School of Information and Computer Technology
Sirindhorn International Institute of Technology
Thammasat University
ITS351 Database Programming Laboratory

Laboratory #5: MySQL

Objective:
- To learn how to connect to MySQL
- To learn how to maintain MySQL, including table creation and MySQL meta commands
- To learn how to make a query in MySQL
- To learn how to update data in MySQL

MySQL Resource: http://www.mysql.com/

1 What is MySQL

MySQL is currently the most popular open source database server in existence. On top of that, it is very commonly used in conjunction with PHP scripts to create powerful and dynamic server-side applications. MySQL is a relational database management system that runs as a server providing multi-user access to a number of databases.

2 Installation of MySQL Database Management Systems

MySQL is available for Windows, Linux and other UNIX variants. You can get MySQL from its web site www.mysql.com.

MySQL download

MySQL download - the WINDOWS version

On the download page of MySQL web site, you will 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 recommended 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 Installation

MySQL installation on Windows

Once you have successfully downloaded the Windows version, installing it is a breeze. Note that if you use Appserv, it will automatically install MySQL and start the service. You can ignore these steps.

1. Create a temporary directory called mysqltem.
2. Unzip the file to this directory.
3. After unzipping is over, you'll find a file called "setup.exe".

4. Close all programs
5. Click on Start - Run and browse to the setup file in mysqltem.
6. Click "OK" to proceed
7. The setup program loads and guides you through the installation process.
8. Choose the "Typical" installation, unless you know what you want!
9. MySQL would be installed in c:\mysql (unless you specified some other directory).
10. Restart Windows.
11. Start an MS-DOS session and migrate to c:\mysql\bin
12. Now, type the following at the prompt:

```
mysqld-shareware –standalone
```

OR (in later versions)

```
mysqld
```

13. Type "mysql" (without the quotes) at the DOS prompt.
14. The prompt is changed to the "mysql" prompt.
15. To test the MySQL server, type "show databases;" as follows.

```
mysql> show databases;

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

17. Type "quit" at the mysql prompt or "\q" to exit mySQL.
18. Since our work is done (for the time being), we should shut down the MySQL server. Issue the following command at the prompt.

```
mysqladmin -u root shutdown
```

<u>MySQL installation on Ubuntu</u>

It is simple to install MySQL on Ubuntu. Note that you need to have a root privilege (administrator's right) to do so.

1. Type the following command at the prompt:

```
sudo apt-get install mysql-server
```

2. The MySQL will start up automatically after the installation. Anyway, we can start up a MySQL manually using the following command.

```
$ sudo /etc/init.d/mysql start
```

```
* Starting MySQL database server mysqld                    [ OK ]
```

3. We can check whether the MySQL daemon start up properly or not by the following command.

```
$ sudo /etc/init.d/mysql status
```

4. We can change the root password of MySQL as follows. Note that the root of MySQL is **not** the root of UNIX (or Linux).

```
$ mysqladmin -uroot password rootpass
```

By this command the root password will be changed to 'rootpass'. We also can change the root password by entering the mysql first as shown below.

```
$ mysql –u root
use mysql
select host, user, password from user;
update user set password = password('rootpass')
where user = 'root';
quit
% mysqladmin –u root reload
% mysql –u root –p
                  🡐 enter password
select host, user, password from user;
```

5. We can create a MySQL database by the following command (use root).

```
$ mysqladmin -uroot -prootpass create testdb
```

We can show the list of databases by the following command.

```
$ mysql -uroot -prootpass
show databases
```

6. We can create a new user in the MySQL database by the following command (use root). Here, the user account is 'testuser', the password is 'testuser' and the database is 'testdb'.

```
$ mysql -uroot -prootpass
use testdb
insert into user (host, user, password) values
('localhost', 'testuser', password('testpass'));
select host, user, password from user;
quit
$ mysqladmin -u root -p reload
$ mysql -utestuser -ptestpass
```

7. We can grant permission to the user 'testuser' to select the database 'testdb'.

```
$ mysql -uroot -prootpass
grant select on testdb.* to testuser;
select host, user, password from user;
quit
```

```
$ mysqladmin -u root -p reload
$ mysql -utestuser -ptestpass
```

8. Once MySQL client is running, you should get the mysql> prompt. Type the following at this prompt and get the following result.

```
mysql > show databases;
+----------------+
| Database       |
+----------------+
| mysql          |
| test           |
+----------------+
2 rows in set (0.00 sec)
```

Okay, we've successfully installed MySQL on your system.

3 MySQL Usage

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.

Why we need 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.

Create a database (for root (admin) user of MySQL only)

In this section, we will learn how to create a database. **(Note that the commands in this section can be run only by the root (admin) user of MySQL.)** The commands for creating a database in Windows and Linux are the same. However, the preliminary commands in Linux are slightly more complex. We will discuss the Windows and Linux systems separately. We will 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.

Windows system

1. Start the MySQL server by issuing the command **mysqld-shareware --standalone** at the prompt in c:\mysql\bin. Refer the previous session Installing MySQL on Windows for further details. Note that if MySQL is installed through AppServ, then the MySQL server usually starts automatically on boot.
2. Now invoke the mysql client program by typing **mysql** at the prompt.
3. The prompt is changed to a **mysql>** prompt. Type the following command and get the following result.

