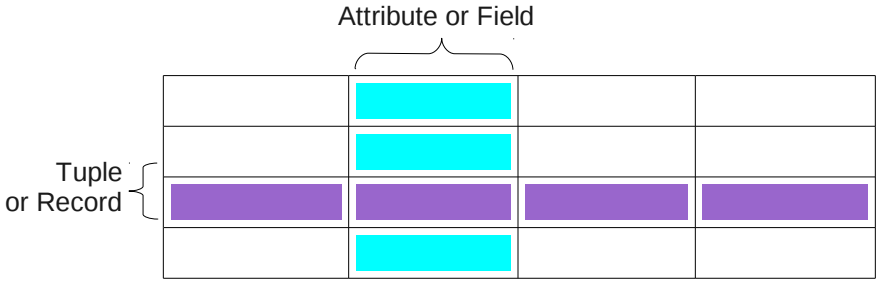
A Brief Introduction to MySQL

by Derek Schuurman

Introduction to Databases

A *database* is a structured collection of logically related data. One common type of database is the relational database, a term that was originally defined and coined by Edgar Codd in 1970.

In a relational database the data is stored in 2-dimensional tables of rows (called *tuples* or *records*) and columns (called *attributes* or *fields*). A *relation* is defined as a collection of *records* or *tuples* that have the same *fields* or *attributes*.



A Relational Database Management System (RDMS) is software used to manage and use a relational database. There are numerous commercial and open source RDMS software systems available. Some examples of open source RDMS's include MySQL amd PostgreSQL.

Structured Query Language (SQL)

Structured Query Language or SQL is a language used to interact with a relational database system and is maintained as an ISO standard. SQL provides a convenient level of abstraction to interact with a database to define, manage, and query data. SQL is a *declarative programming language* as opposed to an *imperative programming language* such as the C programming language in which all the computations are explicitly stated step-by-step. A *declarative language* is one that defines *what* the program should accomplish, rather than describing *how* to go about accomplishing it. An SQL statement is processed by the RDMS which determines the best way to return the requested data.

Although there are small differences in SQL syntax between RDMS systems, the syntax is largely the same across many systems. For the remainder of this tutorial will assume the use of MySQL.

Although there are some graphical user interfaces available for MySQL, this tutorial will focus on the command line interface for MySQL. To enter the command line interface for MySQL, type:

```
mysql -u username -p
```

Where ***username*** is the username that has been assigned to you by your system administrator. In order to use a database or create new ones, you must have sufficient privileges assigned to your username.

To list all the databases on the MySQL server, type:

```
SHOW DATABASES;
```

SQL statements include one or more SQL keywords that are often written in uppercase as a matter of style and which end with a semi-colon. To create new database in SQL from the command line, use the CREATE DATABASE statement as follows:

```
CREATE DATABASE school;
```

After executing this statement, MySQL should return a message indicating whether the command was successful or not. The SHOW DATABASES statement will show all the databases on the database server. After creating a new database, the SHOW DATABASES statement can be used to verify that the new database has been created. To show all the databases on the server, type the following:

```
SHOW DATABASES;
```

MySQL should return with a list of the current databases. The USE command is issued to select and use a specific database. For example, to use the database we just created, type:

```
USE school;
```

A database is a collection of tables. One the database selected, you can query and access the tables in the database. To create a table within the school database, you use the CREATE TABLE statement. For example, to create a table of students names type the following:

```
CREATE TABLE students (
        studentNumber int NOT NULL,
        lastName varchar(50) NOT NULL,
        firstName varchar(50) NOT NULL,
        PRIMARY KEY  (studentNumber)
      );
```

The table name is specified after CREATE TABLE statement and then the columns names are given followed by data type, size, NOT NULL or not. A field which is specified as NOT NULL must contain a value. It is also possible to specify the *primary key* of the table. The primary key is used to uniquely identify each row in the table. Since student numbers are supposed to be unique, the primary key in this example is set to **`studentNumber`**. If the table has more than one primary key, you can separate them by a comma.

In order to view details about a table, including information about fields and data types, use the DESCRIBE statement. For example, type:

```
DESCRIBE students;
```

This will return information about the table we just created and information about the fields that comprise it. The output from MySQL should resemble the following:

```
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| studentNumber | int(10)     | NO   | PRI | NULL    |       |
| lastName      | varchar(50) | NO   |     | NULL    |       |
| firstName     | varchar(50) | NO   |     | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
```

To modify the structure or type of data in existing tables, MySQL provides an ALTER command.

Although the SQL keywords themselves are not case sensitive, the database and tables may be case sensitive depending on the underlying operating system being used.

