Comprehensive SQL Tutorial For Beginer

By :
Oya Suryana
http://ozs.web.id

Bahan Praktikum
Computer Science Kuningan University
2013

What is SQL?

The Structured Query Language is used in manipulating data stored in Relational Database Management Systems (RDBMS). SQL provides commands through which data can be extracted, sorted, updated, deleted and inserted. SQL has the full support of ANSI (American National Standards Institute), which has laid down certain rules for the language.

SQL can be used with any RDBMS such as MySQL, mSQL, PostgresSQL, Oracle, Microsoft SQL Server, Access, Sybase, Ingres etc. All the important and common SQL statements are supported by these RDBMS, however, each has its own set of proprietary statements and extensions.

As I mentioned before, SQL is a general query language for several different kinds of databases. The SQl tutorial on this handbook is has been built around MySQL, which is a open source RDBMS available for download at MySQL.com. However, I have tried to avoid the commands specific to MySQL in this SQL tutorial. The SQL tutorial would help build a strong foundation in the SQL language and you can then port your knowledge to the specific RDBMS you plan to use.

Why MySQL?

Choosing a database system depends on three main factors; the platform on which you work, your finances and what you want to achieve. The reason I chose MySQL is because, I work mostly on the Linux system and MySQL is free for Linux. Also, I am an ardent supporter of Open Source Software movement and firmly believe that the combination of Linux, Apache, MySQL and PHP (LAMP) is hard to beat.

What are databases and why do we need them?

In simplest terms, databases are store houses of data. Suppose you have an ASCII file that contains the names of your friends and their email addresses. Would you consider this file as a database? Technically, yes! You can add, update and delete data from this file. You may also write a small program (say in Perl or C) to extract, sort and display data on the basis of some search criterion.

The need to store data is quite a fundamental to humans. Data storage and representation has been with us for thousands of years. Take the example of cave drawings (graphical data) or charting the course of seasons over the years (this would have helped in knowing the best times to sow or to migrate). Collecting data is important but more significant is the analysis of the data. A case in point is study of the movements of astronomical objects (stars, planets, the moon and the sun). Analysis of this data gave birth to two sciences, Astronomy and Astrology. Astonomy helps us predict tides, eclipses etc. Astrology on the other hand had provided loads of entertainment (!) to avid followers over the years.

Coming back to the topic at hand, storing data is of prime importance because we can then analyse the data and extract some Information.

I have spent a good deal of time in writing this SQL tutorial and if you do find it useful, I would be glad to receive your comments/suggestions/queries. Here are the things that we will cover in this tutorial:

1. Downloading MySQL

MySQL is available for Windows, Linux and other Unix variants. The Windows version is a shareware, which must registered after evaluation. I learnt SQL using this version. Later I shifted completely to Linux.... the Linux version is 100% free.

You can get MySQL from its web site.

MySQL download - the Windows version

On the download page of MySQL web site, you'll find the links to Windows version. According to the MySQL site the Windows version "contains both the Standard and Max server binaries. It also contains a version of the command-line client which uses the Cygwin library to provide command history and editing".

MySQL download - the Linux version

RPM download is recommend for Linux workstations. (Make sure you download all the RPMs; the MySQL server, client and development RPMs. Note: You need the MySQL client software for this tutorial. Check MySQL website, it might have a complete RPM package that contains all the RPMS in one download file). If you run Linux as a server, the tarball download might be better.

Once you have downloaded MySQL, let's see how to install MySQL on Windows and get it up and running.

MySQL On Package Server XAMPP - the windows version

MySQL available on package server like XAMPP or phptriad and other package. If you wil use XAMPP you can get in by download in http://apachefriends.com. And you just setup to use MySQL.
When you finished setup xampp there will be available four server like :

- Apache for web server
- MySQL for database server
- FileZilla for file server
- Mercury for mail server

Runing MySQL on Windows

Once you have successfully downloaded the Windows version, installing it is a breeze... trust me ! (The installation steps below have be tested on Win95 and Win98 using mysql-shareware-3_22_34-win.zip

1. Click start
2. Click Run
3. Type CMD, press OK
4. If Using XAMPP, Now type the following at the prompt:

   ```
   C:\xampp\mysql\bin\mysql –u root
   ```

   This starts the MySQL server.

5. The prompt is changed to the "mysql" prompt, like shown below :

   ```
   Welcome to the MySQL monitor.  Commands end with ;
   or \g.
   Your MySQL connection id is 1
   Server version: 5.1.41 Source distribution

   Type 'help;' or '\h' for help. Type '\c' to clear
   the current input statement.

   mysql>
   ```

6. To test the MySQL server, type "show databases;" at the prompt.
7. This should display something like:

   ```
   +----------------+
   | Database       |
   +----------------+
   | mysql          |
   | test           |
   +----------------+
   2 rows in set (0.00 sec)
   ```

   If you see the above... congrats... you have sucessfully installed MySQL on your system.
8. Type "quit" at the mysql prompt.
9. You are now back to the MS-DOS prompt.
10. Since our work is done (for the time being), we should shut the MySQL server. Issue the following command at the prompt.

    ```
    mysqladmin -u root shutdown
    ```

2. *MySQL beginners tutorial*

The MySQL database package consists of the following:

- The MySQL server: This is the heart of MySQL. You can consider it a program that stores and manages your databases.
- MySQL client programs: MySQL comes with many client programs. The one with which we'll be dealing a lot is called mysql (note: smallcaps). This provides an interface through which you can issue SQL statements and have the results displayed.
- MySQL client Library: This can help you in writing client programs in C. (We won't be taking about this in our tutorial).

The difference between MySQL and mysql

MySQL is used to refer to the entire MySQL distribution package or the MySQL server, while mysql refers to a client program.

Why have client and server programs?

The server and client programs are different entities. Thus, you can use client programs on your system to access data on a MySQL server running on another computer. (Note: you would need appropriate permissions for this. Consult the system administrator of the remote machine.)

Dividing the package into a server and clients separates the actual data from the interface.

3. *Creating a database*

In this section of the MySQL primer we will learn how to create a database. The commands for creating a database in Windows and Linux are the same. However, the prelimnary commands in Linux are slightly more complex. Since this tutorial is meant for the complete newbie, I'll discuss the Windows and Linux systems separately.

We'll create a database called employees that contains details of employees of our company Bignet. The details we plan to store would be names, salaries, age, addresses, emails, birth dates, hobbies, phone numbers etc.

Creating MySQL database on Windows system

1. Start the MySQL server by issuing the command mysqld-shareware --standalone at the prompt in c:\xampp\mysql\bin. Refer the previous session Installing MySQL on Windows for further details.
2. Now invoke the mysql client program by typing mysql at the prompt.
3. The prompt is changed to a **mysql**> prompt. Type:

   ```
   create database employees2;
   ```

   (Note: The command ends with a semi-colon).
4. The MySQL server responds with something like:

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

5. This means that you have sucessfully created the database. Now, let's see how many databases you have on your system. Issue the following command.

   ```
   show databases;
   ```

   The server responds with the list of databases.

   ```
   +----------------+
   | Database       |
   +----------------+
   | employees      |
   | mysql          |
   | test           |
   +----------------+
   3 rows in set (0.00 sec)
   ```

   Here we have three databases, two created by MySQL during installation and our employees database.

6. To come back to the DOS prompt, type quit at the mysql prompt.

4. Creating tables

In this section of the mysql training course we will explore the MySQL commands to create database tables and selecting the database.

Databases store data in tables. So what are these tables ? In simplest terms, tables consist of rows and columns. Each column defines data of a particular type. Rows contain individual records. Consider the following :

Name	Age	Country	Email
Manish Sharma	28	India	manish@simplygraphix.com
John Doe	32	Australia	j.dow@nowhere.com
John Wayne	48	U.S.A.	jw@oldwesterns.com
Alexander	19	Greece	alex@conqueror.com

The table above contains four columns that store the name, age, country and email. Each row contains data for one individual. This is called a record. To find the country and email of Alexander, you'd first pick the name from the first column and and then look in the third and fourth columns of the same row.

A database can have many tables; it is tables, that contain the actual data. Hence, we can segregate related (or unrelated) data in different tables. For our employees database we'll have one table that stores company details of the employees. The other table would contain personal information. Let's make the first table.

The SQL command for creating tables looks complex when you view it for the first time. Don't worry if you get confused, we'll be discussing this in more detail in later sessions.

```
CREATE TABLE employee_data
(
emp_id int unsigned not null auto_increment primary key,
f_name varchar(20),
l_name varchar(20),
title varchar(30),
age int,
yos int,
salary int,
perks int,
email varchar(60)
);
```

Note : In MySQL, commands and column names are not case-sensitive; however, table and database names might be sensitive to case depending on the platform (as in Linux). You can thus, use create table instead of **CREATE TABLE**.

The **CREATE TABLE** keywords are followed by the name of the table we want to create, employee_data. Each line inside the parenthesis represents one column. These columns store the employee id, first name, last name, title, age, years of service with the company, salary, perks and emails of our employees and are given descriptive names emp_id, f_name, l_name, title, age, yos, salary, perks and email, respectively. Each column name is followed by the column type. Column types define the type of data the column is set to contain. In our example, columns, f_name, l_name, title and email would contain small text strings, so we set the column type to varchar, which means varriable characters. The maximum number of characters for varchar columns is specified by a number enclosed in parenthesis immediately following the column name. Columns age, yos, salary and perks would contain numbers (integers), so we set the column type to int.

Our first column (emp_id) contains an employee id. Its column type looks really mean, yeh?. Let's break it down.

int	:	specifies that the column type is an integer (a number).
unsigned	:	determines that the number will be unsigned (positive integer).
not null	:	specifies that the value cannot be null (empty); that is, each row in the column would have a value.
auto_increment	:	When MySQl comes across a column with an auto_increment attribute, it generates a new value that is one greater than the largest value in the column. Thus, we don't need to supply values for this column, MySQL generates it for us! Also, it follows that each value in this column would be unique. (We'll discuss the benefits of having unique values very shortly).
primary key	:	helps in indexing the column that help in faster searches. Each value has to be unique.

