</td>
<td>Recipes for using groupings, count(), min and max.</td>
</tr>
<tr>
<td>8</td>
<td>Introduction to MySQL on the Web</td>
<td>Formatting MySQL output into web pages.</td>
</tr>
<tr>
<td>9</td>
<td>Modifying Tables</td>
<td>Dropping, adding, and changing columns</td>
</tr>
<tr>
<td>5</td>
<td>The Client Program</td>
<td>Recipes for the mysql interface itself.</td>
</tr>
<tr>
<td>5</td>
<td>Sorting Query Results</td>
<td>Recipes for sorting SELECT results.</td>
</tr>
<tr>
<td>7</td>
<td>Generating Summaries</td>
<td>Recipes for using groupings, count(), min and max.</td>
</tr>
<tr>
<td>8</td>
<td>Introduction to MySQL on the Web</td>
<td>Formatting MySQL output into web pages.</td>
</tr>
<tr>
<td>9</td>
<td>Modifying Tables</td>
<td>Dropping, adding, and changing columns</td>
</tr>
<tr>
<td>1</td>
<td>The Client Program</td>
<td>Recipes for the mysql interface itself.</td>
</tr>
<tr>
<td>2</td>
<td>Writing External Programs</td>
<td>Recipes for MySQL-based programs.</td>
</tr>
<tr>
<td>6</td>
<td>Sorting Query Results</td>
<td>Recipes for sorting SELECT results.</td>
</tr>
<tr>
<td>7</td>
<td>Generating Summaries</td>
<td>Recipes for using groupings, count(), min and max.</td>
</tr>
<tr>
<td>8</td>
<td>Introduction to MySQL on the Web</td>
<td>Formatting MySQL output into web pages.</td>
</tr>
<tr>
<td>9</td>
<td>Modifying Tables</td>
<td>Dropping, adding, and changing columns</td>
</tr>
<tr>
<td>1</td>
<td>The Client Program</td>
<td>Recipes for the mysql interface itself.</td>
</tr>
<tr>
<td>2</td>
<td>Writing External Programs</td>
<td>Recipes for MySQL-based programs.</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>pattern matching</td>
</tr>
<tr>
<td>3</td>
<td>Record Selection</td>
<td>Recipes for formatting MySQL output.</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>pattern matching</td>
</tr>
<tr>
<td>4</td>
<td>Working with Strings</td>
<td>Recipes for controlling strings in output.</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>pattern matching</td>
</tr>
<tr>
<td>5</td>
<td>Working with Dates and Times</td>
<td>Recipes for controlling date/time in output.</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>pattern matching</td>
</tr>
<tr>
<td>6</td>
<td>Sorting Query Results</td>
<td>Recipes for sorting SELECT results.</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>pattern matching</td>
</tr>
<tr>
<td>7</td>
<td>Generating Summaries</td>
<td>Recipes for using groupings, count(), min and max.</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>pattern matching</td>
</tr>
<tr>
<td>8</td>
<td>Introduction to MySQL on the Web</td>
<td>Formatting MySQL output into web pages.</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>pattern matching</td>
</tr>
<tr>
<td>9</td>
<td>Modifying Tables</td>
<td>Recipes for controlling strings in output.</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>pattern matching</td>
</tr>
<tr>
<td>1</td>
<td>The Client Program</td>
<td>Recipes for the mysql interface itself.</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>2</td>
<td>Writing External Programs</td>
<td>Recipes for MySQL-based programs.</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>3</td>
<td>Record Selection</td>
<td>Recipes for formatting MySQL output.</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>4</td>
<td>Working with Strings</td>
<td>Recipes for controlling strings in output.</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>5</td>
<td>Working with Dates and Times</td>
<td>Recipes for controlling date/time in output.</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>6</td>
<td>Sorting Query Results</td>
<td>Recipes for sorting SELECT results.</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>7</td>
<td>Generating Summaries</td>
<td>Recipes for using groupings, count(), min and max.</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>8</td>
<td>Introduction to MySQL on the Web</td>
<td>Formatting MySQL output into web pages.</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>9</td>
<td>Modifying Tables</td>
<td>Recipes for controlling strings in output.</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>1</td>
<td>The Client Program</td>
<td>Recipes for the mysql interface itself.</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>date_format</td>
</tr>
<tr>
<td>2</td>
<td>Writing External Programs</td>
<td>Recipes for MySQL-based programs.</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>date_format</td>
</tr>
<tr>
<td>3</td>
<td>Record Selection</td>
<td>Recipes for formatting MySQL output.</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>date_format</td>
</tr>
<tr>
<td>4</td>
<td>Working with Strings</td>
<td>Recipes for controlling strings in output.</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>date_format</td>
</tr>
<tr>
<td>5</td>
<td>Working with Dates and Times</td>
<td>Recipes for controlling date/time in output.</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>date_format</td>
</tr>
<tr>
<td>6</td>
<td>Sorting Query Results</td>
<td>Recipes for sorting SELECT results.</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>date_format</td>
</tr>
<tr>
<td>7</td>
<td>Generating Summaries</td>
<td>Recipes for using groupings, count(), min and max.</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>date_format</td>
</tr>
<tr>
<td>8</td>
<td>Introduction to MySQL on the Web</td>
<td>Formatting MySQL output into web pages.</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>date_format</td>
</tr>
<tr>
<td>9</td>
<td>Modifying Tables</td>
<td>Recipes for controlling strings in output.</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>date_format</td>
</tr>
<tr>
<td>1</td>
<td>The Client Program</td>
<td>Recipes for the mysql interface itself.</td>
</tr>
<tr>
<td>11</td>
<td>6</td>
<td>ORDER BY</td>
</tr>
<tr>
<td>2</td>
<td>Writing External Programs</td>
<td>Recipes for MySQL-based programs.</td>
</tr>
<tr>
<td>11</td>
<td>6</td>
<td>ORDER BY</td>
</tr>
<tr>
<td>3</td>
<td>Record Selection</td>
<td>Recipes for formatting MySQL output.</td>
</tr>
<tr>
<td>11</td>
<td>6</td>
<td>ORDER BY</td>
</tr>
<tr>
<td>4</td>
<td>Working with Strings</td>
<td>Recipes for controlling strings in output.</td>
</tr>
<tr>
<td>11</td>
<td>6</td>
<td>ORDER BY</td>
</tr>
<tr>
<td>5</td>
<td>Working with Dates and Times</td>
<td>Recipes for controlling date/time in output.</td>
</tr>
<tr>
<td>11</td>
<td>6</td>
<td>ORDER BY</td>
</tr>
<tr>
<td>6</td>
<td>Sorting Query Results</td>
<td>Recipes for sorting SELECT results.</td>
</tr>
<tr>
<td>11</td>
<td>6</td>
<td>ORDER BY</td>
</tr>
<tr>
<td>7</td>
<td>Generating Summaries</td>
<td>Recipes for using groupings, count(), min and max.</td>
</tr>
<tr>
<td>11</td>
<td>6</td>
<td>ORDER BY</td>
</tr>
<tr>
<td>8</td>
<td>Introduction to MySQL on the Web</td>
<td>Formatting MySQL output into web pages.</td>
</tr>
</tbody>
</table>
Whoa, that's a lot of data! What you just did was a simple inner join with something called the cartesian product. Here's what happened: MySQL took the table_of_contents table entries and matched (multiplied) it with each entry in the chapter_topics table. So you got 126 (9 table_of_contents x 14 chapter_topics) entries total.
Usually, this is an undesirable result—why would we want every topic in the book listed over and over again with every chapter? So we’re going to refine our SQL query a little bit. Try this:

```
Type the following command at your MySQL prompt:

mysql> select * from table_of_contents, chapter_topics where
    -> table_of_contents.chapter = chapter_topics.chapter;
```

```
+---------+----------------------------------+-----------------------------------------+----------+---------+------------------+
| chapter | chapter_name                     | chapter_description                    | topic_id | chapter | topic            |
+---------+----------------------------------+-----------------------------------------+----------+---------+------------------+
|       1 | The Client Program               | Recipes for the mysql interface itself. |       1 |       1 | queries          |
|       2 | Writing External Programs        | Recipes for MySQL-based programs.      |       2 |       2 | DBI              |
|       2 | Writing External Programs        | Recipes for MySQL-based programs.      |       3 |       2 | PHP              |
|       3 | Record Selection                 | Recipes for formatting MySQL output.   |       4 |       3 | SELECT           |
|       3 | Record Selection                 | Recipes for formatting MySQL output.   |       5 |       3 | WHERE            |
|       3 | Record Selection                 | Recipes for formatting MySQL output.   |       6 |       3 | LIMIT            |
|       3 | Record Selection                 | Recipes for formatting MySQL output.   |       7 |       3 | temporary tables |
|       4 | Working with Strings             | Recipes for controlling strings in output. |       8 |       4 | pattern matching |
|       5 | Working with Dates and Times     | Recipes for controlling date/time in output. |       9 |       5 | TIMESTAMP        |
|       5 | Working with Dates and Times     | Recipes for controlling date/time in output. |      10 |       5 | date_format      |
|       6 | Sorting Query Results            | Recipes for sorting SELECT results.    |      11 |       6 | ORDER BY         |
|       7 | Generating Summaries             | Recipes for using groupings, count(), min and max. |      12 |       7 | GROUP BY         |
|       8 | Introduction to MySQL on the Web | Formatting MySQL output into web pages.   |      13 |       8 | DBI              |
|       8 | Introduction to MySQL on the Web | Formatting MySQL output into web pages.   |      14 |       8 | HTML             |
+---------+----------------------------------+-----------------------------------------+----------+---------+------------------+
14 rows in set (0.01 sec)
```

mysql>

Now we got rid of the cartesian effect and got a table with 14 rows of entries. This makes a lot more sense—using the `WHERE` clause in our SQL statement, we matched the related `chapter` columns to ensure that the correct topics are listed along with the proper chapters.

**Outer Joins**

Did you notice something missing in the above *inner join* results? One of our `table_of_contents` entries, chapter 9, was not included. Why?
Take another look at our two tables:

```sql
mysql> select * from table_of_contents;
+---------+----------------------------------+-----------------------------------------+-----------+
| chapter | chapter_name                      | chapter_description                    | chapter   |
+---------+----------------------------------+-----------------------------------------+-----------+
|       1 | The Client Program               | Recipes for the mysql interface itself |           |
|       2 | Writing External Programs        | Recipes for MySQL-based programs.      |           |
|       3 | Record Selection                 | Recipes for formatting MySQL output.   |           |
|       4 | Working with Strings             | Recipes for controlling strings in output |         |
|       5 | Working with Dates and Times     | Recipes for controlling date/time in output |       |
|       6 | Sorting Query Results            | Recipes for sorting SELECT results.    |           |
|       7 | Generating Summaries             | Recipes for using groupings, count(), min and max. |       |
|       8 | Introduction to MySQL on the Web | Formatting MySQL output into web pages. |           |
|       9 | Modifying Tables                 | Dropping, adding, and changing columns |           |
+---------+----------------------------------+-----------------------------------------+-----------+
9 rows in set (0.01 sec)
```

Observe the results:

```sql
mysql> select * from chapter_topics;
+----------+---------+------------------+
| topic_id | chapter | topic            |
+----------+---------+------------------+
|        1 |       1 | queries          |
|        2 |       2 | DBI              |
|        3 |       2 | PHP              |
|        4 |       3 | SELECT           |
|        5 |       3 | WHERE            |
|        6 |       3 | LIMIT            |
|        7 |       3 | temporary tables |
|        8 |       4 | pattern matching |
|        9 |       5 | TIMESTAMP        |
|       10 |       5 | date_format      |
|       11 |       6 | ORDER BY         |
|       12 |       7 | GROUP BY         |
|       13 |       8 | DBI              |
|       14 |       8 | HTML             |
+----------+---------+------------------+
14 rows in set (0.00 sec)
```

Looking at our two tables, you can see that there are no entries in `chapter_topics` for chapter 9. Because we used an **inner join** to combine our tables, only the entries which exist in **both** tables are included in the resulting table.

But what if we wanted to include EVERY entry in `table_of_contents`, regardless of the existence of a matching `chapter_topics` entry? This is where **outer joins** come into the picture. Let’s try the outer join:
Ah, there's chapter 9! It's now included in the resulting table, with NULL listed for its nonexistent `chapter_topics` values.

Take another look at this SQL statement:

```
select * from table_of_contents
  left outer join chapter_topics
    ON table_of_contents.chapter = chapter_topics.chapter;
```

While an inner join requires that matching entries exist in both tables in order to include them in the result, a left outer join includes ALL the table entries in the left table - in our case, `table_of_contents` — and then matches entries in the right table — `chapter_topics` — if they exist.

**Note** The word "outer" in `left outer join` is optional — `left join` will also work. By the way, there is also a right outer join in MySQL, but it is not used nearly as much as the `left outer join`.