The data type of each of MySQL fields or attributes must also be specified. MySQL supports a variety of numeric, character and binary data types. Some of the common data types supported in MySQL are summarized in the following table:

Data Type	Description
INT	A signed integer (4 bytes)
FLOAT	A floating-point number (4 bytes)
DOUBLE	Double precision floating-point number (8 bytes)
DATE	Date in the format YYYY-MM-DD
DATETIME	Date and time
CHAR(M)	A fixed-length string with a length of **M** characters (between 1 to 255 character)
VARCHAR(M)	A variable-length string with a length of up to **M** characters (between 1 to 255)
TEXT	A text field with a maximum length of 65535 characters.
BLOB	A "binary large object" used to store binary data such as images.

In order to show all the tables in a database, you can use the SHOW TABLES statements as follows:

```
SHOW TABLES;
```

To add records to the students database, use the INSERT statement. For example, to add "John Calvin" with a student ID number of 12345 into the students table, do the following:

```
INSERT INTO students
(studentNumber,firstName,lastName) VALUES
(12345, "John", "Calvin");
```

MySQL Queries

You can also ask MySQL to search for data by submitting a query asking for all the records or rows that match a specific criteria. The SQL SELECT statement is used to perform queries on an SQL table.

For example, to list all the students in the table, type:

```
SELECT * FROM students;
```

The ***** indicates that all columns should be returned. The SELECT query returns the requested data as text in a tabular format like follows:

```
+---------------+----------+-----------+
| studentNumber | lastName | firstName |
+---------------+----------+-----------+
|         12345 | Calvin   | John      |
+---------------+----------+-----------+
```

To display specific columns, replace the ***** with a comma-separated list of columns that you would like to see displayed. For example, to display just the lastName and firstName columns, type:

```
SELECT lastName, firstName FROM students;
```

To list the students in alphabetical order using the ORDER BY clause as follows:

```
SELECT * FROM students ORDER BY lastName, firstName;
```

The order can be explicitly set to be ascending or descending by placing the ASC or DESC keywords at the end of the query.

To prevent the SELECT statement from returning duplicate values in the results, the DISTINCT keyword can be used. For example, to list all the distinct first names in the students table, type:

```
SELECT DISTINCT firstName FROM students;
```

It is also possible to restrict the search to meet specific criteria using the WHERE keyword. For example, to list all the students who have the first name of “John”, type:

```
SELECT * FROM students WHERE firstName = 'John';
```

The conditions to restrict search results can be further combined with boolean operators such as AND and OR operators to express search based on different conditions.

In addition to WHERE, there are other keywords such as LIKE and BETWEEN which can be used to restrict the results of a search. For example, to list all the students whose first name starts with a “J”, type:

```
SELECT * FROM students WHERE firstName LIKE 'J%';
```

where **%** is a wildcard which matches any character or sequence of characters.

The BETWEEN keyword will restrict a search to values that fall in a range between some minimum and maximum value. For example, to return all the last names that lie alphabetically between “Calvin” and “Luther”, type:

```
SELECT * FROM students WHERE lastName
BETWEEN 'Calvin' AND 'Luther';
```

One can also query the number of rows that match a certain condition using the COUNT keyword:

```
SELECT COUNT(*) FROM students WHERE firstName LIKE 'J%';
```

To delete data from a table, use the DELETE statement. For example, to delete all the students with the last name of “Calvin”, do the following:

```
DELETE FROM students WHERE lastName = 'Calvin';
```

To make another table called **courses** that stores a course code and student number for each course a student is enrolled in, use the CREATE statement again as follows:

```
CREATE TABLE courses (
        studentNumber int NOT NULL,
        courseCode varchar(7) NOT NULL );
```

In this table the **studentNumber** will not necessarily be unique since it will be appear once for each course in which a student is enrolled. The **courseCode** will also not necessarily be unique since it will be repeated for each student in the course. Note that the students names do not need to be stored again; they can be retrieved if required by looking up the **studentNumber** in the **students** table.

To add some records to the courses database, use the INSERT statement once again:

```
INSERT INTO courses (studentNumber,courseCode)
VALUES (12345, "CSC101A");
```

To change or modify data in a table, use the UPDATE keyword as follows:

```
UPDATE students SET studentNumber = 123
WHERE firstName = 'John' AND lastName = 'Calvin';
```

This statement will modify the students table and replace the studentNumber for the student with the name John Calvin. It is possible to modify multiple field values with an UPDATE statement using a comma-separated list of assignments. The WHERE clause in this case uses a boolean AND operator to make a more complex condition.

You can also ask MySQL to search data from multiple tables by using a JOIN operation. The JOIN keyword relates two or more tables, typically by using values that are common between them. The **students** and **courses** database have a common value of **studentNumber** that can be used to join them. The ON keyword can be used to specify a condition with which to join tables. For example, to list all the first names and last names of students enrolled in CSC101A, you can use a join operation based on the condition of matching a **studentNumber** in a query as follows:

```
SELECT firstName, lastName
FROM students
   JOIN courses
   ON students.studentNumber = courses.studentNumber
WHERE courseCode = 'CSC101A';
```

To close a database, type the following:

```
CLOSE DATABASE school;
```

It is also possible to delete a table and all it contents using the DROP command. This command should be used with care since it permanently deletes your table and cannot be undone.

```
DROP TABLE students;
```

Finally, you can quit MySQL at any time by typing the QUIT command.