Why have a column with unique values?

Our company Oznet has grown tremendously over the past two years. We've recruited thousands. Don't you think there is a fair chance that two employees might have the same name? Now, when that happens, how can we distinguish the records of these two employees unless we give them unique identification numbers? If we have a column with unique values, we can easily distinguish the two records. The best way to assign unique numbers is to let MySQL do it!

Using a database

We've already created our employees database. Now let's start the mysql client program and select our database. Once at the mysql prompt, issue the command:
SELECT DATABASE();

The system responds with

```
mysql> SELECT DATABASE();
+------------+
| DATABASE() |
+------------+
|            |
+------------+
1 row in set (0.01 sec)
```

The above shows that no database has been selected. Actually, everytime we work with mysql client, we have to specify which database we plan to use. There are several ways of doing it.

Specifying the database name at the start; type the folowing at the system prompt:

```
mysql employees (under Windows)
```

Specifying the database with the USE statement at the mysql prompt:

```
mysql>USE employees;
```

Specifying the database with \u at the mysql prompt:

```
mysql>\u employees;
```

It's necessary to specify the database we plan to use, else MySQL will throw an error.

Creating tables

Once you've selected the employees database, issue the **CREATE TABLE** command at the mysql prompt.

```
CREATE TABLE employee_data
(
emp_id int unsigned not null auto_increment primary key,
f_name varchar(20),
l_name varchar(20),
title varchar(30),
age int,
yos int,
salary int,
perks int,
email varchar(60)
);
```

Note : When you press the enter key after typing the first line, the mysql prompt changes to a ->. This means that mysql understands that the command is not complete and prompts you for additional statements. Remember, each mysql command ends with a semi-colon and each column declaration is separated by a comma. Also, you can type the entire command on one line if you so want.
You screen should look similar to:

```
mysql> CREATE TABLE employee_data
    -> (
    -> emp_id int unsigned not null auto_increment primary key,
    -> f_name varchar(20),
    -> l_name varchar(20),
    -> title varchar(30),
    -> age int,
    -> yos int,
    -> salary int,
    -> perks int,
    -> email varchar(60)
    -> );
Query OK, 0 rows affected (0.01 sec)
```

Okay, we just made our first table.

5. *MySQL tables*

Now that we've created our employee_data table, let's check its listing.
Type SHOW TABLES; at the mysql prompt. This should present you with the following display:

```
mysql> SHOW TABLES;
+---------------------+
| Tables in employees |
+---------------------+
| employee_data       |
+---------------------+
1 row in set (0.00 sec)
```

MySQL Table details - describe table command

MySQL provides up with a command that displays the column details of the tables.
Issue the following command at the mysql prompt:
DESCRIBE employee_data;

The display would be as follows:

```
mysql> DESCRIBE employee_data;
+--------+------------------+------+-----+---------+----------------+
| Field  | Type             | Null | Key | Default | Extra          |
+--------+------------------+------+-----+---------+----------------+
| emp_id | int(10) unsigned |      | PRI | 0       | auto_increment |
| f_name | varchar(20)      | YES  |     | NULL    |                |
| l_name | varchar(20)      | YES  |     | NULL    |                |
| title  | varchar(30)      | YES  |     | NULL    |                |
| age    | int(11)          | YES  |     | NULL    |                |
| yos    | int(11)          | YES  |     | NULL    |                |
| salary | int(11)          | YES  |     | NULL    |                |
| perks  | int(11)          | YES  |     | NULL    |                |
| email  | varchar(60)      | YES  |     | NULL    |                |
+--------+------------------+------+-----+---------+----------------+
9 rows in set (0.00 sec)
```

DESCRIBE lists all the column names along with their column types of the table.
Now let's see how we can insert data into our table.

6. *Inserting data in MySQL tables*

To Inserting data into tables you can use INSERT SQL statement impregnates our table with data. Here is a general form of INSERT.

```
INSERT into table_name (column1, column2....)
values (value1, value2...);
```

where table_name is the name of the table into which we want to insert data; column1, column2 etc. are column names and value1, value2 etc. are values for the respective columns. This is quite simple, isn't it?

The following statement inserts the first record in employee_data table.

```
INSERT INTO employee_data
(f_name, l_name, title, age, yos, salary, perks, email)
values
("Manish", "Sharma", "CEO", 28, 4, 200000,
50000, "manish@bignet.com");
```

As with other MySQL statements, you can enter this command on one line or span it in multiple lines. Some important points:

- The table name is employee_data
- The values for columns f_name, l_name, title and email are text strings and surrounded with quotes.
- Values for age, yos, salary and perks are numbers (intergers) and without quotes.
- You'll notice that we've inserted data in all columns except emp_id. This is because, we leave this job to MySQL, which will check the column for the largest value, increment it by one and insert the new value.

Once you type the above command correctly in the mysql client, it displays a success message.

```
mysql> INSERT INTO employee_data
    -> (f_name, l_name, title, age, yos, salary, perks, email)
    -> values
    -> ("Manish", "Sharma", "CEO", 28, 4, 200000,
    -> 50000, "manish@bignet.com");
Query OK, 1 row affected (0.00 sec)
```

Inserting additional records requires separate INSERT statements. In order to make life easy, I've packed all INSERT statements into a file. Buatlah fil employee.dat.
Once you download the file, open it in a text editor. You'll notice that it's a plain ASCII file with an INSERT statement on each line.

Inserting data into employee_data table with employee.dat file

1. Move the file to c:\XAMPP\MYSQL\bin.
2. Make sure MySQL is running.
3. Issue the following command

```
        mysql employees <employee.dat
```

Our table contains 21 entries (20 from employee.dat file and one from the INSERT statement we issued at the beginning). You can view the table here. (This opens another browser window).

```
+--------+---------+------------+----------------------------+------+------+--------+-------+-------
| emp_id | f_name  | l_name     | title                      | age  | yos  | salary | perks | email
+--------+---------+------------+----------------------------+------+------+--------+-------+-------
|      1 | Manish  | Sharma     | CEO                        |   28 |    4 | 200000 | 50000 | manish
|      2 | John    | Hagan      | Senior Programmer          |   32 |    4 | 120000 | 25000 | john_h
|      3 | Ganesh  | Pillai     | Senior Programmer          |   32 |    4 | 110000 | 20000 | g_pill
|      4 | Anamika | Pandit     | Web Designer               |   27 |    3 |  90000 | 15000 | ana@bi
|      5 | Mary    | Anchor     | Web Designer               |   26 |    2 |  85000 | 15000 | mary@b
|      6 | Fred    | Kruger     | Programmer                 |   31 |    3 |  75000 | 15000 | fk@big
|      7 | John    | MacFarland | Programmer                 |   34 |    4 |  80000 | 16000 | john@b
|      8 | Edward  | Sakamuro   | Programmer                 |   25 |    2 |  75000 | 14000 | eddie@
|      9 | Alok    | Nanda      | Programmer                 |   32 |    3 |  70000 | 10000 | alok@b
|     10 | Hassan  | Rajabi     | Multimedia Programmer      |   33 |    3 |  90000 | 15000 | hasan@
|     11 | Paul    | Simon      | Multimedia Programmer      |   43 |    2 |  85000 | 12000 | ps@big
|     12 | Arthur  | Hoopla     | Multimedia Programmer      |   32 |    1 |  75000 | 15000 | arthur
|     13 | Kim     | Hunter     | Senior Web Designer        |   32 |    2 | 110000 | 20000 | kim@bi
|     14 | Roger   | Lewis      | System Administrator       |   35 |    2 | 100000 | 13000 | roger@
|     15 | Danny   | Gibson     | System Administrator       |   34 |    1 |  90000 | 12000 | danny@
|     16 | Mike    | Harper     | Senior Marketing Executive |   36 |    2 | 120000 | 28000 | mike@b
|     17 | Monica  | Sehgal     | Marketing Executive        |   30 |    3 |  90000 | 25000 | monica
|     18 | Hal     | Simlai     | Marketing Executive        |   27 |    2 |  70000 | 18000 | hal@bi
|     19 | Joseph  | Irvine     | Marketing Executive        |   27 |    2 |  72000 | 18000 | joseph
|     20 | Shahida | Ali        | Customer Service Manager   |   32 |    3 |  70000 |  9000 | shahid
|     21 | Peter   | Champion   | Finance Manager            |   36 |    4 | 120000 | 25000 | peter@
+--------+---------+------------+----------------------------+------+------+--------+-------+-------
```

7. Querying MySQL tables

Our employee_data table now contains enough data for us to work with. Let us see how we can extract (query) it. Querying involves the use of the MySQL SELECT command.

Data is extracted from the table using the SELECT SQL command. Here is the format of a SELECT statement:

```
SELECT column_names from table_name [WHERE ...conditions];
```

The conditions part of the statement is optional (we'll go through this later). Basically, you require to know the column names and the table name from which to extract the data.

For example, in order to extract the first and last names of all employees, issue the following command.

```
SELECT f_name, l_name from employee_data;
```

The statement tells MySQL to list all the rows from columns f_name and l_name.

```
mysql> SELECT f_name, l_name from employee_data;
+---------+------------+
| f_name  | l_name     |
+---------+------------+
| Manish  | Sharma     |
| John    | Hagan      |
| Ganesh  | Pillai     |
| Anamika | Pandit     |
| Mary    | Anchor     |
| Fred    | Kruger     |
| John    | MacFarland |
| Edward  | Sakamuro   |
| Alok    | Nanda      |
| Hassan  | Rajabi     |
| Paul    | Simon      |
| Arthur  | Hoopla     |
| Kim     | Hunter     |
| Roger   | Lewis      |
| Danny   | Gibson     |
| Mike    | Harper     |
| Monica  | Sehgal     |
| Hal     | Simlai     |
| Joseph  | Irvine     |
| Shahida | Ali        |
| Peter   | Champion   |
+---------+------------+
21 rows in set (0.00 sec)
```

On close examination, you'll find that the display is in the order in which the data was inserted. Furthermore, the last line indicates the number of rows our table has (21).