The important new thing we encounter here is the `ON` keyword. This keyword is similar to the `where` keyword, in that a
condition must follow the keyword **ON** as it would follow the **where** keyword. However, **ON** signifies a condition that is _not required_, whereas **WHERE requires** the condition to be met in order for an entry to be included in the results.

In fact, if you want to _require_ a condition within a **left outer join** statement, you have to add a **WHERE** keyword as well. Let’s see how our statement would change if we wanted to search on a topic:

```
Type the following text at your MySQL prompt:
mysql> select table_of_contents.chapter, table_of_contents.chapter_name from table_of_contents
    -> left outer join chapter_topics on table_of_contents.chapter = chapter_topics.chapter
    WHERE chapter_topics.topic = 'ORDER BY';
+---------+-----------------------+
| chapter | chapter_name          |
+---------+-----------------------+
|       6 | Sorting Query Results |
+---------+-----------------------+
1 row in set (0.00 sec)
mysql>
```

### Aliases

Let’s take another look at the previous SQL statement:

```
OBSERVE:
select table_of_contents.chapter, table_of_contents.chapter_name from table_of_contents
left outer join chapter_topics on table_of_contents.chapter = chapter_topics.chapter
WHERE chapter_topics.topic = 'ORDER BY';
```

Did you notice that every time we mention a **column name** within a join statement, the **table name** is included? This is to ensure no ambiguity in column names—for example, there is a **chapter** column in both the **table_of_contents** and the **chapter_topics** tables. In more complex tables, there are many more chances for duplicate column names, so it's best to get into the habit of specifying the table in every instance.

That being said, there's a way to shorten the statement, using **aliases**.

```
Type the following text at your MySQL prompt:
mysql> select tc.chapter, tc.chapter_name from table_of_contents AS tc
    -> left outer join chapter_topics AS ct on tc.chapter = ct.chapter
    WHERE ct.topic = 'ORDER BY';
+---------+-----------------------+
| chapter | chapter_name          |
+---------+-----------------------+
|       6 | Sorting Query Results |
+---------+-----------------------+
1 row in set (0.00 sec)
mysql>
```

Here, through the **AS** keyword, we’ve simply renamed the **table_of_contents** table to **tc** and the **chapter_topics** table to **ct**. Why did we do this? Simply put, renaming the tables with **aliases** does both you and your poor keyboard a favor.

Congratulations, you now know the groundwork of relational databasing! Relational databasing can get tricky sometimes, so try playing around with the MySQL queries to get a strong grasp of the concept. See you in the next lesson.
Managing Query Results

In this lesson, we'll take a look at two very important and useful SQL tools: grouping and ordering. Sorting through query output with **group by** and **order by** is simple in SQL, yet it performs a powerful service - it makes a programmer's life easier by preventing the need for processing query results after the fact.

Make sure you’re connected to your MySQL database, and let’s get started!

**Group By**

Let’s take another look at our two tables, *table_of_contents* and *chapter_topics*, using an **inner join**.

```sql
mysql> select * from table_of_contents as tc, chapter_topics as ct where
    -> tc.chapter = ct.chapter;
```

<table>
<thead>
<tr>
<th>chapter</th>
<th>chapter_name</th>
<th>chapter_description</th>
</tr>
</thead>
<tbody>
<tr>
<td>topic_id</td>
<td>chapter</td>
<td>topic</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>------------------</td>
</tr>
<tr>
<td>1</td>
<td>The Client Program</td>
<td>Recipes for the mysql interface itself.</td>
</tr>
<tr>
<td>1</td>
<td>queries</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Writing External Programs</td>
<td>Recipes for MySQL-based programs.</td>
</tr>
<tr>
<td>2</td>
<td>DBI</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Writing External Programs</td>
<td>Recipes for MySQL-based programs.</td>
</tr>
<tr>
<td>3</td>
<td>PHP</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Record Selection</td>
<td>Recipes for formatting MySQL output.</td>
</tr>
<tr>
<td>4</td>
<td>SELECT</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Record Selection</td>
<td>Recipes for formatting MySQL output.</td>
</tr>
<tr>
<td>3</td>
<td>WHERE</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Record Selection</td>
<td>Recipes for formatting MySQL output.</td>
</tr>
<tr>
<td>6</td>
<td>LIMIT</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Record Selection</td>
<td>Recipes for formatting MySQL output.</td>
</tr>
<tr>
<td>7</td>
<td>temporary tables</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Working with Strings</td>
<td>Recipes for controlling strings in output.</td>
</tr>
<tr>
<td>8</td>
<td>pattern matching</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Working with Dates and Times</td>
<td>Recipes for controlling date/time in output.</td>
</tr>
<tr>
<td>9</td>
<td>TIMESTAMP</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Working with Dates and Times</td>
<td>Recipes for controlling date/time in output.</td>
</tr>
<tr>
<td>10</td>
<td>date_format</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Sorting Query Results</td>
<td>Recipes for sorting SELECT results.</td>
</tr>
<tr>
<td>11</td>
<td>ORDER BY</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Generating Summaries</td>
<td>Recipes for using groupings, count(), min and max.</td>
</tr>
<tr>
<td>12</td>
<td>GROUP BY</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Introduction to MySQL on the Web</td>
<td>Formatting MySQL output into web pages.</td>
</tr>
<tr>
<td>13</td>
<td>DBI</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Introduction to MySQL on the Web</td>
<td>Formatting MySQL output into web pages.</td>
</tr>
<tr>
<td>14</td>
<td>HTML</td>
<td></td>
</tr>
</tbody>
</table>

+---------+----------------------------------+-----------------------------------------+-----------+----------+---------+------------------+
| chapter | chapter_name                     | chapter_description                  |
| topic_id | chapter | topic            |
+---------+----------------------------------+-----------------------------------------+-----------+----------+---------+------------------+
|       1 | The Client Program               | Recipes for the mysql interface itself. |
|        1 | queries |                 |
|       2 | Writing External Programs        | Recipes for MySQL-based programs. |
|        2 | DBI                |                 |
|       2 | Writing External Programs        | Recipes for MySQL-based programs. |
|        3 | PHP                |                 |
|       3 | Record Selection              | Recipes for formatting MySQL output. |
|        4 | SELECT             |                 |
|       3 | Record Selection              | Recipes for formatting MySQL output. |
|        5 | WHERE              |                 |
|       3 | Record Selection              | Recipes for formatting MySQL output. |
|        6 | LIMIT              |                 |
|       3 | Record Selection              | Recipes for formatting MySQL output. |
|        7 | temporary tables        |                 |
|       4 | Working with Strings         | Recipes for controlling strings in output. |
|        8 | pattern matching |                 |
|       5 | Working with Dates and Times | Recipes for controlling date/time in output. |
|        9 | TIMESTAMP           |                 |
|       5 | Working with Dates and Times | Recipes for controlling date/time in output. |
|       10 | date_format         |                 |
|       6 | Sorting Query Results   | Recipes for sorting SELECT results. |
|        11 | ORDER BY           |                 |
|       7 | Generating Summaries    | Recipes for using groupings, count(), min and max. |
|        12 | GROUP BY          |                 |
|       8 | Introduction to MySQL on the Web | Formatting MySQL output into web pages. |
|        13 | DBI                |                 |
|       8 | Introduction to MySQL on the Web | Formatting MySQL output into web pages. |
|        14 | HTML               |                 |

14 rows in set (0.00 sec)

Just like before, we got an entry for every chapter topic that exists - 14 results total. But what if we wanted to gather only a **sampling** of results; say, one topic per chapter? Here’s where **group by** comes in handy.
Type the following text at your MySQL prompt:

```sql
mysql> select * from table_of_contents as tc, chapter_topics as ct where -> tc.chapter = ct.chapter GROUP BY tc.chapter;
```

<table>
<thead>
<tr>
<th>chapter</th>
<th>chapter_name</th>
<th>chapter_description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>The Client Program</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Writing External Programs</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>Record Selection</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Working with Strings</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>Working with Dates and Times</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Sorting Query Results</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>Generating Summaries</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>Introduction to MySQL on the Web</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

8 rows in set (0.00 sec)

mysql>

**Note** Remember, when we use an **inner join**, chapter 9 isn't included—it doesn't have a matching result in `chapter_topics`.

This time, we got only 8 results. That's because, by grouping the results by chapter, we get only one result per chapter. If there is more than one entry in `chapter_topics` for a chapter, SQL simply grabs the first one it finds.

Using **group by** on a certain column, we can ensure that only distinct results are returned for that column, no matter how many times it's repeated normally.

### Using Distinct to Prevent Duplicate Results

Turns out, if you simply want to prevent duplicate results in a query for a certain column, you don't necessarily have to use **group by**.

For instance, perhaps we only care about getting the chapter names for all of the chapters that contain any topics at all, but we don't necessarily care about seeing any of the topics. For this purpose, we could still use **group by**. Or, we could use a keyword called **DISTINCT**:
Type the following text at your MySQL prompt:

```
mysql> select DISTINCT tc.chapter, tc.chapter_name from table_of_contents as tc, chapter_topics as ct where
    -> tc.chapter = ct.chapter;
+-----------------+----------------------------------+
| chapter | chapter_name                  |
+-----------------+----------------------------------+
|       1 | The Client Program               |
|       2 | Writing External Programs        |
|       3 | Record Selection                 |
|       4 | Working with Strings             |
|       5 | Working with Dates and Times     |
|       6 | Sorting Query Results            |
|       7 | Generating Summaries             |
|       8 | Introduction to MySQL on the Web |
+-----------------+----------------------------------+
8 rows in set (0.00 sec)
```

Although **DISTINCT** is a rather limiting keyword, it has its purpose in SQL, making it a worthy weapon in your arsenal of commands. Here's the MySQL manual page on **DISTINCT**.

## Searching and Counting within Groups

Now, distinct results aren't the only reason to use **group by**. This keyword is useful anytime we want to find statistics within column groups.

For instance, what if we want to find out just how many topics are listed under each chapter in our online book?

Type the following text at your MySQL prompt:

```
mysql> select tc.chapter,tc.chapter_name AS Title,count(ct.topic) from table_of_contents as tc LEFT OUTER JOIN chapter_topics as ct
    -> ON tc.chapter = ct.chapter GROUP BY tc.chapter;
+---------+----------+-----------------+
| chapter | chapter_name | count(ct.topic) |
+---------+----------+-----------------+
|       1 | The Client Program |               1 |
|       2 | Writing External Programs |               2 |
|       3 | Record Selection |               4 |
|       4 | Working with Strings |               1 |
|       5 | Working with Dates and Times |               2 |
|       6 | Sorting Query Results |               1 |
|       7 | Generating Summaries |               1 |
|       8 | Introduction to MySQL on the Web |               2 |
|       9 | Modifying Tables |               0 |
+---------+----------+-----------------+
9 rows in set (0.00 sec)
```

Now that's handy. Simply by using the **group by** keyword on the column **tc.chapter**, along with the **group by** function **count()**, we're able to see exactly how many topics go along with each chapter in our book.

There are many more **group by** functions, such as **sum()**, **avg()**, **min()** and **max()**. See MySQL's web site for additional **group by** functions. Be sure to take some time and experiment with these!

## Searching Within Groups

OK, let's up the ante here. Say we're looking for ONLY the chapters which contain 2 or more topics. Let's give it a try with the **WHERE** keyword:
Type the following text at your MySQL prompt:

```sql
mysql> select tc.chapter,tc.chapter_name AS Title,count(ct.topic) from table_of_contents as tc LEFT OUTER JOIN chapter_topics as ct
   -> ON tc.chapter = ct.chapter WHERE count(ct.topic) >= 2 GROUP BY tc.chapter
;
ERROR 1111: Invalid use of group function
mysql>
```

Uh oh, that's not good. You see, the keyword `WHERE` cannot be used with a group column function, like `count()`. This is because group columns depend on `GROUP BY`. Fortunately, there's a different keyword to use: `HAVING`.

Type the following text at your MySQL prompt:

```sql
mysql> select tc.chapter,tc.chapter_name AS Title,count(ct.topic) from table_of_contents as tc LEFT OUTER JOIN chapter_topics as ct
   -> ON tc.chapter = ct.chapter GROUP BY tc.chapter HAVING count(ct.topic) >= 2 ;
+---------+----------------------------------+-----------------+
<table>
<thead>
<tr>
<th>chapter</th>
<th>chapter_name</th>
<th>count(ct.topic)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Writing External Programs</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Record Selection</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>Working with Dates and Times</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>Introduction to MySQL on the Web</td>
<td>2</td>
</tr>
</tbody>
</table>
+---------+----------------------------------+-----------------+
4 rows in set (0.00 sec)
mysql>
```

That's more like it.

