# The City Lit Institute

**Department of Computing**

## Keeley Street, Holborn, London WC2B 4BA

**MySQL - Class Project (50 records required)**

**(with Apache server)**

Resource:

<http://uk.search-results.com/web?l=dis&o=1921&q=world+countries%2C+cities%2Cpopulation%2Clongitude+and+latitudes&atb=sysid%3D406%3Aappid%3D113%3Auid%3Dcfdc6657e0b1de64%3Auc%3D1329072862%3Aq%3Dworld+countries%2C+cities%2Cpopulation%2Clongitude+and+latitudes%3Asrc%3Dieb%3Ao%3D1921>

<http://www.timegenie.com/latitude_and_longitude/>

http://www.infoplease.com/ipa/A0001769.hml

**LECTURER** : **ALEXANDER ADU-SARKODIE**

MSc. Telematics (IT & Telecom), MSc. Eng., Dip. Russ. Lang., Teach. Cert, AMIAEng (UK), MBCS (UK), MIfL(UK)

**Blog**: [**http://www.blogger.com/profile/14800490193632788559**](http://www.blogger.com/profile/14800490193632788559)

**Email:** [**aadusarkodie1@gmail.com**](mailto:aadusarkodie1@gmail.com)

**Class Project**

The table creation is accomplished with the CREATE TABLE statement, in which we give the new table's name. The statement begins with CREATE TABLE, followed by the table name. Then, enclosed in brackets, we put the list of columns, and information about the keys. Each column is assigned a name, data type, the NULL or NOT NULL attribute (here, NOT NULL means the column cannot have a NULL value) and a default value, if appropriate.

CREATE TABLE cities (

id int(11) NOT NULL auto\_increment,

city\_name varchar(50) NOT NULL default ' ',

latitude varchar(15) NOT NULL default ' ',

longitude varchar(15) NOT NULL default ' ',

population int(11) NOT NULL default '0',

country\_code char(2) NOT NULL default '',

PRIMARY KEY (id)

) AUTO\_INCREMENT=1 ;

The id column is our primary key, a column which uniquely identifies each city. Its data type is INT (an integer number), and MySQL will assign unique numbers to it, thanks to the auto\_increment attribute. Note that we cannot use the city name as a primary key, as some **city names** are not **unique** in the world. We also use an integer for the population data.

The other columns use character (CHAR) or variable character (VARCHAR) data types. When we know the exact length of data, it's better to use CHAR, specifying the length of the column as in CHAR(2). Otherwise we use a variable character data type, which will take only the space needed by each piece of data, and we specify the maximum length, as in VARCHAR(15).

After the columns list, we have some table-specific information, like its type, and the first value for the auto-increment column. SQL statements end with a semi-colon.

Having created our cities table, we do the same operation, this time for the countries table.

CREATE TABLE countries (

country\_code char(2) NOT NULL default '',

country\_name varchar(100) NOT NULL default ''

) TYPE=MyISAM;

We notice that the country\_code column is present in both tables. This shows the **relational** **principle**: the country\_code in cities refers to the same column in countries. This way, we save on space, having each country name only once in our database

**Adding Data with INSERT**

Let's first examine the INSERT statement, by looking at the code phpMyAdmin generates when we do an Insert operation. We bring up the Insert sub-page, in Table view for the countries table, and we enter data about a country:

When we click Go, the data is inserted and phpMyAdmin shows us the INSERT statement used:

INSERT INTO `countries` ( `country\_code` , `country\_name` )   
VALUES ('ca', 'Canada');

After the INSERT INTO part, we have the table name. In MySQL, we can enclose table names and column names within backticks, in case there are special characters in them, like **reserved** words or **accented** characters. Then we open a first set of brackets, listing the columns in which we want to insert, separated by commas. The reserved word VALUES follows, then the last set of brackets enclosing the values, in the same order as the columns list. If the values have a character data type, we have to enclose them within quotes.

We can now insert a city.

INSERT INTO `cities` ( `id` , `city\_name` , `latitude` , `longitude` , `population` , `country\_code` )   
VALUES ('', 'Sherbrooke', '45 23 59.00', '-71 46 11.00', 125000, 'ca');

Here, we put an empty value for id, because this column's **auto-increment** attribute will provide a value. We also see that the population value, being **numeric**, does **not need** to be **surrounded by quotes.**

Let's end this section by inserting some data for another country and city, which we will need later.

INSERT INTO `countries` ( `country\_code` , `country\_name` )   
VALUES ('zh', 'China');

INSERT INTO `cities` ( `id` , `city\_name` , `latitude` , `longitude` , `population` , `country\_code` )   
VALUES ('', 'Shanghai', '31 13 58.00', '121 26 59.99', 11000000, 'zh');

**Updating Data with UPDATE**

We first click on Browse for table cities, displayed our single row of data.

By clicking on the small pencil-shaped icon (or Edit link), we go to the Edit panel for this row. We decide to change the population value to 130000. After a click on Save, phpMyAdmin shows the following statement:

UPDATE `cities` SET `population` = '130000' WHERE `id` = '1' LIMIT 1 ;

Here we have the UPDATE keyword, followed by the table name. The **SET** keyword introduces the **list** of **modifications** (here only the population), which follows the format column = new value.

We now see the condition WHERE `id` = '1', which uses the primary key information to limit the change to only this row, i.e. only this city.