To display the entire table, we can either enter all the column names or use a simpler form of the SELECT statement.

```
SELECT * from employee_data;
```

Some of you might recognize the * in the above statement as the wildcard. Though we don't use that term for the character here, it serves a very similar function. The * means 'ALL columns'. Thus, the above statement lists all the rows of all columns.
Querying tables with MySQL Select - Another example
SELECT f_name, l_name, age from employee_data;

Selecting f_name, l_name and age columns would display something like:

```
mysql> SELECT f_name, l_name, age from employee_data;
+---------+-----------+------+
| f_name  | l_name    | age  |
+---------+-----------+------+
| Manish  | Sharma    |   28 |
| John    | Hagan     |   32 |
| Ganesh  | Pillai    |   32 |
| Anamika | Pandit    |   27 |
| Mary    | Anchor    |   26 |
| Fred    | Kruger    |   31 |
| John    | MacFarland |   34 |
| Edward  | Sakamuro  |   25 |
| Alok    | Nanda     |   32 |
| Hassan  | Rajabi    |   33 |
| Paul    | Simon     |   43 |
| Arthur  | Hoopla    |   32 |
| Kim     | Hunter    |   32 |
| Roger   | Lewis     |   35 |
| Danny   | Gibson    |   34 |
| Mike    | Harper    |   36 |
| Monica  | Sehgal    |   30 |
| Hal     | Simlai    |   27 |
| Joseph  | Irvine    |   27 |
| Shahida | Ali       |   32 |
| Peter   | Champion  |   36 |
+---------+-----------+------+
21 rows in set (0.00 sec)
```

ASSIGNMENTS

1. Write the complete SQL statement for creating a new database called **addressbook**
2. Which statement is used to list the information about a table? How do you use this statement?
3. How would you list all the databases available on the system?
4. Write the statement for inserting the following data in employee_data table
   First name: Rudolf
   Last name: Reindeer
   Title: Business Analyst
   Age: 34
   Years of service: 2
   Salary: 95000
   Perks: 17000
   email: rudolf@bugnet.com

5. Give two forms of the SELECT statement that will list all the data in employee_data table.
6. What will **select f_name, email from employee_data;** display?
7. Write the statement for listing data from **salary, perks** and **yos** columns of **employee_data** table.
8. How can you find the number of rows in a table using the SELECT statement?
9. What will select **salary, l_name from employee_data;** display?

8. *selecting data using conditions*

In this section of the MySQL tutorial we'll look at the format of a SELECT statement we met in the last session in detail. We will learn how to use the select statement using the WHERE clause.

```
SELECT column_names from table_name [WHERE ...conditions];
```

Now, we know that the conditions are optional (we've seen several examples in the last session... and you would have encountered them in the assignments too).

The SELECT statement without conditions lists all the data in the specified columns. The strength of RDBMS lies in letting you retrieve data based on certain specified conditions. In this session we'll look at the SQL Comparision Operators. The = and != comparision operators for MySQL Select.

```
SELECT f_name, l_name from employee_data where f_name =
'John';

+--------+------------+
| f_name | l_name     |
+--------+------------+
| John   | Hagan      |
| John   | MacFarland |
+--------+------------+
2 rows in set (0.00 sec)
```

This displays the first and last names of all employees whose first names are John. Note that the word John in the condition is surrounded by single quotes. You can also use double quotes. The quotes are important since MySQL will throw an error if they are missing. Also, MySQL comparisions are case insensitive; which means "john", "John" or even "JoHn" would work!

```
SELECT f_name,l_name from employee_data where title="Programmer";

+--------+-----------+
| f_name | l_name    |
+--------+-----------+
| Fred   | Kruger    |
| John   | MacFarland |
| Edward | Sakamuro  |
| Alok   | Nanda     |
+--------+-----------+
4 rows in set (0.00 sec)
```

Selects the first and last names of all employees who are programmers.

```
SELECT f_name, l_name from employee_data where age = 32;
+---------+--------+
| f_name  | l_name |
+---------+--------+
| John    | Hagan  |
| Ganesh  | Pillai |
| Alok    | Nanda  |
| Arthur  | Hoopla |
| Kim     | Hunter |
| Shahida | Ali    |
+---------+--------+
6 rows in set (0.00 sec)
```

This lists the first and last names of all empoyees 32 years of age. Remember that the column type of age was int, hence it's not necessary to surround 32 with quotes. This is a subtle difference between text and integer column types.

The != means 'not equal to' and is the opposite of the equality operator. The greater than and lesser than operators Okay, let's retrieve the first names of all employees who are older than 32.

```
SELECT f_name, l_name from employee_data where age > 32;
+--------+-----------+
| f_name | l_name    |
+--------+-----------+
| John   | MacFarland |
| Hassan | Rajabi    |
| Paul   | Simon     |
| Roger  | Lewis     |
| Danny  | Gibson    |
| Mike   | Harper     |
| Peter  | Champion   |
+--------+-----------+
7 rows in set (0.00 sec)
```

How about employees who draw more than $120000 as salary...

```
SELECT  f_name,  l_name  from  employee_data  where  salary  >
120000;
+--------+--------+
| f_name | l_name |
+--------+--------+
| Manish | Sharma |
+--------+--------+
1 row in set (0.00 sec)
```

Now, let's list all employees who have had less than 3 years of service in the company.

```
SELECT f_name, l_name from employee_data where yos < 3;
+--------+----------+
| f_name | l_name   |
+--------+----------+
| Mary   | Anchor   |
| Edward | Sakamuro |
| Paul   | Simon    |
| Arthur | Hoopla   |
| Kim    | Hunter   |
| Roger  | Lewis    |
| Danny  | Gibson   |
| Mike   | Harper   |
| Hal    | Simlai   |
| Joseph | Irvine   |
+--------+----------+
10 rows in set (0.00 sec)
```

The <= and >= operators for selecting MySQL data. Used primarily with integer data, the less than equal (<=) and greater than equal (>=)operators provide additional functionality.

```
select  f_name,  l_name,  age,  salary  from  employee_data  where
age >= 33;

+--------+-----------+------+--------+
| f_name | l_name    | age  | salary |
+--------+-----------+------+--------+
| John   | MacFarland |   34 |  80000 |
| Hassan | Rajabi    |   33 |  90000 |
| Paul   | Simon     |   43 |  85000 |
| Roger  | Lewis     |   35 | 100000 |
| Danny  | Gibson    |   34 |  90000 |
| Mike   | Harper    |   36 | 120000 |
| Peter  | Champion  |   36 | 120000 |
+--------+-----------+------+--------+
7 rows in set (0.00 sec)
```

Selects the names, ages and salaries of employees who are more than or equal to 33 years of age..

```
select f_name, l_name from employee_data where yos <= 2;
+--------+----------+
| f_name | l_name   |
+--------+----------+
| Mary   | Anchor   |
| Edward | Sakamuro |
| Paul   | Simon    |
| Arthur | Hoopla   |
| Kim    | Hunter   |
| Roger  | Lewis    |
| Danny  | Gibson   |
| Mike   | Harper   |
| Hal    | Simlai   |
| Joseph | Irvine   |
+--------+----------+
10 rows in set (0.00 sec)
```

Displays employee names who have less than or equal to 2 years of service in the company

ASSIGNMENTS

1. Write the SELECT statement to extract the ids of employees who are more than 30 years of age.
2. Write the SELECT statement to extract the first and last names of all web designers.
3. What will the following SELECT statement display:
4. SELECT * from employee_data where salary <=100000;
5. How will you display the salaries and perks for employees who have more than $16000 as perks?
6. List all employee names (last name followed by first name) who hold the title of Marketing Executive.

9. Pattern Matching with text data

We will now learn at how to match text patterns using the where clause and the LIKE operator in this section of the MySQL reference guide.

The equal to(=) comparision operator helps is selecting strings that are identical. Thus, to list the names of employees whose first names are John, we can use the following SELECT statement.

```
select f_name, l_name from employee_data where f_name = "John";

+--------+------------+
| f_name | l_name     |
+--------+------------+
| John   | Hagan      |
| John   | MacFarland |
+--------+------------+
2 rows in set (0.00 sec)
```

What if we wanted to display employees whose first names begin with the alphabet J? SQL allows for some pattern matching with string data. Here is how it works.

```
select f_name, l_name from employee_data where f_name LIKE "J%";

+--------+------------+
| f_name | l_name     |
+--------+------------+
| John   | Hagan      |
| John   | MacFarland |
| Joseph | Irvine     |
+--------+------------+
3 rows in set (0.00 sec)
```

You'll notice that we've replaced the Equal To sign with LIKE and we've used a percentage sign (%) in the condition. The % sign functions as a wildcard (similar to the usage of * in DOS and Linux systems). It signifies any character. Thus, "J%" means all strings that begin with the alphabet J. Similarly "%S" selects strings that end with S and "%H%", strings that contain the alphabet H. Okay, let's list all the employees that have Senior in their titles.

```
select f_name, l_name, title from employee_data
where title like '%senior%';

+--------+--------+---------------------------+
| f_name | l_name | title                     |
+--------+--------+---------------------------+
| John   | Hagan  | Senior Programmer         |
| Ganesh | Pillai | Senior Programmer         |
| Kim    | Hunter | Senior Web Designer       |
| Mike   | Harper | Senior Marketing Executive |
+--------+--------+---------------------------+
4 rows in set (0.00 sec)
```

Listing all employees whose last names end with A is very simple

```
mysql> select f_name, l_name from employee_data
where l_name like '%a';

+--------+--------+
| f_name | l_name |
+--------+--------+
| Manish | Sharma |
| Alok   | Nanda  |
| Arthur | Hoopla |
+--------+--------+
3 rows in set (0.00 sec)
```

ASSIGNMENTS

1. List all employees whose last names begin with P.
2. Display the names of all employees in the marketing division.
3. What will the following statement display
4. SELECT f_name, l_name, salary from employee_data where f_name like '%k%';
5. List the last names and titles of all programmers