**Note**  Notice that the `HAVING` keyword phrase comes after the `GROUP BY` keyword. This syntax order is essential.

### Renaming Results

We've already talked about renaming tables, condensing their names and giving our carpal tunneled hands a rest by shortening our queries. But did you know that we can even rename our query results? While this won’t shorten our queries, it will provide our result columns with more intuitive labels.

For instance, `count(ct.topic)` doesn’t mean much to someone simply looking at our query results. Perhaps a name like `number_of_topics` would make more sense.
That column alias not only looks nicer, it will come in handy when it's time to program in PHP. Plus, aliases can be used in group searches, like this:

Type the following text at your MySQL prompt:

```sql
mysql> select tc.chapter,tc.chapter_name AS Title,count(ct.topic) AS number_of_topics
    -> from table_of_contents as tc LEFT OUTER JOIN chapter_topics as ct
    -> ON tc.chapter = ct.chapter GROUP BY tc.chapter HAVING number_of_topics >= 1;
```

<table>
<thead>
<tr>
<th>chapter</th>
<th>chapter_name</th>
<th>number_of_topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The Client Program</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Writing External Programs</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Record Selection</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>Working with Strings</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>Working with Dates and Times</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>Sorting Query Results</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>Generating Summaries</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>Introduction to MySQL on the Web</td>
<td>2</td>
</tr>
</tbody>
</table>

8 rows in set (0.01 sec)

mysql>

And guess what? Column aliases aren't even limited to group columns. Let's have a little fun:
Type the following text at your MySQL prompt:

```
mysql> select tc.chapter, tc.chapter_name AS Title, count(ct.topic) AS number_of_topics
    -> from table_of_contents as tc
    -> LEFT OUTER JOIN chapter_topics as ct
    -> ON tc.chapter = ct.chapter WHERE tc.chapter_name like ('Working%')
    -> GROUP BY tc.chapter HAVING number_of_topics >= 1;
```

| chapter | Title                        | number_of_topics |
|---------+------------------------------+------------------|
| 4       | Working with Strings         | 1                |
| 5       | Working with Dates and Times | 2                |

2 rows in set (0.00 sec)

```mysql>
```

**Note** We couldn't use the column alias in the WHERE clause. Any ideas why? Hint: it has to do with the order in which SQL performs tasks.

### Ordering Results

Let's say we want to list our book's index, just like we would at the back of a physical book. Of course, an index is useless unless it's sorted alphabetically.

Let's give it a try:

```
mysql> select topic, chapter from chapter_topics
    -> ORDER BY topic;
```

<table>
<thead>
<tr>
<th>topic</th>
<th>chapter</th>
</tr>
</thead>
<tbody>
<tr>
<td>date_format</td>
<td>5</td>
</tr>
<tr>
<td>DBI</td>
<td>2</td>
</tr>
<tr>
<td>DBI</td>
<td>8</td>
</tr>
<tr>
<td>GROUP BY</td>
<td>7</td>
</tr>
<tr>
<td>HTML</td>
<td>8</td>
</tr>
<tr>
<td>LIMIT</td>
<td>3</td>
</tr>
<tr>
<td>ORDER BY</td>
<td>6</td>
</tr>
<tr>
<td>pattern matching</td>
<td>4</td>
</tr>
<tr>
<td>PHP</td>
<td>2</td>
</tr>
<tr>
<td>queries</td>
<td>1</td>
</tr>
<tr>
<td>SELECT</td>
<td>3</td>
</tr>
<tr>
<td>temporary tables</td>
<td>3</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>5</td>
</tr>
<tr>
<td>WHERE</td>
<td>3</td>
</tr>
</tbody>
</table>

14 rows in set (0.01 sec)

```

This seems simple enough. When we used the ORDER BY clause on the topic column, the results were automatically sorted. SQL's ordering of a column depends on its type, so since topic is of a string type, SQL knew to sort in alphabetical order, rather than numerically or by date.

### Descending Results

And if we wanted to reverse the order of our results?
As you can see, we can list the results in *descending* order simply by following the `ORDER BY` with the keyword `DESC`.

**Nesting Groups and Orders**

The `ORDER BY` clause can even be used in conjunction with `GROUP BY`. Let’s revisit our outer join from above:

```sql
mysql> select tc.chapter, tc.chapter_name AS Title, count(ct.topic) AS number_of_topics
    -> from table_of_contents as tc LEFT OUTER JOIN chapter_topics as ct
    -> ON tc.chapter = ct.chapter GROUP BY tc.chapter HAVING number_of_topics >= 1
    -> ORDER BY number_of_topics;
+---------+----------------------------------+------------------+
<table>
<thead>
<tr>
<th>chapter</th>
<th>chapter_name</th>
<th>number_of_topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Working with Strings</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>Sorting Query Results</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>Generating Summaries</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>The Client Program</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>Working with Dates and Times</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>Introduction to MySQL on the Web</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Writing External Programs</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Record Selection</td>
<td>4</td>
</tr>
</tbody>
</table>
+---------+----------------------------------|------------------+
8 rows in set (0.01 sec)
```

In case you haven’t noticed, the way we order our SQL syntax is really important! The `ORDER BY` clause must go *after* the `GROUP BY` clause.
As you can see, we were able to order by the group column alias `number_of_topics`. The nesting of these two clauses creates numerous powerful possibilities in SQL.

In fact, nesting SQL statements, in general, allows for scores of query options, while keeping the general syntax quite simple. For instance, although we’re ordering by `number_of_topics` above, we might want to sub-order the `chapter_name` column—that is, order the chapter names alphabetically within the group of four entries that has 1 topic, and so on.

Type the following text at your MySQL prompt:

```
mysql> select tc.chapter, tc.chapter_name AS Title, count(ct.topic) AS number_of_topics
  -> from table_of_contents as tc LEFT OUTER JOIN chapter_topics as ct
  -> ON tc.chapter = ct.chapter GROUP BY tc.chapter HAVING number_of_topics >= 1
  -> ORDER BY number_of_topics, tc.chapter_name;
```

```
+---------+----------------------------------+------------------+
| chapter | chapter_name                     | number_of_topics |
+---------+----------------------------------+------------------+
|       7 | Generating Summaries             |                1 |
|       6 | Sorting Query Results            |                1 |
|       1 | The Client Program               |                1 |
|       4 | Working with Strings             |                1 |
|       8 | Introduction to MySQL on the Web |                2 |
|       5 | Working with Dates and Times     |                2 |
|       2 | Writing External Programs        |                2 |
|       3 | Record Selection                 |                4 |
+---------+----------------------------------+------------------+
8 rows in set (0.00 sec)
```

See how that works? `number_of_topics` was sorted first, but in the case where this column’s value was the same, SQL sub-sorted the `chapter_name` column.

**Note** Hey, you can also nest `GROUP BY` columns, multiple `JOIN` tables, and `WHERE` clauses. As your SQL statements become more complex, there’s no end to how much nesting you can do to refine your results.

**Limiting Results**

There’s one more major way you can control how SQL outputs query results: the `LIMIT` keyword.

```
mysql> select chapter, topic from chapter_topics
  -> ORDER BY chapter LIMIT 0, 5;
```

```
+---------+---------+
| chapter | topic   |
+---------+---------+
|       1 | queries |
|       2 | DBI     |
|       2 | PHP     |
|       3 | SELECT  |
|       3 | WHERE   |
+---------+---------+
5 rows in set (0.00 sec)
```

Here, we were able to limit our query results to list only the first five rows, using the `LIMIT` keyword at the end of our
query. But what do the numbers 0, 5 signify?

The LIMIT clause can have the following formats:

```
OBSERVE:

LIMIT x[, y]
```

With two arguments, \( x \) represents the desired starting row in your results (the offset), and \( y \) represents the number of rows to display. This is why LIMIT 0, 5 listed 5 rows, beginning with the first row. LIMIT 5, 8 would list 8 rows, starting with the 6th row.

```
OBSERVE:

LIMIT x
```

With one argument, \( x \) represents the number of resulting rows to display. It will return the rows starting from the first row in the results.

So if we wanted to find the chapter having the most topics?

Type the following text at your MySQL prompt:

```
mysql> select tc.chapter, tc.chapter_name AS Title, count(ct.topic) AS number_of_topics
                    -> from table_of_contents as tc LEFT OUTER JOIN chapter_topics as ct
                    -> ON tc.chapter = ct.chapter GROUP BY tc.chapter HAVING number_of_topics >= 1
                    -> ORDER BY number_of_topics DESC, tc.chapter_name LIMIT 0, 1;
```

```
+---------+------------------+------------------+
| chapter | chapter_name     | number_of_topics |
+---------+------------------+------------------+
| 3       | Record Selection | 4                |
+---------+------------------+------------------+
1 row in set (0.00 sec)
mysql>
```

Wow, we’ve covered a lot of ground! It’s time to start applying our new SQL skills. Be sure to practice, and hand in your assignments. See you in the next lesson!
The PHP/MySQL Relationship

While almost all programming languages can connect to and exchange information with an SQL database, PHP has a special relationship with this language—especially through the MySQLi DBMS (The "i" stands for "improved"). In fact, PHP and MySQLi work so well together on Linux Apache systems, the term LAMP Stack has been coined in the software development world to describe this important set of technologies.

You see, PHP comes with a MySQLi extension built right into the language, as well as an embedded MySQLi library. This means that we can simply use built-in PHP functions to access our SQL databases on MySQL. For more information, check out the Wikipedia page on PHP, or the PHP manual on the MySQLi extension.

With a good grasp of HTML, PHP, and SQL, you can give your website the power to utilize SQL databases. Let's get started!

Connecting to a Database in PHP

We recommend creating a folder for your program files for this course. In the File Browser panel at left, right-click the Home folder and select New folder.... Name the folder phpsql1 or a similar name to describe this course.

In addition to accessing the Unix Terminal and MySQL shell, the CodeRunner Editor is a multi-purpose Integrated Development Environment (IDE) that can be used for editing HTML and PHP, along with several other languages. To change the syntax to PHP, click on the Syntax dropdown menu and select PHP from the list of programming languages.

Now we can start experimenting with the built-in MySQLi functions within PHP. Let's start by connecting to our database. Enter the following code in the Code Editor area below:
```php
<?php
$host = "sql.useractive.com"; // This is the server where your database resides
$user = "username"; // This should be your OST username
$pw = "password"; // This should be your OST password
$database = "username"; // Your database name is the same as your OST username

$db = new mysqli($host,$user,$pw,$database)
or die("Cannot connect to MySQL.");

echo "Success!  Connected to database ".$database;

$db->close();
?>
```

**Note**  Remember, we created your database, and it has the same name as your OST username. Be sure to replace `username` with your Sandbox login, and `password` with your password.

- Save it in the `phpsql1` folder as `connect.php` and click Preview (·):

**Success! Connected to database certjosh**

MySQL functions follow the naming convention `mysqli_*()`, where the `asterisk` represents the function name. The `mysqli_connect()` function is an exception to this rule. It is a constructor meaning it is used to create new objects. Constructors have unique naming conventions, and so this function appears as `mysqli()` in our code.

PHP allows the MySQLi functions to be used with two different syntaxes: object-oriented and procedural.

For this course, we will use the object-oriented style, because it is generally more popular among real-world programmers. Keep in mind throughout the lessons that the function names may be introduced as `mysqli_close()`, but when used in the code, will appear as `$variable->close()`.

**Note**

Object-oriented style:

```php
$db->close();
```

Procedural style:

```php
mysqli_close($db);
```

Let's take another look at the code.

```php
$db = new mysqli($host,$user,$pw,$database)
or die("Cannot connect to MySQL.");

$db->close();
```

Here, we use two built-in MySQLi functions in PHP. The first, `mysqli_connect()` (see PHP.net), is a constructor that returns a resource called a link identifier to the MySQL server. We store this value in the `$db` variable, so we can use it to perform SQL tasks. Because the function is a constructor, we use the keyword `new` to indicate that we are creating a new connection. The `mysqli_connect()` function is how we connect to a MySQL server. To use it, we pass in four parameters: `$host`, which in our case happens to be "sql.useractive.com"; `$user`, which is our OST username; `$password`, our password; and of course, `$database`, the database to which we are connecting. Essentially, `mysqli_connect()` performs the exact same function as the Linux/Unix command we use to connect and the command we use to select our database:
OBSERVE:

cold:~$ mysql -h sql.useractive.com -u yourlogin -p password
mysql> use username;

Note If the connection doesn't work for any reason, the function die() simply exits the program with an error message. For more information, see PHP.net.

The second function, mysqli_close() (see PHP.net), simply closes the connection to the MySQL server, much as the SQL exit command would.

**Executing SQL Commands**

Now let's try using PHP to retrieve information from our friends table.