**The limit 1 part is a safeguard added by phpMyAdmin, in case there would be no primary key defined, to avoid doing the change to more than one row.**

More than one column can be changed in a single UPDATE operation:

UPDATE `cities` SET `city\_name` = 'Sherbrooke, Quebec',  
`population` = '130001' WHERE `id` = '1' LIMIT 1 ;

**Deleting Data with DELETE**

In Browse mode on table cities, clicking on the small red trash-can icon (or Delete link) brings up a dialog to confirm the execution of the following statement:

DELETE FROM `cities` WHERE `id` = '1' LIMIT 1 ;

The syntax is simple, involving just the table name, and the condition to apply for the delete operation.

Omitting the WHERE condition in an UPDATE or DELETE operation is perfectly legal in SQL, but then the operation takes place on every rows of the table!

# Retrieving Data with SELECT

Retrieving information from our tables is probably the operation we do most of the times. This is the way to get answers to questions like what are the cities with a population over a certain number?.

In fact, we previously did a SELECT when we clicked on the Browse link for table cities. This generated a simple form of the SELECT statement:

SELECT \* FROM `cities` LIMIT 0,30;

Here, the asterisk means all the columns. We add FROM and the name of the table which we want to query. The LIMIT 0,30 means to start at row number 0 (the first one), and select a maximum of 30 rows.

Let's try a Search to see more options for the SELECT. We go to the Search sub-page for table cities, and we choose only some columns we need.

Then at the bottom of the page, we choose to display by the result by population in descending order:

Executing the search generates the following query:

SELECT `city\_name` , `population`   
FROM `cities`   
WHERE 1   
ORDER BY `population` DESC LIMIT 0,30

We see that the asterisk has been replaced by a comma-separated list of columns. A condition **WHERE 1 has been added by phpMyAdmin, this is a condition which is always true and selects all rows**. We will see in a moment that we can replace it with some other condition. Also, the clause ORDER BY appears, followed by the column on which we want to sort results, and the keyword DESC for descending order (we could also use ASC for ascending).

**Conditions**

To easily add a condition, on the results page we can click on SQL-query: Edit, which brings the Query window popup. We add a condition on the country:

SELECT `city\_name` , `population`

FROM `cities`

WHERE country\_code = 'zh'

ORDER BY `population` DESC

which displays all cities located in China (ok, we were a bit lazy with data entry, but you get the picture).

Conditions can be expressed using a rich array of operators and functions. Here are two examples:

Finding the Canadian cities with a population over 100000:

WHERE population > 100000 AND country\_code = 'ca'

Finding the cities whose name starts with A:

WHERE city\_name like 'A%'

**Aggregate functions**

Summary information may be generated by grouping on a specific column. Here we ask the average city population per country:

SELECT country\_code, AVG(population)

FROM cities

GROUP BY country\_code

Other possible aggregate functions are MIN(), MAX(), SUM() and COUNT(), which compute respectively the minimum value, maximum value, sum of values, and number of values. For example, we could get the number of cities per country with:

SELECT country\_code, count(city\_name)

FROM cities

GROUP BY country\_code

## Joins

Normally, a relational database involves many tables, linked on common keys. We may need at times to run queries on more than one table. Linking, or joining, tables can be done using different techniques; we will focus on a simple method involving key comparison.

In the following query, the FROM clause contains a comma-separated list of tables. In the columns list, we use the table name and a dot as a prefix (name space) before each column name (not strictly necessary if each column name is only present in one table).

SELECT cities.city\_name, cities.population, countries.country\_name  
FROM cities, countries  
WHERE cities.country\_code = countries.country\_code LIMIT 0,30

**Working** **with PHP**

Use the PHP code below as basis for writing your PHP application to transpose data from your tables and interrogate them with queries. Write 10 different queries to retrieve and filter results from your table. Save the results of each query for inspection.

<!-- Program Name: sample.php: Description: PHP program that sends an SQL query to the MySQL server and displays the results.

Author ; Alexander Adu-Sarkodie-->

<html>

<head>

<title>SQL Query Sender</title>

<style>

.red {color:#ff0000;}

.gold {color:#ff9933; }

.torquoise {color:#ff00ff; }

</style>

</head>

<body>

<?php

echo "<p><span class=gold><div align=center><strong>Hello Pet World</strong></div></span></p>";

$user="root";

$host="localhost";

$password="c00k1e1";

$database = 'mysqlphp';

$query = $\_GET['query'];

$query = stripSlashes($query) ;

echo "<h3>$query</h3>";

if (!mysql\_connect($host,$user,$password)) {echo "<p>Unable to connect to database</p>";}

if (!mysql\_select\_db($database)) {echo "<p><strong>Unable to select database</span></strong>";}

$r = mysql\_query($query);

if (!$r) {echo "<p align=center><strong>Unable to query to database</strong></p>";}

else {

while ($row = mysql\_fetch\_array($r)) {

echo "<p align=center>";

echo $row['Pet\_ID']." || ";

echo $row['Description']." || ";

echo $row['Price']." || ";

echo '<img src="/lectures/images/'.$row["pix"].'.jpg">';

echo "</p>";

}

}

?>

<form action=<?=$\_SERVER['PHP\_SELF']?> method="get">

<table align="center"><tr><td align="right" valign="top"><b><span class=torquoise>Type in SQL query</span></b></td><td><textarea name="query" cols="60" rows="10" value=''><?php echo $query ?></textarea></td></tr><tr><td colspan="2" align="center"><input type="submit" value="Submit Query"></td></tr></table>

</form>

</body>

</html>