10. Logical Operators

In this section of the SQL primer we look at how to select data based on certain conditions presented through MySQL logical operators. SQL conditions can also contain Boolean (logical) operators. They are :

a. AND
b. OR
c. NOT

Their usage is quite simple. Here is a SELECT statement that lists the names of employees who draw more than $70000 but less than $90000.

```
SELECT f_name, l_name from employee_data
where salary > 70000 AND salary < 90000;

+--------+-----------+
| f_name | l_name    |
+--------+-----------+
| Mary   | Anchor    |
| Fred   | Kruger    |
| John   | MacFarland |
| Edward | Sakamuro  |
| Paul   | Simon     |
| Arthur | Hoopla    |
| Joseph | Irvine    |
+--------+-----------+
7 rows in set (0.00 sec)
```

Let's display the last names of employees whose last names start with the alphabet S or A.

```
SELECT l_name from employee_data where l_name like 'S%' OR l_name
like 'A%';

+----------+
| l_name   |
+----------+
| Sharma   |
| Anchor   |
| Sakamuro |
| Simon    |
| Sehgal   |
| Simlai   |
| Ali      |
+----------+
7 rows in set (0.00 sec)
```

Okay here is a more complex example... listing the names and ages of employees whose last names begin with S or P and who are less than 30 years of age.

```
SELECT f_name, l_name , age from employee_data
where (l_name like 'S%' OR l_name like 'A%') AND
age < 30;

+--------+----------+------+
| f_name | l_name   | age  |
+--------+----------+------+
| Manish | Sharma   |   28 |
| Mary   | Anchor   |   26 |
| Edward | Sakamuro |   25 |
| Hal    | Simlai   |   27 |
+--------+----------+------+
4 rows in set (0.00 sec)
```

Note the usage of parenthesis in the statement above. The parenthesis are meant to separate the various logical conditions and remove any abiguity. The NOT operator helps in listing all non programmers. (Programmers include Senior programmers, Multimedia Programmers and Programmers).

```
SELECT f_name, l_name, title from employee_data
where title NOT LIKE "%programmer%";

+---------+----------+----------------------------+
| f_name  | l_name   | title                      |
+---------+----------+----------------------------+
| Manish  | Sharma   | CEO                        |
| Anamika | Pandit   | Web Designer               |
| Mary    | Anchor   | Web Designer               |
| Kim     | Hunter   | Senior Web Designer        |
| Roger   | Lewis    | System Administrator       |
| Danny   | Gibson   | System Administrator       |
| Mike    | Harper   | Senior Marketing Executive |
| Monica  | Sehgal   | Marketing Executive        |
| Hal     | Simlai   | Marketing Executive        |
| Joseph  | Irvine   | Marketing Executive        |
| Shahida | Ali      | Customer Service Manager   |
| Peter   | Champion | Finance Manager            |
+---------+----------+----------------------------+
12 rows in set (0.00 sec)
```

A final example before we proceed to the assignments. Displaying all employees with more than 3 years or service and more than 30 years of age.

```
select f_name, l_name from employee_data
where yos > 3 AND age > 30;

+--------+-----------+
| f_name | l_name    |
+--------+-----------+
| John   | Hagan     |
| Ganesh | Pillai    |
| John   | MacFarland |
| Peter  | Champion   |
+--------+------------+
4 rows in set (0.00 sec)
```

ASSIGNMENTS

1. List the first and last names of all employees who draw less than or equal to $90000 and are not Programmes, Senior programmers or Multimedia programmers.
2. What is displayed by the following statement ?

   ```
   SELECT l_name, f_name from employee_data where title
   NOT LIKE '%marketing%' AND age < 30;
   ```

3. List all ids and names of all employees between 32 and 40 years of age.
4. Select names of all employees who are 32 years of age and are not programmers.

11. IN and BETWEEN

This section of the tutorial MySQL looks at the In and BETWEEN operators To list employees who are Web Designers and System Administrators, we use a SELECT statement as

```
SELECT f_name, l_name, title from
    -> employee_data where
    -> title = 'Web Designer' OR
    -> title = 'System Administrator';

+---------+--------+----------------------+
| f_name  | l_name | title                |
+---------+--------+----------------------+
| Anamika | Pandit | Web Designer         |
| Mary    | Anchor | Web Designer         |
| Roger   | Lewis  | System Administrator |
| Danny   | Gibson | System Administrator |
+---------+--------+----------------------+
4 rows in set (0.01 sec)
```

SQL also provides an easier method with IN. Its usage is quite simple.

```
SELECT f_name, l_name, title from
    -> employee_data where title
    -> IN ('Web Designer', 'System Administrator');

+---------+--------+----------------------+
| f_name  | l_name | title                |
+---------+--------+----------------------+
| Anamika | Pandit | Web Designer         |
| Mary    | Anchor | Web Designer         |
| Roger   | Lewis  | System Administrator |
| Danny   | Gibson | System Administrator |
+---------+--------+----------------------+
4 rows in set (0.00 sec)
```

Suffixing NOT to IN will display data that is NOT found IN the condition. The following lists employees who hold titles other than Programmer and Marketing Executive.

```
SELECT f_name, l_name, title from
    -> employee_data where title NOT IN
    -> ('Programmer', 'Marketing Executive');

+---------+----------+---------------------------+
| f_name  | l_name   | title                     |
+---------+----------+---------------------------+
| Manish  | Sharma   | CEO                       |
| John    | Hagan    | Senior Programmer         |
| Ganesh  | Pillai   | Senior Programmer         |
| Anamika | Pandit   | Web Designer              |
| Mary    | Anchor   | Web Designer              |
| Hassan  | Rajabi   | Multimedia Programmer     |
| Paul    | Simon    | Multimedia Programmer     |
| Arthur  | Hoopla   | Multimedia Programmer     |
| Kim     | Hunter   | Senior Web Designer       |
| Roger   | Lewis    | System Administrator      |
| Danny   | Gibson   | System Administrator      |
| Mike    | Harper   | Senior Marketing Executive |
| Shahida | Ali      | Customer Service Manager  |
| Peter   | Champion | Finance Manager           |
+---------+----------+---------------------------+
14 rows in set (0.00 sec)
```

BETWEEN is employed to specify integer ranges. Thus instead of age >= 32 AND age <= 40, we can use age BETWEEN 32 and 40.

```
select f_name, l_name, age from
    -> employee_data where age BETWEEN
    -> 32 AND 40;

+---------+------------+------+
| f_name  | l_name     | age  |
+---------+------------+------+
| John    | Hagan      |   32 |
| Ganesh  | Pillai     |   32 |
| John    | MacFarland |   34 |
| Alok    | Nanda      |   32 |
| Hassan  | Rajabi     |   33 |
| Arthur  | Hoopla     |   32 |
| Kim     | Hunter     |   32 |
| Roger   | Lewis      |   35 |
| Danny   | Gibson     |   34 |
| Mike    | Harper     |   36 |
| Shahida | Ali        |   32 |
| Peter   | Champion   |   36 |
+---------+------------+------+
12 rows in set (0.00 sec)
```

You can use NOT with BETWEEN as in the following statement that lists employees who draw salaries less than $90000 and more than $150000.

```
select f_name, l_name, salary
    -> from employee_data where salary
    -> NOT BETWEEN
    -> 90000 AND 150000;

+---------+-----------+--------+
| f_name  | l_name    | salary |
+---------+-----------+--------+
| Manish  | Sharma    | 200000 |
| Mary    | Anchor    |  85000 |
| Fred    | Kruger    |  75000 |
| John    | MacFarland|  80000 |
| Edward  | Sakamuro  |  75000 |
| Alok    | Nanda     |  70000 |
| Paul    | Simon     |  85000 |
| Arthur  | Hoopla    |  75000 |
| Hal     | Simlai    |  70000 |
| Joseph  | Irvine    |  72000 |
| Shahida | Ali       |  70000 |
+---------+-----------+--------+
11 rows in set (0.00 sec)
```

ASSIGNMENTS

1. List all employees who hold the titles of "Senior Programmer" and "Multimedia Programmer".
2. List all employee names with salaries for employees who draw between $70000 and $90000.
3. What will the following statement display?

   ```
   SELECT f_name, l_name, title from
   employee_data where title NOT IN
   ('Programmer', 'Senior Programmer',
   'Multimedia Programmer');
   ```

4. Here is a more complex statement that combines both BETWEEN and IN. What will it display?

   ```
   SELECT f_name, l_name, title, age
   from employee_data where
   title NOT IN
   ('Programmer', 'Senior Programmer',
   'Multimedia Programmer') AND age
   NOT BETWEEN 28 and 32;
   ```

12. Ordering data

This section of the online MySQL tutorial looks at how we can change the display order of the data extracted from MySQL tables using the ORDER BY clause of the SELECT statement.