```php
<?php
$host = "sql.useractive.com"; // This is the server where your database resides
$user = "username"; // This should be your OST username
$pw = "password"; // This should be your OST password
$database = "username"; // Your database name is the same as your OST username

$db = new mysqli($host,$user,$pw,$database)
or die("Cannot connect to MySQL.");

echo "Success! Connected to database ".$database;

$command = "SELECT * FROM friends;";
echo "Command: ".$command."<br;";

$result = $db->query($command);
echo "Result: ".$result."<br;";

$db->close();
?>
```

and to see what happens.

Command: SELECT * FROM friends;

Catchable fatal error: Object of class mysqli_result could not be converted to string in /users/certjosh/lesson08.php on line 15

OK, the command looks familiar. But what's with the result? Let's take another look:

```php
$result = $db->query($command);
echo "Result: ",$result."<br;";
```

As you may have already guessed, mysqli_query() performs the query passed in as $command, just as it would in the MySQL shell. However, it does not return a text-based table as we would expect. Instead, mysqli_query() returns a resource, much like mysqli_connect() does. We've stored this resource in a variable called $result. We tried to use echo to print out the value of the resource, but it yields the error we saw above. Fortunately, we can access this resource through other means.
<?php

$host = "sql.useractive.com"; // This is the server where your database resides
$user = "username";       // This should be your OST username
$pw = "password";         // This should be your OST password
$database = "username";   // Your database name is the same as your OST username

$db = new mysqli($host,$user,$pw,$database)
or die("Cannot connect to MySQL.");

$command = "SELECT * first_name,last_name FROM friends;";
    echo "Command: "$command."<br>";

$result = $db->query($command);
    echo "Result: "$result."<br>";

if ($result = $db->query($command)) {
    while ($data = $result->fetch_object()) {
        echo $data->first_name." ";
        echo $data->last_name."<br>";
    }
    $result->free();
}

$db->close();
?>

and to see what happens.

Hoyoul Kang
Trish Gray
Kerry Beck
Dessie Coale
Kerry Parmelee

Now, those are tangible results! How were we able to convert that "Object of class mysqli_result" into real data? Take a look:

```php
if ($result = $db->query($command)) {
    while ($data = $result->fetch_object()) {
        echo $data->first_name." ";
        echo $data->last_name."<br>";
    }
    $result->free();
}
```

Here we've used one more built-in function, `mysqli_fetch_object()` (see PHP.net). This function is special in that it takes in the resource we called $result and returns an object that we call $data. This PHP object contains within it properties that correspond to each row's column names—in our case, `first_name` and `last_name`. We can access these properties through the operator -> to get strings that we can print out. After calling `mysqli_fetch_object()` on a resource, when we are finished using the resource, we use `mysqli_result_free()` to free the memory that was storing it.
We only have to free the memory if the command was one that would have returned a table (SELECT, SHOW, DESCRIBE, or EXPLAIN). If entering the command into a MySQL shell would have caused a table to be printed out, then that data is being stored in the PHP variable when we make the same query from PHP. However, commands that perform an action without returning anything (INSERT, DELETE, etc.) do not store values in the variables, so we don’t need to use mysqli_result::free() in those instances. For more information see the php.net documentation.

Delimiting Queries

Now let’s try using mysqli_query() to insert a new friend into our table:

```php
<?php
$host = "sql.useractive.com"; // This is the server where your database resides
$user = "username"; // This should be your OST username
$pw = "password"; // This should be your OST password
$database = "username"; // Your database name is the same as your OST username

$db = new mysqli($host,$user,$pw,$database)
or die("Cannot connect to MySQL.");

$new_first_name = "Tim";
$new_last_name = "O'Reilly";
$command = "INSERT INTO friends (first_name, last_name) VALUES ('" . $new_first_name."','".$new_last_name."');";
$result = $db->query($command);

$command = "SELECT first_name,last_name FROM friends;";
if ($result = $db->query($command)) {
  while ($data = $result->fetch_object()) {
    echo $data->first_name." ";
    echo $data->last_name."<br>;"
  }
  $result->free();
}

$db->close();
?>
```

Note The `(first_name, last_name)` code in the SQL command allows us to insert only the specified columns. The remaining columns simply get their default values.

Since an `INSERT INTO` command doesn’t return any rows, we don’t need to do anything beyond `mysqli_query()` to execute the command.

Now and to see our new friend:

- Hoyoul Kang
- Trish Gray
- Kerry Beck
- Dessie Coale
- Kerry Parmelee

Hey, where’s our new friend? Obviously something has gone wrong. Time to debug:
<?php
$host = "sql.useractive.com"; // This is the server where your database resides
$user = "username"; // This should be your OST username
$pw = "password"; // This should be your OST password
$database = "username"; // Your database name is the same as your OST username

$db = new mysqli($host,$user,$pw,$database)
or die("Cannot connect to MySQL.");

$new_first_name = "Tim";
$new_last_name = "O'Reilly";
$command = "INSERT INTO friends (first_name, last_name) VALUES ('".
    "$new_first_name."','".".$new_last_name."');";

echo "Command = ",$command."<br>;

$result = $db->query($command);

$command = "SELECT first_name,last_name FROM friends;";
if ($result = $db->query($command)) {
    while ($data = $result->fetch_object()) {
        echo $data->first_name." ";
        echo $data->last_name."<br>;
    }
    $result->free();
}
$db->close();
?>

and to find out what's going on:

Command = INSERT INTO friends (first_name, last_name) VALUES ('Tim','O'Reilly');
Hoyoul Kang
Trish Gray
Kerry Beck
Dessie Coale
Kerr Parmelee

Hmm. Something looks fishy with that SQL command. Switch back over to your MySQL shell and type that code in:

INTERACTIVE SESSION:

mysql> INSERT INTO friends (first_name, last_name) VALUES ('Tim','O'Reilly');

Uh oh, that's not good. MySQL is telling us that we never completed a single quote ('), and looking at our command, we can see why—"O'Reilly" actually contains an apostrophe—a single quote! We need a way to handle unexpected quotes in our SQL commands so that kind of thing doesn't happen.
Ah, there's our new friend Tim. How did we fix the problem? Using the built-in PHP function `mysqli_real_escape_string()` on our input data, we converted all the single quotes ('') to backslash single quotes ('\'). Doing this helps us to avoid errors we might otherwise encounter in the MySQL command line—it's called *delimiting*. Any time we insert or update data of type string into a table, it makes sense to use `mysqli_real_escape_string()` to delimit the data as a precaution.

In the next few lessons, we'll see how to use more SQL statements to make web-based applications in PHP.

Be sure to hand in your assignments; see you in the next lesson!
Project Address/Phone Book, Part 1

Project Description

Now that we’ve gone over the basics of the PHP MySQL extension, we'll make an online web-based address/phone book. With an address book on your website, you can access your phone numbers and addresses anywhere in the world so long as you have Internet access. We'll also make an HTML form so that you or your friends can go online and enter information into the address book.

We will construct an HTML form on a page named **addentry.html** which submits data to a PHP program called **addentry.php**. The **addentry.php** program will in turn store the data into the MySQL database, as long as the data is valid. Next, we’ll make **viewbook.php**, which will retrieve MySQL database entries and display the data we’ve stored, allowing us to view our address book. Here is a general diagram to help you visualize how it will work:

![Diagram of address book process](image)

Table Layout

Before we jump into making the HTML forms and PHP programs, we'll need a standard database layout/description. What kind of data would you hold in an address/phone book? This is where many database designs go awry. More often than not, a good database is the result of lot of good database design and relatively little actual programming. Let's take a look at our **addressbook** table design.

<table>
<thead>
<tr>
<th>Data Name</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>int unsigned NOT NULL auto_increment</td>
</tr>
<tr>
<td>first_name</td>
<td>varchar(20)</td>
</tr>
<tr>
<td>last_name</td>
<td>varchar(20)</td>
</tr>
<tr>
<td>phone</td>
<td>varchar(15)</td>
</tr>
<tr>
<td>address</td>
<td>varchar(100)</td>
</tr>
<tr>
<td>city</td>
<td>varchar(20)</td>
</tr>
<tr>
<td>state</td>
<td>char(2)</td>
</tr>
<tr>
<td>zipcode</td>
<td>int(5) unsigned</td>
</tr>
<tr>
<td>birthday</td>
<td>date</td>
</tr>
</tbody>
</table>

Table Creation
With the data names and types in mind, let's create a table in your MySQL shell and name it `addressbook`.

Type the following commands at your MySQL prompt:

```sql
mysql> create table addressbook (id int unsigned NOT NULL auto_increment, 
-> first_name varchar(20), last_name varchar(20), phone varchar(15), 
-> address varchar(100), city varchar(20), state char(2), 
-> zipcode int(5) unsigned, birthday date, primary key(id));
Query OK, 0 rows affected (0.00 sec)
```

```sql
mysql> describe addressbook;
```

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Key</th>
<th>Default</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>int(10) unsigned</td>
<td>NO</td>
<td>PRI</td>
<td>NULL</td>
<td>auto_increment</td>
</tr>
<tr>
<td>first_name</td>
<td>varchar(20)</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>last_name</td>
<td>varchar(20)</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>phone</td>
<td>varchar(15)</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>address</td>
<td>varchar(100)</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>city</td>
<td>varchar(20)</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>state</td>
<td>char(2)</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>zipcode</td>
<td>int(5) unsigned</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>birthday</td>
<td>date</td>
<td>YES</td>
<td></td>
<td>NULL</td>
<td></td>
</tr>
</tbody>
</table>

9 rows in set (0.00 sec)

The phone number was set as `varchar(15)` instead of `int(15)` for two reasons: First, by doing this, people are able to enter dashes and parentheses in the form, both of which would be unacceptable for an integer data type in SQL. Second, by using `varchar` instead of `int`, people who have phone numbers that use text to allow easy memorization like "1-800-CONTACT" will be able to enter them as well.

Once you're done setting up the table, you're done with this leg of our project. As always, be sure to hand in your assignments! See you in the next lesson.

---

**Copyright © 1998-2014 O'Reilly Media, Inc.**

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License. See [http://creativecommons.org/licenses/by-sa/3.0/legalcode](http://creativecommons.org/licenses/by-sa/3.0/legalcode) for more information.
Starting with the HTML Form

Before we get started on this lesson, be sure you've made the addressbook table in the last lesson.

Now we can get started on the PHP programming part of the project. Our PHP script needs to allow web users to enter data into the addressbook table that we just set up. But first, we'll need an HTML form to provide inputs for the PHP script to process and store.

Set the CodeRunner Editor syntax to HTML. We'll be creating both HTML and PHP files for this project.

Type the following code, with the syntax set to HTML:

```
<html>
    <head><title>This the Address Book Form</title></head>
    <body>
        <h1>Welcome to my address book!</h1>
        <form action="addentry.php" method="POST">
            First Name: <input type="text" name="first_name" size="20"><br>
            Last Name: <input type="text" name="last_name" size="20"><br>
            Phone Number: <input type="text" name="phone" size="15"><br>
            Address: <input type="text" name="address" size="20"><br>
            City: <input type="text" name="city" size="20"><br>
            State: <input type="text" name="state" size="2"><br>
            Zip Code: <input type="text" name="zipcode" size="5"><br>
            Birthday (MM-DD-YYYY):<br>
            <input type="text" name="month" size="2">-
            <input type="text" name="day" size="2">-
            <input type="text" name="year" size="4"><br>
            <input type="submit" value="Submit">  
        </form>
    </body>
</html>
```

Save it in your /phpsql1 folder as addentry.html, and click:

Welcome to my address book!

First Name: 
Last Name: 
Phone Number: 
Address: 
City: 
State: 
Zip Code: 
Birthday (MM-DD-YYYY): - -  

Notice that the entry fields match the columns we created for our addressbook table in SQL. Setting up the elements of our forms and tables to correspond with each other makes it easier to drop the information we want into our database.
Storing the Data into SQL through PHP

Now that we have our HTML form ready to submit our data, let's work on the PHP script that will put the data into the MySQL database for us.

We'll organize the PHP script into several sub-components, breaking the program down into the following:

- A functions area
- A user variables section
- A main program body

The main reason we divide the script into these sections is to make it easier for others to edit, read, and ultimately, reuse our code.

Start a new file and switch the CodeRunner Editor syntax to PHP.
<?php

#----------------------#
# Functions            #
#----------------------#

function check_input ($form_array) {
    if ($form_array['first_name'] && $form_array['last_name'] &&
        $form_array['phone'] && $form_array['address'] &&
        $form_array['city'] && $form_array['state'] &&
        $form_array['zipcode'] && $form_array['month'] &&
        $form_array['day'] && $form_array['year']) {
        return 1;
    } else return 0;
}

function get_birthday ($form_array) {
    $birthday = $form_array['year']."-".$form_array['month']."-".$form_array['day'];
    return $birthday;
}

#----------------------#
# User Variables       #
#----------------------#
$host = "sql.useractive.com"; // This is the server where your database resides
$user = "username"; // This should be your OST username
$pw = "password"; // This should be your OST password
$database = "yourlogin"; // Your database is the same as your OST username
$table_name = "addressbook";

#----------------------#
# Main Body            #
#----------------------#
if (check_input($_POST)) {
    $birthday = get_birthday($_POST);
    $db = new mysqli($host,$user,$pw,$database)
        or die("Cannot connect to MySQL.");

    $command = '"insert into $table_name
    values(\"\',\".$db->real_escape_string($_POST['first_name'])."\","
    \
    \
    \
    \
    \
    \
    \".$db->real_escape_string($_POST['last_name'])."\","
    \
    \
    \
    \
    \
    \
    \".$db->real_escape_string($_POST['phone'])."\","
    \
    \
    \
    \
    \
    \
    \".$db->real_escape_string($_POST['address'])."\","
    \
    \
    \
    \
    \
    \
    \".$db->real_escape_string($_POST['city'])."\","
    \
    \
    \
    \
    \
    \
    \".$db->real_escape_string($_POST['state'])."\","
    \
    \
    \
    \
    \
    \
    \".$db->real_escape_string($_POST['zipcode'])."\","
    \
    \
    \
    \
    \
    \
    \".$db->real_escape_string($birthday)."\")");
    $result = $db->query($command);

    print "Data was successfully entered.<br>");
    $db->close();
} else { print "Data was NOT entered due to errors.<br>"); }

<a href="/viewbook.php">View my address book</a>
Save it in your /phsql1 folder as addentry.php.

In the preview of your addentry.html page, start entering data. Submit the form. You should see a response letting you know whether the data was successfully entered into the database.

For instance, if you entered the following into your form:

Welcome to my address book!

First Name: Trish
Last Name: Gray
Phone Number: 555-555-5555
Address: 555 5th Street
City: Fiveville
State: CA
Zip Code: 55555
Birthday (MM-DD-YYYY): 05-05-1955

Submit

Data was successfully entered.
View my address book

Now, if you check your SQL addressbook table, you should get something like this:

<table>
<thead>
<tr>
<th>id</th>
<th>first_name</th>
<th>last_name</th>
<th>phone</th>
<th>address</th>
<th>city</th>
<th>state</th>
<th>zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Trish</td>
<td>Gray</td>
<td>555-555-5555</td>
<td>555 5th Street</td>
<td>Fiveville</td>
<td>CA</td>
<td>5555</td>
</tr>
</tbody>
</table>

Breaking Down addentry.php

Let’s look more closely at the script. First, let's study the new functions we’ve defined (check_input() and get_birthday()). After that, we’ll go over the main body.

The first function, check_input(), looks like this:
function check_input ($form_array) {
    if ($form_array['first_name'] && $form_array['last_name'] &&
        $form_array['phone'] && $form_array['address'] &&
        $form_array['city'] && $form_array['state'] &&
        $form_array['zipcode'] && $form_array['month'] &&
        $form_array['day'] && $form_array['year']) {
        return 1;
    } else return 0;
}

The check_input() function checks to see whether the user filled out the form completely. It takes in the parameter $form_array, which is assumed to be a superglobal like $_POST. If any of the form fields are left blank, the if statement in the check_input() function would return "false", causing the function to return a zero. However, if all the form fields have been completed, it returns a value of 1.

Note Remember that in PHP a zero or NULL is false in a conditional statement, and anything else is true.

The other function we've defined is get_birthday():

function get_birthday ($form_array) {
    $birthday = $form_array['year']."-".$form_array['month']."-".$form_array['day'];
    return $birthday;
}

Since the dates must be in the format YYYY-MM-DD (Year-Month-Day), the get_birthday function reads the date from the form fields (again, passed into the parameter $form_array), assembles them together into a variable named $birthday, and returns it.

Now let's talk about the main mody of the program. Here it is again:

if (check_input($_POST)) {
    $birthday = get_birthday($_POST);
    $db = new mysqli($host,$user,$pw,$database)
        or die("Cannot connect to MySQL.");
    $command = "insert into $table_name
            values('',"'.".$db->real_escape_string($_POST['first_name'])."',
            '"'.".$db->real_escape_string($_POST['last_name'])."',
            '"'.".$db->real_escape_string($_POST['phone'])."',
            '"'.".$db->real_escape_string($_POST['address'])."',
            '"'.".$db->real_escape_string($_POST['city'])."',
            '"'.".$db->real_escape_string($_POST['state'])."',
            '"'.".$db->real_escape_string($_POST['zipcode'])."',
            '"'.".$db->real_escape_string($birthday)."');";
    $result = $db->query($command);
    print "Data was successfully entered\n";
    $db->close();
} else { print "Data was NOT entered due to errors.\n"; }

The argument of the if statement calls the check_input() function, which will return true if the user has filled everything in or false if she has not filled in everything. If check_input() returns true, we run the mysqli_connect() function to initialize our database connection as before. Then we run the mysqli_query() function to execute the SQL statement stored in the variable $command. We use the mysqli_real_escape_string() function to delimit each of
the form inputs, so that nothing unexpected happens to cause an SQL error.

Now that we're able to enter data into our addressbook from the web, it would be nice to be able to read the things we've put into it. In the next lesson, we'll learn to make viewbook.php, which we'll use to read elements of our database.

Keep up the great work! Hand in your assignments, and see you in the next lesson.

Copyright © 1998-2014 O'Reilly Media, Inc.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License. See http://creativecommons.org/licenses/by-sa/3.0/legalcode for more information.
Construction of viewbook.php

We're already halfway done with our address/phone book. The next part will be easier now that you've made addentry.php. So, what should viewbook.php do? Let's keep it simple and have it list all the entries that are stored in the addressbook table.

To get started, create a new file in the CodeRunner Editor with the syntax set to PHP. Be sure you have saved addentry.php from the last lesson before you go on!

Type the following code with the syntax set to PHP:

```php
<?php
#----------------------#
# User Variables       #
#----------------------#
$host = "sql.useractive.com"; // This is the server where your database resides
$user = "username"; // This should be your OST username
$pw = "password"; // This should be your OST password
$database = "yourlogin"; // Your database is the same as your OST username
$table_name = "addressbook";
#----------------------#
# Main Body            #
#----------------------#
$db = new mysqli($host,$user,$pw,$database)
or die("Cannot connect to MySQL.");
?>;

<h1>My Address Book</h1>
<TABLE BORDER="1">
<TR><TD>Name</TD><TD>Phone Number</TD><TD>Address</TD><TD>Birthday</TD></TR>
<?
$command = "select * from ";$table_name;
$result = $db->query($command);
while ($data = $result->fetch_object()) {
} $result->free();
$db->close;
?>;
</TABLE>
<br>
<a href="/addentry.html">Add a new entry</a>
```

Save it in your /phpsql1 directory as viewbook.php. Now click or call up viewbook.php directly in the web browser, to make sure it works:
Add a new entry:

The only new thing within this code is the while loop we used to display the SQL data, fetched by the mysql_fetch_object() function. Let's take a closer look at the details:

```
OBSERVE:

$command = "select * from ".$table_name;
$result = $db->query($command);
while ($data = $result->fetch_object()) {
    print "<TR>;<TD>".$data->last_name."",".".$data->first_name."</TD>;".$data->phone."</TD>;
    print "<TD>".$data->address."<BR>".$data->city."",".".$data->state."".$data->zipcode."<BR>";</TD>;
    print "<TD>".$data->birthday."</TD>;</TR>;
}
```

Once the mysql_fetch_object() function fetches a row from the $result resource, it automatically moves to the next row, and the next, until it finally hits NULL. This is why we’re able to use the while loop without causing an infinite loop. Pretty handy for tasks like these, huh?

Congratulations, our project is finally complete! See you in the next lesson.

Copyright © 1998-2014 O'Reilly Media, Inc.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License. See http://creativecommons.org/licenses/by-sa/3.0/legalcode for more information.
## Date and Time Functions

We'll cover these functions in this lesson:

<table>
<thead>
<tr>
<th>Date Functions</th>
<th>Time Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAYOFWEEK(date)</td>
<td>CURRENT_TIME()</td>
</tr>
<tr>
<td>WEEKDAY(date)</td>
<td>HOUR(time)</td>
</tr>
<tr>
<td>DAYOFMONTH(date)</td>
<td>MINUTE(time)</td>
</tr>
<tr>
<td>DAYOFYEAR(date)</td>
<td>SECOND(time)</td>
</tr>
<tr>
<td>YEAR(date)</td>
<td>TIME_TO_SEC(time)</td>
</tr>
<tr>
<td>MONTH(date)</td>
<td>SEC_TO_TIME(sec)</td>
</tr>
<tr>
<td>DAYNAME(date)</td>
<td>TIME_FORMAT(time,format)</td>
</tr>
<tr>
<td>MONTHNAME(date)</td>
<td></td>
</tr>
<tr>
<td>QUARTER(date)</td>
<td></td>
</tr>
<tr>
<td>WEEK(date)</td>
<td></td>
</tr>
<tr>
<td>WEEK(date,weekday)</td>
<td></td>
</tr>
<tr>
<td>TO_DAYS(date)</td>
<td>SYSDATE()</td>
</tr>
<tr>
<td>FROM_DAYS(x)</td>
<td>CURRENT_TIMESTAMP</td>
</tr>
<tr>
<td>CURDATE()</td>
<td>UNIX_TIMESTAMP()</td>
</tr>
<tr>
<td>CURRENT_DATE</td>
<td>UNIX_TIMESTAMP(date)</td>
</tr>
<tr>
<td>PERIOD_ADD(period,x)</td>
<td>FROM_UNIXTIME(unix_timestamp)</td>
</tr>
<tr>
<td>PERIOD_DIFF(period1,period2)</td>
<td>FROM_UNIXTIME(unix_timestamp,format)</td>
</tr>
<tr>
<td>DATEDIFF(date1,date2)</td>
<td></td>
</tr>
<tr>
<td>DATE_FORMAT(date,format)</td>
<td></td>
</tr>
</tbody>
</table>

### Date/Time Mathematical Functions

- DATE_ADD(date/time,INTERVAL expr type)
- ADDDATE(date/time,INTERVAL expr type)
- DATE_SUB(date/time,INTERVAL expr type)
- SUBDATE(date/time,INTERVAL expr type)
- EXTRACT(expr type FROM date/time)

We'll explain each of these functions in detail, with examples of its usage, in this lesson. We'll explain the parameters of the functions and the output as well.

### DAYOFWEEK(date)

This function returns the weekday of the particular date in integers that conform with the ODBC standards. 1 = Sunday, 2 = Monday, 3 = Tuesday, etc.
**WEEKDAY(date)**

This function returns the weekday of the particular date in integers. 6 = Sunday, 0 = Monday, 1 = Tuesday, etc.

**DAYOFMONTH(date)**

Returns the day of the month of the date parameter as an integer.

**DAYOFYEAR(date)**

Returns the day of the year of the date parameter, ranging from 1 to 366.

**YEAR(date)**

Returns the year of the date parameter as an integer between 1000 and 9999.
**MONTH(date)**

Returns the month of the **date** parameter as an integer between 1 and 12.

**INTERACTIVE SESSION:**

```
mysql> select month('2019-11-02'), month('2019-09-09');
+---------------------+--------------------+
| month('2019-11-02') | month('2019-09-09')|
|---------------------+--------------------+
|                  11 |                  9 |
+---------------------+--------------------+
```

**DAYNAME(date)**

This function resembles the **weekday()** function, except that instead of returning an integer between 0 and 6, it returns a text string describing the day, like "Monday."

**INTERACTIVE SESSION:**

```
mysql> select dayname('2019-11-02');
+-----------------------+
| dayname('2019-11-02') |
|-----------------------+
| Saturday              |
+-----------------------+
```

**MONTHNAME(date)**

Returns the month name of the **date** parameter as a string between "January" and "December."

**INTERACTIVE SESSION:**