Backing Up Your MySQL Data

You can use the **mysqldump** utility to create a simple backup of your database to a file using the following syntax:

```
mysqldump -u username -p password databasename > backup.sql
```

where ***username*** and ***password*** are your MySQL username and password and ***databasename*** is the name of the database you want to backup. The resultant file called **backup.sql** will contain all the SQL statements needed to create the table and populate the table in a new database server. If you examine the file backup.sql in a text editor you will observe the necessary SQL commands to create the database, its tables, and all the data contents within the tables. The data can be restored by typing the following:

```
mysql -u username -p password databasename < backup.sql
```

More information about MySQL can be found at: http://dev.MySQL.com

Using PHP and MySQL

Once you are familiar with MySQL syntax in the command-line environment, you can begin to write PHP code which can connect to a MySQL database and query it using SQL statements. PHP includes several functions to connect to a MySQL server and perform various queries. Some of the many PHP MySQL functions are shown in the following table:

Function	Description
`mysql_connect`	Opens a connection to a MySQL database server
`mysql_select_db`	Opens a database on the MySQL server
`mysql_query`	Performs a MySQL query on the currently selected database
`mysql_close`	Closes a MySQL database connection
`mysql_fetch_array`	Returns an associative array with the next row from a MySQL query
`mysql_error`	Returns the text of the error message from a previous MySQL operation
`mysql_create_db`	Create a MySQL database
`mysql_drop_db`	Drop a MySQL database
`mysql_real_escape_string`	This function should be used to make data entered by a user safe before sending a query to MySQL

The SELECT syntax used on the MySQL command line is the same syntax used to query data using the PHP `mysql_query` function (the query is passed as a string argument). However, before data can be queried, the proper PHP functions must be called in order to connect to the MySQL server and to select the appropriate database.

For example, the following PHP code connects to the school database and retrieves a list of students and displays it as a list within a webpage:

```
<ul>
<?php
      $db = mysql_connect("localhost", "username",
"password");
      mysql_select_db("school");
      $results = mysql_query("SELECT * FROM students");
      while ($row = mysql_fetch_array($results)) {  ?>
         <li> <?= $row["firstName"]." ".$row["lastName"] ?>
</li>
      <?php
      }
      mysql_close($db);
?>
</ul>
```

More info about PHP can be found at: http://www.PHP.net

Using MySQL with Java

MySQL can also be used with the Java Programming language. The JDBC (Java Database Connectivity) API is provides DBMS connectivity to a wide range of SQL databases including MySQL as well as access to other tabular data sources such as spreadsheets. The JDBC API includes several classes all found in the **java.sql** package.

More information about using MySQL in Java can be found at:
http://dev.mysql.com/usingmysql/java/

Exercises

Here are some exercises to help you practice using MySQL:

1) Create a new database (or use the database assigned to you by your system administrator) and make a new table called **students** to store student information including: first name, last name, gender, and student ID number (the primary key). Select appropriate data type for each of the fields in the table.
2) In the same database, create a table called **courses** that can store: course names, room number (a 3 digit number), and a course ID number (use a 7 character alpha-numeric course code such as "CSC101A" - this will also serve as the primary key), and a teacher ID.
3) Create a table called "teachers" that has a first name, last name, and teacher ID (the primary key).
4) Finally, create a table of course enrollments in a course called "courses_students" that stores course IDs and student Ids (both of these are *foreign keys*). Each course that a student is enrolled in will be stored in one row.
5) Using a tool such as **Umbrello**, draw the database schema for your tables.
6) Populate your tables with some "fake data" with 12 students, 6 courses, 3 teachers, and then populate the registrations such that several different students are enrolled in each course.
7) Devise SQL queries to determine the following:

- all the students in the school
- all the students in the school in descending alphabetical order
- the number of students enrolled in a particular course
- the number of males and females in the school
- the list of courses for a given student in alphabetical order
- the teacher name for a given course
- all the student names taught by a certain teacher in alphabetical order and such that no student names are repeated
- the number of courses taught by a given teacher

8) Using the databases you entered, write a PHP program that can list each course, the course teacher, and all the students enrolled for each and every course.

9) Backup your database using the *mysqldump* command. Open the backup file in a text editor and observe the SQL commands that can be used to restore your database.

10) Remove all the tables in your example database using the DROP command. Next, use your *mysqldump* file to restore all your data. Verify that the data has been properly restored.