The data that we have retrieved so far was always displayed in the order in which it was stored in the table. Actually, SQL allows for sorting of retrieved data with the ORDER BY clause. This clause requires the column name based on which the data will be sorted. Let's see how to display employee names with last names sorted alphabetically (in ascending order).

```
SELECT l_name, f_name from
employee_data ORDER BY l_name;

+------------+---------+
| l_name     | f_name  |
+------------+---------+
| Ali        | Shahida |
| Anchor     | Mary    |
| Champion   | Peter   |
| Gibson     | Danny   |
| Hagan      | John    |
| Harper     | Mike    |
| Hoopla     | Arthur  |
| Hunter     | Kim     |
| Irvine     | Joseph  |
| Kruger     | Fred    |
| Lewis      | Roger   |
| MacFarland | John    |
| Nanda      | Alok    |
| Pandit     | Anamika |
| Pillai     | Ganesh  |
| Rajabi     | Hassan  |
| Sakamuro   | Edward  |
| Sehgal     | Monica  |
| Sharma     | Manish  |
| Simlai     | Hal     |
| Simon      | Paul    |
+------------+---------+
21 rows in set (0.00 sec)
```

Here are employees sorted by age.

```
SELECT f_name, l_name, age
from employee_data
ORDER BY age;

+---------+------------+------+
| f_name  | l_name     | age  |
+---------+------------+------+
| Edward  | Sakamuro   |   25 |
| Mary    | Anchor     |   26 |
| Anamika | Pandit     |   27 |
| Hal     | Simlai     |   27 |
| Joseph  | Irvine     |   27 |
| Manish  | Sharma     |   28 |
| Monica  | Sehgal     |   30 |
| Fred    | Kruger     |   31 |
| John    | Hagan      |   32 |
| Ganesh  | Pillai     |   32 |
| Alok    | Nanda      |   32 |
| Arthur  | Hoopla     |   32 |
| Kim     | Hunter     |   32 |
| Shahida | Ali        |   32 |
| Hassan  | Rajabi     |   33 |
| John    | MacFarland |   34 |
| Danny   | Gibson     |   34 |
| Roger   | Lewis      |   35 |
| Mike    | Harper     |   36 |
| Peter   | Champion   |   36 |
| Paul    | Simon      |   43 |
+---------+------------+------+
21 rows in set (0.00 sec)
```

The ORDER BY clause can sort in an ASCENDING (ASC) or DESCENDING (DESC) order depending upon the argument supplied. To list employee first names in descending order, we'll use the statement below.

```
SELECT f_name from employee_data
ORDER by f_name DESC;

+---------+
| f_name  |
+---------+
| Shahida |
| Roger   |
| Peter   |
| Paul    |
| Monica  |
| Mike    |
| Mary    |
| Manish  |
| Kim     |
| Joseph  |
| John    |
| John    |
| Hassan  |
| Hal     |
| Ganesh  |
| Fred    |
| Edward  |
| Danny   |
| Arthur  |
| Anamika |
| Alok    |
+---------+
21 rows in set (0.00 sec)
```

Note: The ascending (ASC) order is the default.

ASSIGNMENTS

1. Order all employees on the basis of the salary they draw.
2. List all employees in descending order of their years of service.
3. What does the following statement display?
```
SELECT emp_id, l_name, title, age
from employee_data ORDER BY
title DESC, age ASC;
```
4. Display employees (last names followed by first names) who hold the title of either "Programmer" or "Web Designer" and sort their last names alphabetically.

13. *Limiting data retrieval*

This section of the online MySQL lesson looks at how to limit the number of records displayed by the SELECT statement.

As your tables grow, you'll find a need to display only a subset of data. This can be achieved with the LIMIT clause.
For example, to list only the names of first 5 employees in our table, we use LIMIT with 5 as argument.

```
SELECT f_name, l_name from
employee_data LIMIT 5;

+---------+--------+
| f_name  | l_name |
+---------+--------+
| Manish  | Sharma |
| John    | Hagan  |
| Ganesh  | Pillai |
| Anamika | Pandit |
| Mary    | Anchor |
+---------+--------+
5 rows in set (0.01 sec)
```

These are the first five entries in our table. You can couple LIMIT with ORDER BY. Thus, the following displays the 4 senior most employees.

```
SELECT f_name, l_name, age from
employee_data ORDER BY age DESC
LIMIT 4;

+--------+----------+------+
| f_name | l_name   | age  |
+--------+----------+------+
| Paul   | Simon    |   43 |
| Mike   | Harper   |   36 |
| Peter  | Champion |   36 |
| Roger  | Lewis    |   35 |
+--------+----------+------+
4 rows in set (0.00 sec)
```

Cool, yeh? Similarly, we can list the two youngest employees.

```
SELECT f_name, l_name, age from
employee_data ORDER BY age
LIMIT 2;

+--------+----------+------+
| f_name | l_name   | age  |
+--------+----------+------+
| Edward | Sakamuro |   25 |
| Mary   | Anchor   |   26 |
+--------+----------+------+
2 rows in set (0.01 sec)
```

Extracting Subsets

Limit can also be used to extract a subset of data by providing an additional argument. The general form of this LIMIT is:

```
SELECT (whatever) from table LIMIT starting row, Number to extract;
SELECT f_name, l_name from
employee_data LIMIT 6,3;

+--------+-----------+
| f_name | l_name    |
+--------+-----------+
| John   | MacFarland |
| Edward | Sakamuro  |
| Alok   | Nanda     |
+--------+-----------+
3 rows in set (0.00 sec)
```

This extracts 3 rows starting from the sixth row.

ASSIGNMENTS

1. List the names of 5 youngest employees in the company.
2. Extract the next 5 entries starting with the 10 row.
3. Display the names and salary of the employee who draws the largest salary.
4. What does the following statement display?
   ```
   SELECT emp_id, age, perks
   from employee_data ORDER BY
   perks DESC LIMIT 10;
   ```

14. The DISTINCT keyword

In this section of the online MySQL guide, we will look at how to select and display records from MySQL tables using the DISTINCT keyword that eliminates the occurences of the same data.
To list all titles in our company database, we can throw a statement as:

```
select title from employee_data;
```

```
+----------------------------+
| title                      |
+----------------------------+
| CEO                        |
| Senior Programmer          |
| Senior Programmer          |
| Web Designer               |
| Web Designer               |
| Programmer                 |
| Programmer                 |
| Programmer                 |
| Programmer                 |
| Multimedia Programmer      |
| Multimedia Programmer      |
| Multimedia Programmer      |
| Senior Web Designer        |
| System Administrator       |
| System Administrator       |
| Senior Marketing Executive |
| Marketing Executive        |
| Marketing Executive        |
| Marketing Executive        |
| Customer Service Manager   |
| Finance Manager            |
+----------------------------+
21 rows in set (0.00 sec)
```

You'll notice that the display contains multiple occurences of certain data. The SQL DISTINCT clause lists only unique data. Here is how you use it.

```
select DISTINCT title from employee_data;
```

```
+----------------------------+
| title                      |
+----------------------------+
| CEO                        |
| Customer Service Manager   |
| Finance Manager            |
| Marketing Executive        |
| Multimedia Programmer      |
| Programmer                 |
| Senior Marketing Executive |
| Senior Programmer          |
| Senior Web Designer        |
| System Administrator       |
| Web Designer               |
+----------------------------+
11 rows in set (0.00 sec)
```

This shows we have 11 unique titles in the company. Also, you can sort the unique entries using ORDER BY.

```
select DISTINCT age from employee_data ORDER BY age;
```

```
+------+
| age  |
+------+
|   25 |
|   26 |
|   27 |
|   28 |
|   30 |
|   31 |
|   32 |
|   33 |
|   34 |
|   35 |
|   36 |
|   43 |
+------+
12 rows in set (0.00 sec)
```

DISTINCT is often used with the COUNT aggregate function, which we'll meet in later sessions.

ASSiGNMENTS

1. How many unique salary packages does our company, Bignet, offer? List them is descending order.
2. How many distinct first names do we have in our database?

15. Finding the minimum and maximum values

MySQL provides inbuilt functions to find the minimum and maximum values.

SQL provides 5 aggregate functions. They are:

MIN()	: Minimum value
MAX()	: Maximum value
SUM()	: The sum of values
AVG()	: The average values
COUNT()	: Counts the number of entries

In this session of the online MySQL course, we'll look at finding the minimum and maximum values in a column.

MySQL MIN() - Minimum value

```
select MIN(salary) from employee_data;

+-------------+
| MIN(salary) |
+-------------+
|       70000 |
+-------------+
1 row in set (0.00 sec)
```

MySQL MAX() - Maximum value

```
select MAX(salary) from employee_data;

+-------------+
| MAX(salary) |
+-------------+
|      200000 |
+-------------+
1 row in set (0.00 sec)
```

ASSIGNMENTS

1. List the minimum perks package.
2. List the maximum salary given to a "Programmer".
3. Display the age of the oldest "Marketing Executive".
4. (Tricky!) Find the first and last names of the oldest employee.

16. Finding the average and sum

Totalling column values with MySQL SUM
The SUM() aggregate function calculates the total of values in a column. You require to give the column name, which should be placed inside parenthesis.

Let's see how much Bignet spends on salaries.

```
select SUM(salary) from employee_data;

+-------------+
| SUM(salary) |
+-------------+
|     1997000 |
+-------------+
1 row in set (0.00 sec)
```

Similarly, we can display the total perks given to employees.

```
select SUM(perks) from employee_data;

+------------+
| SUM(perks) |
+------------+
|     390000 |
+------------+
1 row in set (0.00 sec)
```

How about finding the total of salaries and perks?

```
select sum(salary) + sum(perks) from employee_data;

+-------------------------+
| sum(salary)+ sum(perks) |
+-------------------------+
|                 2387000 |
+-------------------------+
1 row in set (0.01 sec)
```

This shows a hidden gem of the SELECT command. You can add, subtract, multiply or divide values. Actually, you can write full blown arithemetic expressions. Cool!

MySQL AVG() - Finding the Average
The AVG() aggregate function is employed for calculating averages of data in columns.

```
select avg(age) from employee_data;
+----------+
| avg(age) |
+----------+
|  31.6190 |
+----------+
1 row in set (0.00 sec)
```

This displays the average age of employees in Bignet and the following displays the average salary.

```
select avg(salary) from employee_data;

+-------------+
| avg(salary) |
+-------------+
|  95095.2381 |
+-------------+
1 row in set (0.00 sec)
```

ASSIGNMENTS
5

1. Display the sum of ages of employees working at Bignet.
2. How would you calculate the total of years of service the employees of Bignet have in the company?
3. Calculate the sum of salaries and the average age of employees who hold "Programmer" title.
4. What do you understand from the following statement?