```
mysql> select monthname('2019-11-02');
+-------------------------+
| monthname('2019-11-02') |
|-------------------------+
| November                |
+-------------------------+
```

**QUARTER(date)**

Returns the quarter (as an integer) of the year of the **date** parameter between 1 and 4.
**WEEK(date)**

Returns the week number of the year of the date parameter, with Sunday as the first day of the week. For example, January 1 falls on the first week of the year, and December 31 falls on the 52nd week.

**WEEK(date,weekday)**

In this function, weekday can be either a 1 or a 0. This is identical to the week() function. This function returns the week number of the year of the date parameter with Sunday as the first day of the week if weekday is 0, and Monday as the first day of the week if weekday is 1.

**TO_DAYS(date)**

Converts the date to number of days as an integer.

**FROM_DAYS(x)**

This function is complementary to the to_days() function. Converts the integer x to a date.
INTERACTIVE SESSION:

mysql> select from_days(737728);
+-------------------+
| from_days(737728) |
| 2019-10-31         |
+-------------------+

CURDATE(), CURRENT_DATE

Both these functions return the current date in the standard SQL format of YYYY-MM-DD. Notice that the latter function does not have parentheses.

INTERACTIVE SESSION:

mysql> select curdate(), current_date;
+------------+--------------+
| curdate()  | current_date |
| 2014-11-03 | 2014-11-03   |
+------------+--------------+

PERIOD_ADD(period,x)

Adds x months to period and returns the result in the format of YYYYMM. The period parameter can be in the format of YYYYMM or YYMM.

INTERACTIVE SESSION:

mysql> select period_add(1911,2), period_add(201911,2);
+--------------------+----------------------+
| period_add(1911,2) | period_add(201911,2) |
| 202001            | 202001               |
+--------------------+----------------------+

PERIOD_DIFF(period1,period2)

Returns the difference between the two periods (period1 and period2 in either YYYYMM or YYMM format) as a number of months.

INTERACTIVE SESSION:

mysql> select period_diff(201911,1901);
+--------------------------+
| period_diff(201911,1901) |
| 10                       |
+--------------------------+

DATEDIFF(date1,date2)

Returns the number of days between date1 and date2. date1 and date2 may be either date or date-and-time expressions, including now().
**DATE_FORMAT(date,format)**

Formats the date as specified by the `format` parameter. Here is a list of format items you can use in your format string (info from mysql.com):

<table>
<thead>
<tr>
<th>Format Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>%M - month name (January..December)</td>
</tr>
<tr>
<td>%W - weekday name (Sunday..Saturday)</td>
</tr>
<tr>
<td>%D - day of the month with suffix (1st, 2nd, 3rd, etc.)</td>
</tr>
<tr>
<td>%Y - year, numeric, 4 digits</td>
</tr>
<tr>
<td>%y - year, numeric, 2 digits</td>
</tr>
<tr>
<td>%a - abbreviated weekday name (Sun..Sat)</td>
</tr>
<tr>
<td>%d - day of the month, numeric (00..31)</td>
</tr>
<tr>
<td>%e - day of the month, numeric (0..31)</td>
</tr>
<tr>
<td>%m - month, numeric (01..12)</td>
</tr>
<tr>
<td>%c - month, numeric (1..12)</td>
</tr>
<tr>
<td>%b - abbreviated month name (Jan..Dec)</td>
</tr>
<tr>
<td>%j - day of year (001..366)</td>
</tr>
<tr>
<td>%h - hour (00..23)</td>
</tr>
<tr>
<td>%k - hour (0..23)</td>
</tr>
<tr>
<td>%l - hour (1..12)</td>
</tr>
<tr>
<td>%I - hour (01..12)</td>
</tr>
<tr>
<td>%i - minutes, numeric (00..59)</td>
</tr>
<tr>
<td>%r - time, 12-hour (hh:mm:ss [AP]M)</td>
</tr>
<tr>
<td>%T - time, 24-hour (hh:mm:ss)</td>
</tr>
<tr>
<td>%S - seconds (00..59)</td>
</tr>
<tr>
<td>%s - seconds (00..59)</td>
</tr>
<tr>
<td>%p - AM or PM</td>
</tr>
<tr>
<td>%w - day of the week (0=Sunday..6=Saturday)</td>
</tr>
<tr>
<td>%U - week (0..52), where Sunday is the first day of the week</td>
</tr>
<tr>
<td>%u - week (0..52), where Monday is the first day of the week</td>
</tr>
<tr>
<td>% - the % symbol</td>
</tr>
<tr>
<td>All other characters are just copied to the result without interpretation.</td>
</tr>
</tbody>
</table>

**Date/Time Mathematical Functions**

The next four functions require date/time as well an interval parameters.
### List of formats:

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Definition</th>
<th>Expected Expression Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECOND</td>
<td>Seconds</td>
<td>SECONDS</td>
<td></td>
</tr>
<tr>
<td>MINUTE</td>
<td>Minutes</td>
<td>MINUTES</td>
<td></td>
</tr>
<tr>
<td>HOUR</td>
<td>Hours</td>
<td>HOURS</td>
<td></td>
</tr>
<tr>
<td>DAY</td>
<td>Days</td>
<td>DAYS</td>
<td></td>
</tr>
<tr>
<td>MONTH</td>
<td>Months</td>
<td>MONTHS</td>
<td></td>
</tr>
<tr>
<td>YEAR</td>
<td>Years</td>
<td>YEARS</td>
<td></td>
</tr>
<tr>
<td>MINUTE_SECOND</td>
<td>Minutes and seconds</td>
<td>MINUTES:SECONDS</td>
<td></td>
</tr>
<tr>
<td>HOUR_MINUTE</td>
<td>Hours and minutes</td>
<td>HOURS:MINUTES</td>
<td></td>
</tr>
<tr>
<td>DAY_HOUR</td>
<td>Days and hours</td>
<td>DAYS HOURS</td>
<td></td>
</tr>
<tr>
<td>YEAR_MONTH</td>
<td>Years and months</td>
<td>YEARS-MONTHS</td>
<td></td>
</tr>
<tr>
<td>HOUR_SECOND</td>
<td>Hours, minutes,</td>
<td>HOURS:MINUTES:SECONDS</td>
<td></td>
</tr>
<tr>
<td>DAY_MINUTE</td>
<td>Days, hours, minutes</td>
<td>DAYS HOURS:MINUTES</td>
<td></td>
</tr>
<tr>
<td>DAY_SECOND</td>
<td>Days, hours, minutes, seconds</td>
<td>DAYS HOURS:MINUTES:SECONDS</td>
<td></td>
</tr>
</tbody>
</table>

### Note

Typing a negative sign (-) in front of expression numbers causes Date_Add functions to subtract and Date_Sub functions to add. (Table info from mysql.com.)

**DATE_ADD(date/time,INTERVAL expr type), ADDDATE(date/time,INTERVAL expr type)**

Adds the time interval expression to the `date/time` parameter and returns it.

**INTERACTIVE SESSION:**

```sql
mysql> select date_add('2019-10-31 02:44:10', interval '5 10:03' day_minute);
+----------------------------------------------------------------+
<table>
<thead>
<tr>
<th>date_add('2019-10-31 02:44:10', interval '5 10:03' day_minute)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019-11-05 12:47:10</td>
</tr>
</tbody>
</table>
+----------------------------------------------------------------+

mysql> select adddate('2019-10-31 02:44:10', interval '5 10:03' day_minute);
+---------------------------------------------------------------+
<table>
<thead>
<tr>
<th>adddate('2019-10-31 02:44:10', interval '5 10:03' day_minute)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019-11-05 12:47:10</td>
</tr>
</tbody>
</table>
+---------------------------------------------------------------+
```

**DATE_SUB(date/time,INTERVAL expr type), SUBDATE(date/time,INTERVAL expr type)**

Subtracts the time interval expression from the `date/time` parameter and returns it.
**EXTRACT (expr type FROM date/time)**

Returns a portion of the date/time.

**CURRENT_TIME()**

Returns the current time in the format 'HH:MM:SS' as a string.

**HOUR(time)**

Returns the hour of the time parameter.
**MINUTE(time)**

Returns the minutes of the time parameter.

**SECOND(time)**

Returns the seconds of the time parameter.

**TIME_TO_SEC(time)**

Converts the time into seconds since midnight (00:00:00).

**SEC_TO_TIME(sec)**

Converts the seconds (since midnight) back to time.
INTERACTIVE SESSION:

mysql> select sec_to_time(54170);
+--------------------+
| sec_to_time(54170) |
+--------------------+
| 15:02:50           |
+--------------------+

TIME_FORMAT(time,format)

Identical to Date_Format except that only the hour, minute, and seconds specifiers in the format parameter will work. Otherwise a NULL is returned.

INTERACTIVE SESSION:

mysql> select time_format('15:32:33', '%h:%i %p');
+-------------------------------------+
| time_format('15:32:33', '%h:%i %p') |
+-------------------------------------+
| 03:32 PM                            |
+-------------------------------------+

NOW(), SYSDATE(), CURRENT_TIMESTAMP()

Returns the current date and time in format 'YYYY-MM-DD HH:MM:SS' if used in the context of string. If it is used in the context of an integer, this format is returned: YYYYMMDDHHMMSS.

INTERACTIVE SESSION:

mysql> select now(), sysdate() + '0', current_timestamp();
+---------------------+-----------------+---------------------+
| now()               | sysdate() + '0' | current_timestamp() |
+---------------------+-----------------+---------------------+
+---------------------+-----------------+---------------------+

UNIX_TIMESTAMP(), UNIX_TIMESTAMP(date)

Returns the Unix timestamp. If a date parameter is provided, that date is returned as a Unix timestamp.

INTERACTIVE SESSION:

mysql> select unix_timestamp(), unix_timestamp(now());
+------------------+-----------------------+
| unix_timestamp() | unix_timestamp(now()) |
+------------------+-----------------------+
|       1415054660 |            1415054660 |
+------------------+-----------------------+

FROM_UNIXTIME(unix_timestamp), FROM_UNIXTIME(unix_timestamp, format)

Converts the unix_timestamp back to normal time format 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS. If a format was specified (using specifiers mentioned in DATE_FORMAT), the date/time is returned in that format.
INTERACTIVE SESSION:

```sql
mysql> select from_unixtime(unix_timestamp()), from_unixtime('1415054660');
+---------------------------------+-----------------------------+
| from_unixtime(unix_timestamp()) | from_unixtime('1415054660') |
+---------------------------------+-----------------------------+
+---------------------------------+-----------------------------+
```

Copyright © 1998-2014 O'Reilly Media, Inc.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.
See [http://creativecommons.org/licenses/by-sa/3.0/legalcode](http://creativecommons.org/licenses/by-sa/3.0/legalcode) for more information.
Advanced SQL Syntax, Part 2

Math Functions

We'll explain each of these functions in detail with samples of its usage, and we'll explain the parameters and output of the functions. You should explore each of them on your own in order to fully understand them.

You can use this lesson as a reference. We'll cover these functions in this lesson (x and y are numbers (int or float)):

<table>
<thead>
<tr>
<th>General Functions</th>
<th>Rounding Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOD(x,y)</td>
<td>TRUNCATE(x,y)</td>
</tr>
<tr>
<td>ABS(x)</td>
<td>FLOOR(x)</td>
</tr>
<tr>
<td>SIGN(x)</td>
<td>CEILING(x)</td>
</tr>
<tr>
<td>LEAST(x,y,...)</td>
<td>ROUND(x)</td>
</tr>
<tr>
<td>GREATEST(x,y,...)</td>
<td>ROUND(x,y)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Wave Functions</th>
<th>Exponential Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIN(x)</td>
<td>EXP(x)</td>
</tr>
<tr>
<td>COS(x)</td>
<td>LOG(x)</td>
</tr>
<tr>
<td>TAN(x)</td>
<td>LOG10(x)</td>
</tr>
<tr>
<td>ASIN(x)</td>
<td>SQRT(x)</td>
</tr>
<tr>
<td>ACOS(x)</td>
<td>POW(x,y)</td>
</tr>
<tr>
<td>ATAN(x)</td>
<td>POWER(x,y)</td>
</tr>
<tr>
<td>ATAN2(x,y)</td>
<td>COT(x)</td>
</tr>
<tr>
<td>PI()</td>
<td>DEGREES(x)</td>
</tr>
<tr>
<td>RADIANS(x)</td>
<td></td>
</tr>
</tbody>
</table>

**MOD(x,y)**

Returns the remainder of x divided by y. The return is an integer. If both parameters are of the floating type, the remainder of the division will be rounded to the nearest integer.

**INTERACTIVE SESSION:**