   ```
   select (SUM(perks)/SUM(salary) * 100)
   from employee_data;
   ```

17. Naming Columns

MySQL allows you to name the displayed columns. So instead of f_name or l_name etc. you can use more descriptive terms. This is achieved with AS.

```
select avg(salary) AS
'Average Salary' from
employee_data;

+----------------+
| Average Salary |
+----------------+
|     95095.2381 |
+----------------+
1 row in set (0.00 sec)
```

Such pseudo names make will the display more clear to users. The important thing to remember here is that if you assign pseudo names that contain spaces, enclose the names in quotes. Here is another example:

```
select (SUM(perks)/SUM(salary) * 100)
AS 'Perk Percentage' from
employee_data;

+-----------------+
| Perk Percentage |
+-----------------+
|           19.53 |
+-----------------+
1 row in set (0.00 sec)
```

18. Counting

The COUNT() aggregate functions counts and displays the total number of entries. For example, to count the total number of entries in the table, issue the command below.

```
select COUNT(*) from employee_data;

+----------+
| COUNT(*) |
+----------+
|       21 |
+----------+
1 row in set (0.00 sec)
```

As you have learnt, the * sign means "all data". Now, let's count the total number of employees who hold the "Programmer" title.

```
select COUNT(*) from employee_data
where title = 'Programmer';
```

```
+----------+
| COUNT(*) |
+----------+
|        4 |
+----------+
1 row in set (0.01 sec)
```

The MySQL GROUP BY clause

The GROUP BY clause allows us to group similar data. Thus, to list all unique titles in our table we can issue

```
select title from employee_data
GROUP BY title;

+----------------------------+
| title                      |
+----------------------------+
| CEO                        |
| Customer Service Manager   |
| Finance Manager            |
| Marketing Executive        |
| Multimedia Programmer      |
| Programmer                 |
| Senior Marketing Executive |
| Senior Programmer          |
| Senior Web Designer        |
| System Administrator       |
| Web Designer               |
+----------------------------+
11 rows in set (0.01 sec)
```

You'll notice that this is similar to the usage of DISTINCT, which we encountered in a previous session. Okay, here is how you can count the number of employees with different titles.

```
select title, count(emp_id)
from employee_data GROUP BY title;

+----------------------------+----------+
| title                      | count(*) |
+----------------------------+----------+
| CEO                        |        1 |
| Customer Service Manager   |        1 |
| Finance Manager            |        1 |
| Marketing Executive        |        3 |
| Multimedia Programmer      |        3 |
| Programmer                 |        4 |
| Senior Marketing Executive |        1 |
| Senior Programmer          |        2 |
| Senior Web Designer        |        1 |
| System Administrator       |        2 |
| Web Designer               |        2 |
+----------------------------+----------+
11 rows in set (0.00 sec)
```

For the command above, MySQL first groups different titles and then executes count on each group.

Sorting the data in MySQL

Now, let's find and list the number of employees holding different titles and sort them using ORDER BY.

```
select title, count(*) AS Number
from employee_data
GROUP BY title
ORDER BY Number;

+----------------------------+--------+
| title                      | Number |
+----------------------------+--------+
| CEO                        |      1 |
| Customer Service Manager   |      1 |
| Finance Manager            |      1 |
| Senior Marketing Executive |       1 |
| Senior Web Designer        |       1 |
| Senior Programmer          |       2 |
| System Administrator       |       2 |
| Web Designer               |       2 |
| Marketing Executive        |       3 |
| Multimedia Programmer      |       3 |
| Programmer                 |       4 |
+----------------------------+---------+
11 rows in set (0.00 sec)
```

ASSIGNMENTS

1. Count the number of employees who have been with Bignet for four or more years.
2. Count employees based on their ages.
3. Modify the above so that the ages are listed in a descending order.
4. Find the average age of employees in different departments (titles).
5. Change the above statement so that the data is displayed in a descending order of average ages.
6. Find and list the percentage perk (perk/salary X 100) for each employee with the % perks sorted in a descending order.

19. HAVING clause

To list the average salary of employees in different departments (titles), we use the GROUP BY clause, as in:

```
select title, AVG(salary)
from employee_data
GROUP BY title;
```

```
+----------------------------+-------------+
| title                      | AVG(salary) |
+----------------------------+-------------+
| CEO                        | 200000.0000 |
| Customer Service Manager   |  70000.0000 |
| Finance Manager            | 120000.0000 |
| Marketing Executive        |  77333.3333 |
| Multimedia Programmer      |  83333.3333 |
| Programmer                 |  75000.0000 |
| Senior Marketing Executive | 120000.0000 |
| Senior Programmer          | 115000.0000 |
| Senior Web Designer        | 110000.0000 |
| System Administrator       |  95000.0000 |
| Web Designer               |  87500.0000 |
+----------------------------+-------------+
11 rows in set (0.00 sec)
```

Now, suppose you want to list only the departments where the average salary is more than $100000, you can't do it, even if you assign a pseudo name to AVG(salary) column. Here, the HAVING clause comes to our rescue.

```
select title, AVG(salary)
from employee_data
GROUP BY title
HAVING AVG(salary) > 100000;
```

```
+----------------------------+-------------+
| title                      | AVG(salary) |
+----------------------------+-------------+
| CEO                        | 200000.0000 |
| Finance Manager            | 120000.0000 |
| Senior Marketing Executive | 120000.0000 |
| Senior Programmer          | 115000.0000 |
| Senior Web Designer        | 110000.0000 |
+----------------------------+-------------+
5 rows in set (0.00 sec)
```

ASSIGNMENTS

1. List departments and average ages where the average age in more than 30.

20. A little more on the MySQL SELECT statement

The MySQL SELECT command is something like a print or write command of other languages. You can ask it to display text strings, numeric data, the results of mathematical expressions etc.

Displaying the MySQL version number

```
select version();

+-----------+
| version() |
+-----------+
| 3.22.32   |
+-----------+
1 row in set (0.00 sec)
```

Displaying the current date and time

```
select now();

+---------------------+
| now()               |
+---------------------+
| 2001-05-31 00:36:24 |
+---------------------+
1 row in set (0.00 sec)
```

Displaying the current Day, Month and Year

```
SELECT DAYOFMONTH(CURRENT_DATE);
+--------------------------+
| DAYOFMONTH(CURRENT_DATE) |
+--------------------------+
|                       28 |
+--------------------------+
1 row in set (0.01 sec)
```

```
SELECT MONTH(CURRENT_DATE);
+---------------------+
| MONTH(CURRENT_DATE) |
+---------------------+
|                   1 |
+---------------------+
1 row in set (0.00 sec)
```

```
SELECT YEAR(CURRENT_DATE);
+--------------------+
| YEAR(CURRENT_DATE) |
+--------------------+
|               2001 |
+--------------------+
1 row in set (0.00 sec)
```

Displaying text strings

```
select 'I Love MySQL';
```

```
+--------------+
| I Love MySQL |
+--------------+
| I Love MySQL |
+--------------+
1 row in set (0.00 sec)
```

Obviously you can provide pseudo names for these columns using AS.

```
select 'Manish Sharma' as Name;
```

```
+---------------+
| Name          |
+---------------+
| Manish Sharma |
+---------------+
1 row in set (0.00 sec)
```

Evaluating expressions in MySQL

```
select ((4 * 4) / 10 ) + 25;
```

```
+----------------------+
| ((4 * 4) / 10 ) + 25 |
+----------------------+
|                26.60 |
+----------------------+
1 row in set (0.00 sec)
```

Concatenating in MySQL

With SELECT you can concatenate values for display. CONCAT accepts arguments between parenthesis. These can be column names or plain text strings. Text strings have to be surrounded with quotes (single or double).

```
SELECT CONCAT(f_name, " ", l_name)
```

```
from employee_data
where title = 'Programmer';

+----------------------------+
| CONCAT(f_name, " ", l_name) |
+----------------------------+
| Fred Kruger                |
| John MacFarland            |
| Edward Sakamuro            |
| Alok Nanda                 |
+----------------------------+
4 rows in set (0.00 sec)
```

You can also give descriptive names to these columns using AS.

```
select CONCAT(f_name, " ", l_name)
AS Name
from employee_data
swhere title = 'Marketing Executive';

+---------------+
| Name          |
+---------------+
| Monica Sehgal |
| Hal Simlai    |
| Joseph Irvine |
+---------------+
3 rows in set (0.00 sec)
```

ASSIGNMENTS

1. Which command displays the MySQL version?
2. Use the SELECT command to evaluate 4 X 4 X 4 and name the column Cube of 4.
3. Display your name with SELECT.

21. MySQL mathematical Functions

In addition to the four basic arithmetic operations addition (+), Subtraction (-), Multiplication (*) and Division (/), MySQL also has the Modulo (%) operator. This calculates the remainder left after division.

```
select 87 % 9;
+--------+
| 87 % 9 |
+--------+
|      6 |
+--------+
1 row in set (0.00 sec)
```

MySQL - MOD(x, y)
Displays the remainder of x divided by y, SImilar to the Modulus operator.

```
select MOD(37, 13);
```

```
+-------------+
| MOD(37, 13) |
+-------------+
|          11 |
+-------------+
1 row in set (0.00 sec)
```

MySQL ABS(x)
Calculates the Absolute value of number x. Thus, if x is negative its positive value is returned.

```
select ABS(-4.05022);
+---------------+
| ABS(-4.05022) |
+---------------+
|       4.05022 |
+---------------+
1 row in set (0.00 sec)
```

```
select ABS(4.05022);
+--------------+
| ABS(4.05022) |
+--------------+
|      4.05022 |
+--------------+
1 row in set (0.00 sec)
```

SQL SIGN(x)
Returns 1, 0 or -1 when x is positive, zero or negative, respectively.

```
select SIGN(-34.22);
+--------------+
| SIGN(-34.22) |
+--------------+
|           -1 |
+--------------+
1 row in set (0.00 sec)
```

```
select SIGN(54.6);
+------------+
| SIGN(54.6) |
+------------+
|          1 |
+------------+
1 row in set (0.00 sec)
```

```
select SIGN(0);
+---------+
| SIGN(0) |
+---------+
|       0 |
+---------+
1 row in set (0.00 sec)
```

POWER(x,y)
Calculates the value of x raised to the power of y.

```
select POWER(4,3);
+------------+
| POWER(4,3) |
+------------+
|  64.000000 |
+------------+
1 row in set (0.00 sec)
```

SQRT(x)
Calculates the square root of x.

```
select SQRT(3);
+----------+
| SQRT(3)  |
+----------+
| 1.732051 |
+----------+
1 row in set (0.00 sec)
```