```sql
mysql> select mod(10.5,6), mod(10.45,6);
+-------------+--------------+
| mod(10.5,6) | mod(10.45,6) |
+-------------+--------------+
|         4.5 |         4.45 |
+-------------+--------------+
1 row in set (0.00 sec)
```

**ABS(x)**

Returns the absolute value of x.
**SIGN(x)**

Returns one of only three kinds of returns: If \(x\) is a negative number, returns \(-1\). If \(x\) is positive, returns \(1\). If \(x\) is zero, returns zero.

**LEAST(x,y,...)**

Requires at least two parameters. This function searches the list of entries and returns the smallest value within the list. This includes strings as well—for characters and strings, lowercase letters have a higher value than uppercase letters.

**GREATEST(x,y,...)**

Requires at least two parameters. This function searches the list of entries and returns the largest value within the list. This includes strings as well—for characters and strings, uppercase letters have a higher value than lowercase letters.

**TRUNCATE(x,y)**

Chops off the decimal numbers that come after the \(y\)th decimal and returns the result. However, the \(y\)th decimal is rounded down if the \(x\) value is negative.
**INTERACTIVE SESSION:**

mysql> select truncate(1.123, 2), truncate(5.345,1), truncate(-5.345,1);
+--------------------+-------------------+--------------------+
| truncate(1.123, 2) | truncate(5.345,1) | truncate(-5.345,1) |
+--------------------+-------------------+--------------------+
|               1.12 |               5.3 |               -5.3 |
+--------------------+-------------------+--------------------+

**FLOOR(x)**

Rounds x down to the nearest integer.

**INTERACTIVE SESSION:**

mysql> select floor(1.60), floor(-3.78);
+-------------+--------------+
| floor(1.60) | floor(-3.78) |
+-------------+--------------+
|           1 |           -4 |
+-------------+--------------+

**CEILING(x)**

Rounds x up to the nearest integer.

**INTERACTIVE SESSION:**

mysql> select ceiling(1.20), ceiling(-3.78);
+---------------+----------------+
| ceiling(1.20) | ceiling(-3.78) |
+---------------+----------------+
|             2 |             -3 |
+---------------+----------------+

**ROUND(x), ROUND(x,y)**

Rounds x to the nearest integer. However, if a second parameter y is entered, this function rounds x to the nearest yth decimal. You can even use negative values for y, so as to round by tens (1 to the right of the decimal, thus y as -1).

**INTERACTIVE SESSION:**

mysql> select round(2.34), round(-3.76), round(5.379,2), round(34.56, -1);
+-------------+--------------+----------------+------------------+
| round(2.34) | round(-3.76) | round(5.379,2) | round(34.56, -1) |
+-------------+--------------+----------------+------------------+
|           2 |           -4 |           5.38 |               30 |
+-------------+--------------+----------------+------------------+
1 row in set (0.00 sec)

**SIN(x)**

Returns the sine of x in radians. The x value should be in radians.
INTERACTIVE SESSION:

mysql> select sin(3.14);
+-----------+
| sin(3.14) |
+-----------+
|  0.001593 |
+-----------+

COS(x)
Returns the cosine of x in radians. The x value should be in radians.

INTERACTIVE SESSION:

mysql> select cos(5.20);
+-----------+
| cos(5.20) |
+-----------+
|  0.468517 |
+-----------+

TAN(x)
Returns the tangent of x in radians. The x value should be in radians.

INTERACTIVE SESSION:

mysql> select tan(-2.49);
+-----------+
| tan(-2.49) |
+-----------+
|  0.762721 |
+-----------+

ASIN(x)
Returns the arc sine of x in radians. However, a NULL is returned if x is not within the range of -1 and 1.

INTERACTIVE SESSION:

mysql> select asin(-1), asin(0), asin(1), asin(1.1);
+-----------+----------+----------+-----------+
| asin(-1)  | asin(0)  | asin(1)  | asin(1.1) |
+-----------+----------+----------+-----------+
| -1.570796 | 0.000000 | 1.570796 |      NULL |
+-----------+----------+----------+-----------+

ACOS(x)
Returns the arc cosine of x in radians. However, a NULL is returned if x is not within the range of -1 and 1.
### ATAN(x)

Returns the arc tangent of `x` in radians.

**INTERACTIVE SESSION:**

```sql
mysql> select atan(-1), atan(0), atan(1), atan(1.1);
+-------------------+---------+------------------+------------------+
| atan(-1)          | atan(0) | atan(1)          | atan(1.1)        |
| -0.78539816339745 |       0 | 0.78539816339745 | 0.83298126667443 |
+-------------------+---------+------------------+------------------+
```

### ATAN2(x,y)

Returns the arc tangent of `y/x` in radians, but the signs of `x` and `y` are used to determine the quadrant.

**INTERACTIVE SESSION:**

```sql
mysql> select atan(-3,1), atan(2,-4);
+------------+------------+
| atan(-3,1) | atan(2,-4) |
| -1.249046  |   2.677945 |
+------------+------------+
```

### COT(x)

Returns the cotangent of `x` in radians. If the value returned is out of bounds, a **NULL** is returned instead.

**INTERACTIVE SESSION:**

```sql
mysql> select cot(2.3), cot(0);
+-------------+--------+
| cot(2.3)    | cot(0) |
| -0.89348446 |   NULL |
+-------------+--------+
```

### PI()

Returns the constant pi.
**INTERACTIVE SESSION:**

```sql
mysql> select pi();
+----------+
| PI()     |
+----------+
| 3.141593 |
+----------+
```

**DEGREES(x)**

Converts x from radians into degrees and returns the value.

**INTERACTIVE SESSION:**

```sql
mysql> select degrees(pi());
+---------------+
| degrees(pi()) |
+---------------+
| 180.000000    |
+---------------+
```

**RADIANS(x)**

Converts the value x from degrees into radians and returns the value.

**INTERACTIVE SESSION:**

```sql
mysql> select radians(270);
+--------------+
| radians(270) |
+--------------+
| 4.712389     |
+--------------+
```

**EXP(x)**

Returns natural logarithms e to the power of x.

**INTERACTIVE SESSION:**

```sql
mysql> select exp(2), exp(-3);
+----------+----------+
| exp(2)   | exp(-3)  |
+----------+----------+
| 7.389056 | 0.049787 |
+----------+----------+
```

**LOG(x)**

Returns natural logarithm of x.
**INTERACTIVE SESSION:**

```
mysql> select log(3), log(-2.3);
+----------+-----------+
| log(3)   | log(-2.3) |
+----------+-----------+
| 1.098612 |      NULL |
+----------+-----------+
```

**LOG10(x)**

Returns natural logarithm base 10 of \( x \).

**INTERACTIVE SESSION:**

```
mysql> select log10(4), log10(100), log10(-10);
+----------+------------+------------+
| log10(4) | log10(100) | log10(-10) |
+----------+------------+------------+
| 0.602060 |   2.000000 |       NULL |
+----------+------------+------------+
```

**SQRT(x)**

Returns square root of \( x \).

**INTERACTIVE SESSION:**

```
mysql> select sqrt(4), sqrt(9), sqrt(-25);
+----------+----------+-----------+
| sqrt(4)  | sqrt(9)  | sqrt(-25) |
+----------+----------+-----------+
| 2.000000 | 3.000000 |      NULL |
+----------+----------+-----------+
```

**POW(x,y), POWER(x,y)**

Pow and Power are identical functions. You can use either one. This function returns \( x \) to the power of \( y \).

**INTERACTIVE SESSION:**

```
mysql> select pow(2,3), power(3,-4);
+----------+-------------+
| pow(2,3) | power(3,-4) |
+----------+-------------+
| 8.000000 |    0.012346 |
+----------+-------------+
```
### String and Character Functions

You can refer back to this lesson later as a reference. We will cover these functions:

<table>
<thead>
<tr>
<th>Character Manipulating Functions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII(str)</td>
<td>OCT(x)</td>
</tr>
<tr>
<td>CONV(x,y,z)</td>
<td>HEX(x)</td>
</tr>
<tr>
<td>BIN(x)</td>
<td>CHAR(x,y,z,...)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>String Info Functions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LENGTH(str)</td>
<td>CHAR_LENGTH(str)</td>
</tr>
<tr>
<td>OCTET_LENGTH(str)</td>
<td>CHARACTER_LENGTH(str)</td>
</tr>
<tr>
<td>LOCATE(str1,str2)</td>
<td>INSTR(str1,str2)</td>
</tr>
<tr>
<td>POSITION(str1 IN str2)</td>
<td>SOUNDEx(str)</td>
</tr>
<tr>
<td>LOCATE(str1,str2,x)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>String Manipulating Functions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CONCAT(str1,str2,...)</td>
<td>SUBSTRING(str,x)</td>
</tr>
<tr>
<td>LPAD(str1,x,str2)</td>
<td>SUBSTRING(str FROM x)</td>
</tr>
<tr>
<td>RPAD(str1,x,str2)</td>
<td>SUBSTRING(str,x,y)</td>
</tr>
<tr>
<td>LEFT(str,x)</td>
<td>SUBSTRING(str FROM x FOR y)</td>
</tr>
<tr>
<td>RIGHT(str,x)</td>
<td>MID(str,x,y)</td>
</tr>
<tr>
<td>LTRIM(str)</td>
<td>SUBSTRING_INDEX(str1,str2,x)</td>
</tr>
<tr>
<td>RTRIM(str)</td>
<td></td>
</tr>
<tr>
<td>LOAD_FILE(filen)</td>
<td></td>
</tr>
<tr>
<td>SPACE(str)</td>
<td>LOWER(str)</td>
</tr>
<tr>
<td>REPLACE(str1,str2,str3)</td>
<td>UPPER(str)</td>
</tr>
<tr>
<td>REPEAT(str,x)</td>
<td>MAKE_SET(bit,str)</td>
</tr>
<tr>
<td>REVERSE(str)</td>
<td>EXPORT_SET(bit,str1,str2,[str3,[x]])</td>
</tr>
<tr>
<td>INSERT(str1,x,y,str2)</td>
<td>FIND_IN_SET(str1,strlist)</td>
</tr>
<tr>
<td>ELT(x,str1,str2,str3,...)</td>
<td>FIELD(str1,str2,str3,str4,...)</td>
</tr>
<tr>
<td>TRIM([BOTH</td>
<td>LEADING</td>
</tr>
</tbody>
</table>

*filen* represents a filename including path to the file, *str* represents string, *strlist* represents a list string, *bit* represents bits, and *x*, *y*, and *z* represent numbers.

We'll explain each of these in detail, with examples of usage. We'll also explain the parameters and output.

**ASCII(str)**

Returns the ASCII code value of the leftmost character of string *str*. 
**CONV(x,y,z)**

This function converts numbers between different bases, such as hexadecimal to decimal, and binary to octal. x is the number being converted from base y to base z. The bases y and z can range from 2 to 36. If the base has a preceding negative sign, x is treated as an unsigned number. x can also be a string. conv() works with 64-bit precision.

**BIN(x)**

Returns the binary of x, where x is in base 10. This is identical to conv(x,10,2).

**OCT(x)**

Returns the octal value of x, where x is a decimal number. This is identical to conv(x,10,8).
HEX(x)

Returns the hexadecimal value of x, where x is a decimal number. This is identical to conv(x,10,16).

**INTERACTIVE SESSION:**

```sql
mysql> select hex(123), conv(123,10,16);
+----------+-----------------+
<table>
<thead>
<tr>
<th>hex(123)</th>
<th>conv(123,10,16)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7B</td>
<td>7B</td>
</tr>
</tbody>
</table>
+----------+-----------------+
```

CHAR(x,y,z,...)

Converts x, y, z, etc., to ASCII characters. x, y, z, etc. are decimal ASCII codes. If any of the parameters are float values, they are truncated to integers.

**INTERACTIVE SESSION:**

```sql
mysql> select char(72, 79, 89.3, '79.9', 85, 76);
+------------------------------------+
<table>
<thead>
<tr>
<th>char(72, 79, 89.3, '79.9', 85, 76)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOYOUL</td>
</tr>
</tbody>
</table>
+------------------------------------+
```

LENGTH(str), CHAR_LENGTH(str), OCTET_LENGTH(str), CHARACTER_LENGTH(str)

These functions are all identical. They return the length of the string str.

**INTERACTIVE SESSION:**

```sql
mysql> select length('this'), char_length('is');
+----------------+-------------------+
<table>
<thead>
<tr>
<th>length('this')</th>
<th>char_length('is')</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>
+----------------+-------------------+
mysql> octet_length('really'), character_length('cool');
+------------------------+--------------------------+
<table>
<thead>
<tr>
<th>octet_length('really')</th>
<th>character_length('cool')</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>
+------------------------+--------------------------+
```

LOCATE(str1,str2), POSITION(str1 IN str2)

Returns the position of the first occurrence of substring str1 in the string str2.
### LOCATE(str1,str2,x)

Returns the position of the first occurrence of substring `str1` in the string `str2`, starting at position `x`. If `str2` is not found, returns a zero.

**INTERACTIVE SESSION:**

```sql
mysql> select locate('hak','what'), position('ssy' in 'missy');
+----------------------+----------------------------+
| locate('hak','what') | position('ssy' in 'missy') |
+----------------------+----------------------------+
|                    0 |                          3 |
+----------------------+----------------------------+
```

**LOCATE(str1,str2,x)**

Returns the position of the first occurrence of substring `str1` in the string `str2`, starting at position `x`. If `str2` is not found, returns a zero.

**INTERACTIVE SESSION:**

```sql
mysql> select locate('ufo','didyoufocus?',2);
+--------------------------------+
| locate('ufo','didyoufocus?',2) |
+--------------------------------+
|                              6 |
+--------------------------------+
mysql> select locate('ufo','didyoufucus?',9);
+--------------------------------+
| locate('ufo','didyoufucus?',9) |
+--------------------------------+
|                              0 |
+--------------------------------+
```

### INSTR(str1, str2)

Returns the position of the first occurrence of substring `str2` in the string `str1`. This is identical to `locate` and `position`, except that the substring is the second argument in instr.

**INTERACTIVE SESSION:**