ROUND(x) and ROUND(x,y)
Returns the value of x rounded to the nearest integer. ROUND can also accept an additional argument y that will round x to y decimal places.

```
select ROUND(14.492);
+---------------+
| ROUND(14.492) |
+---------------+
|            14 |
+---------------+
1 row in set (0.00 sec)

select ROUND(4.5002);
+---------------+
| ROUND(4.5002) |
+---------------+
|             5 |
+---------------+
1 row in set (0.00 sec)

select ROUND(-12.773);
+----------------+
| ROUND(-12.773) |
+----------------+
|            -13 |
+----------------+
1 row in set (0.00 sec)

select ROUND(7.235651, 3);
+--------------------+
| ROUND(7.235651, 3) |
+--------------------+
|              7.236 |
+--------------------+
1 row in set (0.00 sec)
```

FLOOR(x)
Returns the largest integer that is less than or equal to x.

```
select FLOOR(23.544);
+---------------+
| FLOOR(23.544) |
+---------------+
|            23 |
+---------------+
1 row in set (0.00 sec)
```

```
select FLOOR(-18.4);
+--------------+
| FLOOR(-18.4) |
+--------------+
|          -19 |
+--------------+
1 row in set (0.00 sec)
```

CEILING(x)
Returns the smallest integer that is greater than or equal to x.

```
select CEILING(54.22);
+----------------+
| CEILING(54.22) |
+----------------+
|             55 |
+----------------+
1 row in set (0.00 sec)
```

```
select CEILING(-62.23);
+-----------------+
| CEILING(-62.23) |
+-----------------+
|             -62 |
+-----------------+
1 row in set (0.00 sec)
```

TAN(x), SIN(x) and COS(x)
Calculate the trignometic ratios for angle x (measured in radians).

```
select SIN(0);
+----------+
| SIN(0)   |
+----------+
| 0.000000 |
+----------+
1 row in set (0.00 sec)
```

22. Updating records

The SQL UPDATE command updates the data in tables. Its format is quite simple.

```
UPDATE table_name SET
```

```
column_name1 = value1,
column_name2 = value2,
column_name3 = value3 ...
[WHERE conditions];
```

Obviously, like other SQL commands, you can type in in one line or multiple lines.

Let's look at some examples.
Bignet has been doing good business, the CEO increases his salary by $20000 and perks by $5000. His previous salary was $200000 and perks were $50000.

```
UPDATE employee_data SET
salary=220000, perks=55000
WHERE title='CEO';

Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

You can test this out by listing the data in the table.

```
select salary, perks from
employee_data WHERE
title = 'CEO';

+--------+-------+
| salary | perks |
+--------+-------+
| 220000 | 55000 |
+--------+-------+
1 row in set (0.00 sec)
```

Actually, you don't need to know the previous salary explicitly. You can be cheeky and use arithmetic operators. Thus, the following statement would have done the same job without us knowing the original data beforehand.

```
UPDATE employee_data SET
salary = salary + 20000,
perks = perks + 5000
WHERE title='CEO';

Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Another progressive (???) step Bignet takes is changing the titles of Web Designer to Web Developer.

```
mysql> update employee_data SET
    -> title = 'Web Developer'
    -> WHERE title = 'Web Designer';
Query OK, 2 rows affected (0.00 sec)
Rows matched: 2  Changed: 2  Warnings: 0
```

It's important that you take a long hard look at the condition part in the statement before executing update, else you might update the wrong data. Also, an UPDATE statement without conditions will update all the data in the column in all rows!
Be very caferul.

ASSiGNMENTS

1. Our CEO falls in love with the petite Web Developer, Anamika Pandit. She now wants her last name to be changed to 'Sharma'.
2. All Multimedia Programmers now want to be called Multimedia Specialists.
3. After his marriage, the CEO gives everyone a raise. Increase the salaries of all employees (except the CEO) by $10000.

23. MySQL Date column type part 1

Till now we've dealt with text (varchar) and numbers (int) data types. To understand date type, we'll create one more table.
Download employee_per.dat file below and follow the instructions. The file contain the CREATE table command as well as the INSERT statements.

```
employee_per.dat
```

These instructions are a repeat of those in session 6 and 8.
1. Move the file to c:\xampp\mysql\bin.
2. Issue the command at DOS prompt.

   ```
   c:\xampp\mysql\bin> mysql employees <employee_per.dat
   ```

3. Start mysql client program and check if the table has been created using SHOW TABLES; command.

The details of the table can be displayed with DESCRIBE command.

```
mysql> DESCRIBE employee_per;

+------------+------------------+------+-----+---------+-------+
| Field      | Type             | Null | Key | Default | Extra |
+------------+------------------+------+-----+---------+-------+
| e_id       | int(10) unsigned |      | PRI | 0       |       |
| address    | varchar(60)      | YES  |     | NULL    |       |
| phone      | int(11)          | YES  |     | NULL    |       |
| p_email    | varchar(60)      | YES  |     | NULL    |       |
| birth_date | date             | YES  |     | NULL    |       |
| sex        | enum('M','F')    | YES  |     | NULL    |       |
| m_status   | enum('Y','N')    | YES  |     | NULL    |       |
| s_name     | varchar(40)      | YES  |     | NULL    |       |
| children   | int(11)          | YES  |     | NULL    |       |
+------------+------------------+------+-----+---------+-------+
9 rows in set (0.01 sec)
```

Notice that column birth_date has date as column type. I've also introduced another column type ENUM, which we'll discuss later.

e-id	: are employee ids, same as that in table employee_data
address	: Addresses of employees
phone	: Phone numbers

p_email	: Personal email addresses
birth_date	: Birth dates
sex	: The sex of the employee, Male or Female
m_status	: Marital status,Yes or No.
s_name	: Name of Spouse (NULL if employee is unmarried)
children	: Number of children (NULL if employee is unmarried)

MySQL dates are ALWAYS represented with the year followed by the month and then the date. Often you'll find dates written as YYYY-MM-DD, where YYYY is 4 digit year, MM is 2 digit month and DD, 2 digit date. We'll look at DATE and related column types in the session on column types.

Operations on Date
Date column type allow for several operations such as sorting, testing conditions using comparision operators etc.

Using = and != operators on dates

```
select p_email, phone
from employee_per
where birth_date = '1969-12-31';

+---------------------------+---------+
| p_email                   | phone   |
+---------------------------+---------+
| killeratlarge@elmmail.com | 6666666 |
+---------------------------+---------+
1 row in set (0.00 sec)
```

Note: MySQL requires the dates to be enclosed in quotes.
Using >= and <= operators

```
select e_id, birth_date
from employee_per where
birth_date >= '1970-01-01';

+------+------------+
| e_id | birth_date |
+------+------------+
|    1 | 1972-03-16 |
|    4 | 1972-08-09 |
|    5 | 1974-10-13 |
|    8 | 1975-01-12 |
|   17 | 1970-04-18 |
|   18 | 1973-10-09 |
|   19 | 1973-01-20 |
+------+------------+
7 rows in set (0.00 sec)
```

Specifying date ranges in MySQL

```
select e_id, birth_date
from employee_per where
birth_date BETWEEN
'1969-01-01' AND '1974-01-01';
```

```
+------+------------+
| e_id | birth_date |
+------+------------+
|    1 | 1972-03-16 |
|    4 | 1972-08-09 |
|    6 | 1969-12-31 |
|   17 | 1970-04-18 |
|   18 | 1973-10-09 |
|   19 | 1973-01-20 |
+------+------------+
6 rows in set (0.00 sec)
```

```
select e_id, birth_date
from employee_per where
birth_date >= '1969-01-01'
AND birth_date <= '1974-01-01';
```

```
+------+------------+
| e_id | birth_date |
+------+------------+
|    1 | 1972-03-16 |
|    4 | 1972-08-09 |
|    6 | 1969-12-31 |
|   17 | 1970-04-18 |
|   18 | 1973-10-09 |
|   19 | 1973-01-20 |
+------+------------+
6 rows in set (0.00 sec)
```

ASSIGNMENTS

1. List all employee ids and birth dates who were born before 1965.
2. Display Ids and birth dates of employees born in and between 1970 and 1972. (This is tricky... you need to give it a little thought)

24. MySQL Date column type part 2

Using Date to sort data

```
select e_id, birth_date
from employee_per
ORDER BY birth_date;
```

```
+------+------------+
| e_id | birth_date |
+------+------------+
|   11 | 1957-11-04 |
|   16 | 1964-03-06 |
|   21 | 1964-06-13 |
|   14 | 1965-04-28 |
|   15 | 1966-06-23 |
|    7 | 1966-08-20 |
|   10 | 1967-07-06 |
|   20 | 1968-01-25 |
|   12 | 1968-02-15 |
|    2 | 1968-04-02 |
|    9 | 1968-05-19 |
|   13 | 1968-09-03 |
|    3 | 1968-09-22 |
|    6 | 1969-12-31 |
|   17 | 1970-04-18 |
|    1 | 1972-03-16 |
|    4 | 1972-08-09 |
|   19 | 1973-01-20 |
|   18 | 1973-10-09 |
|    5 | 1974-10-13 |
|    8 | 1975-01-12 |
+------+------------+
```

Selecting data using Dates

Here is how we can select employees born in March.

```
select e_id, birth_date
from employee_per
where MONTH(birth_date) = 3;

+------+------------+
| e_id | birth_date |
+------+------------+
|    1 | 1972-03-16 |
|   16 | 1964-03-06 |
+------+------------+
2 rows in set (0.00 sec)
```

Alternatively, we can use month names instead of numbers.

```
select e_id, birth_date
from employee_per
where MONTHNAME(birth_date) = 'January';

+------+------------+
| e_id | birth_date |
+------+------------+
|    8 | 1975-01-12 |
|   19 | 1973-01-20 |
|   20 | 1968-01-25 |
+------+------------+
3 rows in set (0.00 sec)
```