```sql
mysql> select instr('missy','ssy');
+----------------------+
| instr('missy','ssy') |
+----------------------+
|                    3 |
+----------------------+
```

### SOUNDEX(str)

This is an interesting function. Returns the soundex of the string `str`. Similar-sounding strings should have the same soundex value. Treats non-alphabet strings as vowels.

**INTERACTIVE SESSION:**

```sql
mysql> select soundex('greetings'), soundex('meeting'), soundex('mayor');
+----------------------+--------------------+------------------+
| soundex('greetings') | soundex('meeting') | soundex('mayor') |
+----------------------+--------------------+------------------+
| G6352                | M352               | M600             |
+----------------------+--------------------+------------------+
```
**CONCAT**\( (str1, str2, \ldots) \)

Returns a string composed of \( str1 \), \( str2 \), etc. If the parameters are integers, float or other non-string values, they are converted to a string.

**INTERACTIVE SESSION:**

```sql
mysql> select concat('po', 'ke', 'm', 'on');
+-----------------------------------------------+
| concat('po', 'ke', 'm', 'on')               |
+-----------------------------------------------+
| pokemon                                      |
+-----------------------------------------------+
```

**LPAD**\( (str1, x, str2) \)

The string \( str2 \) is added to the left side of string \( str1 \) until the length of the string is \( x \).

**INTERACTIVE SESSION:**

```sql
mysql> select lpad('h no!', 8, 'oo');
+------------------------+
| lpad('h no!', 8, 'oo') |
+------------------------+
| oooh no!               |
+------------------------+
```

**RPAD**\( (str1, x, str2) \)

The string \( str2 \) is added to the right side of string \( str1 \) until the length of the string is \( x \).

**INTERACTIVE SESSION:**

```sql
mysql> select rpad('a ghost said b', 18, 'oo');
+----------------------------------+
| rpad('a ghost said b', 18, 'oo') |
+----------------------------------+
| a ghost said boooo               |
+----------------------------------+
```

**LEFT**\( (str, x) \)

Returns part of the \( str \) string of length \( x \), starting from the beginning of the string.

**INTERACTIVE SESSION:**

```sql
mysql> select left('jimmythecricket', 5);
+----------------------------+
| left('jimmythecricket', 5) |
+----------------------------+
| jimmy                      |
+----------------------------+
```

**RIGHT**\( (str, x) \)

Returns part of the \( str \) string of length \( x \), starting from the right side of the string.
INTERACTIVE SESSION:

```
mysql> select right('jimmythecricket', 7);
+-----------------------------+
| right('jimmythecricket', 7) |
+-----------------------------+0
| cricket                     |
+-----------------------------+
```

**LTRIM(str)**

Trims the space characters off of the string \texttt{str} from the left side of the string and returns it.

INTERACTIVE SESSION:

```
mysql> select ltrim('   whoopy');
+--------------------+
| ltrim('   whoopy') |
+--------------------+
| whoopy             |
+--------------------+
```

**RTRIM(str)**

Trims the space characters off the right side of the string \texttt{str} and returns it.

INTERACTIVE SESSION:

```
mysql> select rtrim('snoopy   ');
+--------------------+
| rtrim('snoopy   ') |
+--------------------+
| snoopy             |
+--------------------+
```

**LOAD_FILE(filen)**

Reads the file indicated by the string \texttt{filen} (with full path) and returns the contents of the file as a string. Note that most servers—including OST's—do not grant MySQL users file access through MySQL, so this function usually won't work.

INTERACTIVE SESSION:

```
mysql> update table_name set some_column = load_file("/tmp/blah");
```

**SPACE(x)**

Returns a string of spaces \texttt{x} characters long.
**INTERACTIVE SESSION:**

```sql
mysql> select concat('front', space(7), 'end');
+----------------------------------+
| concat('front', space(7), 'end') |
+----------------------------------+
| front       end                  |
+----------------------------------+
```

**REPLACE(str1,str2,str3)**

Replaces all occurrences of `str2` to `str3` in string `str1`.

**INTERACTIVE SESSION:**

```sql
mysql> select replace('u am a chump', 'u', 'i');
+-----------------------------------+
| replace('u am a chump', 'u', 'i') |
+-----------------------------------+
| i am a chimp                      |
+-----------------------------------+
```

**REPEAT(str,x)**

Returns a string composed of string `str` x times.

**INTERACTIVE SESSION:**

```sql
mysql> select repeat('super ', 3);
+---------------------+
| repeat('super ', 3) |
+---------------------+
| super super super   |
```

**REVERSE(str)**

Returns the reverse string of `str`.

**INTERACTIVE SESSION:**

```sql
mysql> select reverse('evian');
+------------------+
| reverse('evian') |
+------------------+
| naive            |
+------------------+
```

**INSERT(str1,x,y,str2)**

Inserts the string `str2` into the string `str1` at position `x` and replaces `y` characters from position `x`.

**WARNING** Be careful! This is different from the `insert` command used to insert into a database.
**ELT(x,str1,str2,str3,...)**

Returns string \(\text{str1}\) if \(x\) is 1, \(\text{str2}\) if \(x\) is 2, \(\text{str3}\) if \(x\) is 3 and so on. If the index value is out of range, returns NULL.

**FIELD(str1,str2,str3,str4,...)**

This function searches through string parameters from \(\text{str2}\) to the end for string \(\text{str1}\). If found, the index of the string parameter is returned. For instance, 1 is returned if \(\text{str2}\) is equal to \(\text{str1}\), 2 is returned if \(\text{str3}\) is equal to \(\text{str1}\) and so on. If \(\text{str1}\) is not found, a zero is returned.

**TRIM([[BOTH | LEADING | TRAILING] [str2] FROM] str1)**

This one is easier to explain with some examples than in text.
**INTERACTIVE SESSION:**

```sql
mysql> select trim(' ab cd ');  
+----------------------+
| trim(' ab cd ')     |
+----------------------+
| ab cd               |
+----------------------+

mysql> select trim(both ' ' from ' ab cd ');  
+------------------------------------+  
| trim(both ' ' from ' ab cd ')     |
+------------------------------------+
| ab cd                             |
+------------------------------------+

mysql> select trim(both '?!' from '?!?!ab  cd?!?!!');  
+---------------------------------------+  
| trim(both '?!' from '?!?!ab  cd?!?!!')|
+---------------------------------------+
| ab cd                                 |
+---------------------------------------+

mysql> select trim(leading '?!' from '?!?!ab  cd?!?!!');  
+------------------------------------------+  
| trim(leading '?!' from '?!?!ab  cd?!?!!')|
+------------------------------------------+
| ab cd?!?!                               |
+------------------------------------------+

mysql> select trim(trailing '?!' from '?!?!ab  cd?!?!!');  
+-------------------------------------------+  
| trim(trailing '?!' from '?!?!ab  cd?!?!!')|
+-------------------------------------------+
| ?!?!ab  cd                                |
+-------------------------------------------+
```

**SUBSTRING** *(str, x), SUBSTRING(str FROM x)*

These functions are identical. Both return a substring of *str* from position *x* to the end of the string.

**INTERACTIVE SESSION:**

```sql
mysql> select substring('beginning to the end', 11);  
+---------------------------------------+  
| substring('beginning to the end', 11) |
+---------------------------------------+
| to the end                             |
+---------------------------------------+

mysql> select substring('beginning to the end' from 11);  
+-------------------------------------------+  
| substring('beginning to the end' from 11) |
+-------------------------------------------+
| to the end                               |
+-------------------------------------------+
```

**SUBSTRING** *(str, x, y), SUBSTRING(str FROM x FOR y), MID**(str, x, y)**

Same as the substring function, with an extra parameter. Here the function returns the substring of string *str* from position *x* with a length of *y* characters.
**SUBSTRING_INDEX(str1,str2,x)**

The string `str2` acts as the delimiter and splits the string `str1`. Integer `x` will determine which delimiter to use to split the string `str1`. The sign of `x` will determine whether to select the portion of the string to the left or to the right of the delimiter.

**LCASE(str), LOWER(str)**

Both of these functions return a lowercase version of string `str`.
INTERACTIVE SESSION:

```sql
mysql> select lcase('WhAt Is GoInG oN?');
+-----------------------------+
<table>
<thead>
<tr>
<th>lcase('WhAt Is GoInG oN?')</th>
</tr>
</thead>
<tbody>
<tr>
<td>what is going on?</td>
</tr>
<tr>
<td>----------------------------</td>
</tr>
</tbody>
</table>
mysql> select lower('WhAt Is GoInG oN?');
+-----------------------------+
<table>
<thead>
<tr>
<th>lower('WhAt Is GoInG oN?')</th>
</tr>
</thead>
<tbody>
<tr>
<td>what is going on?</td>
</tr>
<tr>
<td>----------------------------</td>
</tr>
</tbody>
</table>
```

**UCASE(str), UPPER(str)**

Both of these functions return an uppercase version of string str.

INTERACTIVE SESSION:

```sql
mysql> select ucase('WhAt Is GoInG oN?');
+-----------------------------+
<table>
<thead>
<tr>
<th>ucase('WhAt Is GoInG oN?')</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHAT IS GOING ON?</td>
</tr>
<tr>
<td>----------------------------</td>
</tr>
</tbody>
</table>
mysql> select upper('WhAt Is GoInG oN?');
+-----------------------------+
<table>
<thead>
<tr>
<th>upper('WhAt Is GoInG oN?')</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHAT IS GOING ON?</td>
</tr>
<tr>
<td>----------------------------</td>
</tr>
</tbody>
</table>
```

**MAKE_SET(bit,str1,str2,...)**

Returns a set (a string containing substrings separated by commas(',')) consisting of the strings that have the corresponding bit in bits set. str1 corresponds to bit 0, str2 to bit 1, etc. NULL strings in str1, str2, and so on, are not appended to the result.

INTERACTIVE SESSION:

```sql
mysql> select make_set(1,'a','b','c');
+-------------------------+
<p>| make_set(1,'a','b','c') |</p>
<table>
<thead>
<tr>
<th>a</th>
</tr>
</thead>
</table>
mysql> select make_set(1 | 4,'hello','nice','world');
+----------------------------------------+
<p>| make_set(1 | 4,'hello','nice','world') |</p>
<table>
<thead>
<tr>
<th>hello,world</th>
</tr>
</thead>
</table>
mysql> select make_set(0,'a','b','c');
+-------------------------+
<p>| make_set(0,'a','b','c') |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
</table>
```
**EXPORT_SET(bit,str1,str2,[str3,[x]]])**

Returns a string where for every bit set in 'bit', you get an 'on' string and for every reset bit, you get an 'off' string. Each string is separated with 'separator' (default ',') and only 'number_of_bits' (default 64) of 'bits' is used.

**INTERACTIVE SESSION:**

```
mysql> select export_set(5,'Y','N',';',4);
+-----------------------------------------------+
| export_set(5,'Y','N',';',4) |
+-----------------------------------------------+
| Y,N,Y,N                     |
+-----------------------------------------------+
```

**FIND_IN_SET(str1,strlist)**

The strlist is a list of strings separated by commas. This function searches for str1 in the string list and returns the index/position relative to the strlist.

**INTERACTIVE SESSION:**

```
mysql> select find_in_set('ogre', 'wild,gandolf,ogre,blah');
+-----------------------------------------------+
| find_in_set('ogre', 'wild,gandolf,ogre,blah') |
+-----------------------------------------------+
|                                             3 |
+-----------------------------------------------+
```
Final Project

The goal of this project is to expand on the shopping cart you created in the Introduction to PHP course. In case you didn’t take that course, you can create a simple version of this cart from scratch. The new shopping cart will not only have products, prices, registration, and a checkout area, but it will also have a managers’ area in which products, customers, and purchases can be viewed, and new products can be entered. You can make this shopping cart any way you wish.

Final Project, Part 1

For the first part of this project, you will need to design and create your SQL tables for the cart’s products, registered customers, and purchases. Be sure to think carefully about how the tables will relate to one another for the purposes of your cart. When you are finished creating your tables, hand them in.

Final Project, Part 2

Now it’s time to create your shopping cart and managers’ area. Although how you implement this is up to you, you should include these elements:

For the shopping cart:

- A way for the customers to view the products from the database, along with prices.
- A way to add each product to the customer’s shopping cart.
- A way to view the shopping cart.
- A way to register and checkout.

For the managers’ area:

- A way to view, update, and add products and prices.
- A way to view customers and their purchases.

For the sake of evaluation, try to include as many of the elements discussed in this course as you can. You should also observe good programming practices, with comments, code reusability, and readability.

Hand in two files: The first file should be the starting point for your customers. The second file should be the starting point for managers.

Be creative and have fun! You want to present yourself in a professional yet friendly way, so feel free to express yourself!

Copyright © 1998-2014 O'Reilly Media, Inc.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License. See http://creativecommons.org/licenses/by-sa/3.0/legalcode for more information.