Be careful when using month names as they are case sensitive. Thus, January will work but JANUARY will not! Similarly, you can select employees born in a specific year or under specific dates.

```
select e_id, birth_date
from employee_per
where year(birth_date) = 1972;

+------+------------+
| e_id | birth_date |
+------+------------+
|    1 | 1972-03-16 |
|    4 | 1972-08-09 |
+------+------------+
2 rows in set (0.00 sec)
```

```
select e_id, birth_date
from employee_per
where DAYOFMONTH(birth_date) = 20;

+------+------------+
| e_id | birth_date |
+------+------------+
|    7 | 1966-08-20 |
|   19 | 1973-01-20 |
+------+------------+
2 rows in set (0.00 sec)
```

Current dates

We had seen in the session on SELECT statement (A little more on the SELECT statement) that current date, month and year can be displayed with CURRENT_DATE argument to DAYOFMONTH(), MONTH() and YEAR() clauses, respectively. The same can be used to select data from tables.

```
select e_id, birth_date
from employee_per where
MONTH(birth_date) = MONTH(CURRENT_DATE);

+------+------------+
| e_id | birth_date |
+------+------------+
|    8 | 1975-01-12 |
|   19 | 1973-01-20 |
|   20 | 1968-01-25 |
+------+------------+
3 rows in set (0.00 sec)
```

ASSIGNMENTS

1. List ids, birth dates and emails of employes born in April.
2. Display Ids, birth dates and spouse names of employees born in 1969 and sort the entires on the basis of their spouse names.
3. List the emploee ids for employees born under the current month.
4. How many unique birth years do we have?
5. Display a list of unique birth years and the number of employees born under each.
6. How many employees were born under each month? The display should have month names (NOT numbers) and the entries should be sorted with the month having the largest number listed first.

25. Null column type

The NULL column type is special in many ways. To insert a NULL value, just leave the column name from the INSERT statement. Columns have NULL as default unless specified by NOT NULL. You can have null values for integers as well as text or binary data.

NULL cannot be compared using arithemetic operators. Comparisions for NULL take place with IS NULL or IS NOT NULL.

```
select e_id, children
from employee_per
where children IS NOT NULL;

+------+----------+
| e_id | children |
+------+----------+
|    2 |        3 |
|    3 |        2 |
|    7 |        3 |
|    9 |        1 |
|   11 |        4 |
|   12 |        3 |
|   13 |        2 |
|   15 |        3 |
|   16 |        2 |
|   17 |        1 |
|   21 |        2 |
+------+----------+
11 rows in set (0.00 sec)
```

The above lists ids and no. of children of all employees who have children.

ASSIGNMENTS

1. Find and list the ids and spouse names of all employees who are married.
2. Change the above so that the display is sorted on spouse names.
3. How many employees do we have under each sex (male and female)?
4. How many of our employees are married and unmarried?
5. Find the total number of children.
6. Make unique groups of children and find the number in each group. Sort the display with the group having maximum children, at the top.

26. MySQL table joins

Till now, we've used SELECT to retrieve data from only one table. However, we can extract data from two or more tables using a single SELECT statement.

The strength of RDBMS lies in allowing us to relate data from one table with data from another. This correlation can only be made if atleast one column in the two tables contain related data.

In our example, the columns that contain related data are emp_id of employee_data and e_id of employee_per.

Let's conduct a table join and extract the names (from employee_data) and spouse names (from employee_per) of married employee.

```
select CONCAT(f_name, " ", l_name) AS Name,
s_name as 'Spouse Name' from
```

```
employee_data, employee_per
where m_status = 'Y' AND
emp_id = e_id;

+-----------------+-----------------+
| Name            | Spouse Name     |
+-----------------+-----------------+
| Manish Sharma   | Anamika Sharma  |
| John Hagan      | Jane Donner     |
| Ganesh Pillai   | Sandhya Pillai  |
| Anamika Sharma  | Manish Sharma   |
| John MacFarland | Mary Shelly     |
| Alok Nanda      | Manika Nanda    |
| Paul Simon      | Muriel Lovelace |
| Arthur Hoopla   | Rina Brighton   |
| Kim Hunter      | Matt Shikari    |
| Danny Gibson    | Betty Cudly     |
| Mike Harper     | Stella Stevens  |
| Monica Sehgal   | Edgar Alan      |
| Peter Champion  | Ruby Richer     |
+-----------------+-----------------+
13 rows in set (0.00 sec)
```

The FROM clause takes the names of the two tables from which we plan to extract data. Also, we specify that data has to be retrieved for only those entries where the emp_id and e_id are same.

The names of columns in the two tables are unique. However, this may not true always, in which case we can explicitly specify column names along with table name using the dot notation.

```
select CONCAT(employee_data.f_name, " ", employee_data.l_name)
AS Name, employee_per.s_name AS 'Spouse Name'
from employee_data, employee_per
where employee_per.m_status = 'Y'
AND employee_data.emp_id = employee_per.e_id;

+-----------------+-----------------+
| Name            | Spouse Name     |
+-----------------+-----------------+
| Manish Sharma   | Anamika Sharma  |
| John Hagan      | Jane Donner     |
| Ganesh Pillai   | Sandhya Pillai  |
| Anamika Sharma  | Manish Sharma   |
| John MacFarland | Mary Shelly     |
| Alok Nanda      | Manika Nanda    |
| Paul Simon      | Muriel Lovelace |
| Arthur Hoopla   | Rina Brighton   |
| Kim Hunter      | Matt Shikari    |
| Danny Gibson    | Betty Cudly     |
| Mike Harper     | Stella Stevens  |
| Monica Sehgal   | Edgar Alan      |
| Peter Champion  | Ruby Richer     |
+-----------------+-----------------+
13 rows in set (0.00 sec)
```

27. Deleting entries from tables

The SQL delete statement requires the table name and optional conditions.

```
DELETE from table_name [WHERE conditions];
```

NOTE: If you don't specify any conditions ALL THE DATA IN THE TABLE WILL BE DELETED!!!

One of the Multimedia specialists 'Hasan Rajabi' (employee id 10) leaves the company. We'll delete his entry.

```
DELETE from employee_data
WHERE emp_id = 10;

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

28. Dropping tables

To remove all entries from the table we can issue the DELETE statement without any conditions.

```
DELETE from employee_data;
Query OK, 0 rows affected (0.00 sec)
```

However, this does not delete the table. The table still remains, which you can check with SHOW TABLES;

```
mysql> SHOW TABLES;
+---------------------+
| Tables in employees |
+---------------------+
| employee_data       |
+---------------------+
1 rows in set (0.00 sec)
```

To delete the table, we issue a DROP table command.

```
DROP TABLE employee_data;
Query OK, 0 rows affected (0.01 sec)
```

Now, we won't get this table in SHOW TABLES; listing

29. MySQL database Column Types

The three major types of column types used in MySQL are

a. Integer
b. Text

c. Date

Choosing a column data type is very important in order to achieve speed, effective storage and retrieval. Hence, I've dedicated two sessions to this topic. Now, I'll be touching only the surface; for a thorough explanation refer the resources in What Next? session.

MySQL Numeric Column Types

In addition to int (Integer data type), MySQL also has provision for floating-point and double precision numbers. Each integer type can take also be UNSIGNED and/or AUTO_INCREMENT.

TINYINT	:	very small numbers; suitable for ages. Actually, we should have used this data type for employee ages and number of children. Can store numbers between 0 to 255 if UNSIGNED clause is applied, else the range is between -128 to 127.
SMALLINT	:	Suitable for numbers between 0 to 65535 (UNSIGNED) or -32768 to 32767.
MEDIUMINT	:	0 to 16777215 with UNSIGNED clause or -8388608 to 8388607.
INT	:	UNSIGNED integers fall between 0 to 4294967295 or -2147683648 to 2147683647.
BIGINT	:	Huge numbers. (-9223372036854775808 to 9223372036854775807)
FLOAT	:	Floating point numbers (single precision)
DOUBLE	:	Floating point numbers (double precision)
DECIMAL	:	Floating point numbers represented as strings.

Date and time column types

DATE	:	YYYY-MM-DD (Four digit year followed by two digit month and date) range from 1000-01-01 to 9999-12-31.
TIME	:	hh:mm:ss (Hours:Minutes:Seconds)
DATETIME	:	YYYY-MM-DD hh:mm:ss (Date and time separated by a space character)
TIMESTAMP	:	YYYYMMDDhhmmss range 1970-01-01 00:00:00 to 2037-12-31 23:59:59
YEAR	:	YYYY (4 digit year) range 1970 to 2069

30. Column Types part 2

MySQL Text data type

Text can be fixed length (char) or variable length strings. Also, text comparisions can be case sensitive or insensitive depending on the type you choose.

CHAR(x)	:	where x can range from 1 to 255.
VARCHAR(x)	:	x ranges from 1 - 255
TINYTEXT	:	small text, case insensitive

TEXT :	slightly longer text, case insensitive 65535 character
MEDIUMTEXT:	medium size text, case insensitive 16777215 character
LONGTEXT :	really long text, case insensitive 4294967295 character
TINYBLOB :	Blob means a Binary Large OBject. You should use blobs for case sensitive searches.
BLOB :	slightly larger blob, case sensitive.
MEDIUMBLOB:	medium sized blobs, case sensitive.
LONGBLOB :	really huge blobs, case sensitive.
ENUM :	Enumeration data type have fixed values and the column can take only one value from the given set. The values are placed in parenthesis following ENUM declaration. An example, is the marital status column we encountered in employee_per table. m_status ENUM("Y", "N") Thus, m_status column will take only Y or N as values. If you specify any other value with the INSERT statement, MYSQL will not return an error, it just inserts a NULLvalue in the column.
SET :	An extension of ENUM. Values are fixed and placed after the SET declaration; however, SET columns can take multiple values from the values provided. Consider a column with the SET data type as hobbies SET ("Reading", "Surfing", "Trekking", "Computing")

You can have 0 or all the four values in the column.

```
INSERT tablename (hobbies) values ("Surfing", "Computing");
```

31. MySQL guide - Last Words

Okay! This completes the tutorial on MySQL.