```
mysql> create database employees;
Query OK, 1 row affected (0.00 sec)
```

(Note: The command ends with a semi-colon).

4. This means that you have successfully created the database. Now, let's see how many databases you have on your system. Issue the following command and then get the following result.

```
mysql> show 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.

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

Linux system

1. We assume that you are working from your account and not the root. Start a terminal session and become the superuser (Type su at the prompt and then enter the root password).
2. Now we'll access the MySQL server. Type:

```
mysql -u root -p
```

The system prompts for the MySQL root password that you set up in Installing MySQL on Linux. (Note: This is not the Linux root password but the MySQL root password). Enter the password, which is not displayed for security reasons. Once you are successfully logged in, the system prints a welcome message and displays the **mysql** prompt ... something like

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 3.22.32

Type 'help' for help.
mysql>
```

3. Now we are ready for creating the employees database. Issue the command and get the following result.

```
mysql> create database employees;
Query OK, 1 row affected (0.00 sec)
```

4. An important point to note is that this database is created by the root and so will not be accessible to any other user unless permitted by the root. Thus, in order to use this database from my account, I have to set the permissions by issuing the following command:

```
mysql> GRANT ALL ON employees.* TO username@localhost
        IDENTIFIED BY "123456";
```

The above command *grants* my account (*username@localhost*) all the permissions *on employees* database and sets my password to ***123456***. You should replace *username* with your user name and choose an appropriate password.

5. Then close the mysql session by typing `quit` at the prompt. Exit from superuser and come back to your account. (Type exit).

(Note: From now on, any mysql users can use the commands on server)

6. To connect to MySQL from your account, type:

```
$ mysql  -u [user_name] –p
```

Specifying the database name at the start by using the command to access mysql at the system prompt:

```
$ mysql [database_name] -u [user_name]  -p
```

You need to type in the password when prompted. (This password was set by the GRANTS ALL... command above). The system displays the welcome message once you have successfully logged on to MySQL. Here is how your session should look like:

```
[test@localhost test]$ mysql -u u48n0001 -p

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 273 to server version: 5.0.10-beta-Max

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

6. Typing the command *`SHOW DATABASES;`* will list all the databases available on the system. You should get the following result.

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| test               |
| u49n0001db         |
+--------------------+

3 rows in set (1.20 sec)
```

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

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

Using a database

Let's start the mysql client program and select our database. At the mysql prompt, issue the command:

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

Typing the command *SELECT DATABASE();* will list the database that we selected to use. The above shows that no database has been selected. Actually, every time we work with mysql client, we have to specify which database we plan to use. There are several ways of doing it. It's necessary to specify the database we plan to use; else MySQL will throw an error.

1) Specifying the database with the USE statement at the mysql prompt is done as follows. In this example, u49n0001db is the name of the database.

```
mysql> use u49n0001db;
```

2) Specifying the database with \u at the mysql prompt:

```
mysql> \u u49n0001db;
```

Create tables

In this section, we will explore the MySQL commands to create database tables. Databases store data in 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; they are 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 later.

```
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 **var**riable **char**acters. 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 long. Let's break it down.

Type	Description
int	specify that the column type is an integer (a number).
unsigned	determine 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 *Bignet* 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!

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)
mysql>
```

Okay, we just made our first table.

Using 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)
```

Describing tables

MySQL provides us with a command that displays the column details of the tables.
Issue the following command at the mysql prompt:

```
Mysql> DESC 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 or **DESC** 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.

Insert data into tables

The **INSERT** SQL statement impregnates our table with data as follows.

```
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. The following statement inserts the first record in **employee_data** table.

```
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");
```

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 easier, INSERT statements can be packed into a file (one statement on one line). In this example, the file is called **employee.dat.** Note that the file extension is not necessarily ".dat". It could be ".txt" or anything as needed.

employee.dat

```
INSERT INTO employee_data (f_name, l_name, title, age, yos, salary, perks, email)
VALUES ("John", "Hagan", "Senior Programmer", 32, 4, 120000, 25000,
"john_hagan@bignet.com");

INSERT INTO employee_data (f_name, l_name, title, age, yos, salary, perks, email)
VALUES ("Ganesh", "Pillai", "Senior Programmer", 32, 4, 110000, 20000,
"g_pillai@bignet.com");

INSERT INTO employee_data (f_name, l_name, title, age, yos, salary, perks, email)
VALUES ("Anamika", "Pandit", "Web Designer", 27, 3, 90000, 15000, "ana@bignet.com");

INSERT INTO employee_data (f_name, l_name, title, age, yos, salary, perks, email)
VALUES ("Mary", "Anchor", "Web Designer", 26, 2, 85000, 15000, "mary@bignet.com");

INSERT INTO employee_data (f_name, l_name, title, age, yos, salary, perks, email)
VALUES ("Fred", "Kruger", "Programmer", 31, 3, 75000, 15000, "fk@bignet.com");

INSERT INTO employee_data (f_name, l_name, title, age, yos, salary, perks, email)
```

```
VALUES ("John", "MacFarland", "Programmer", 34, 4, 80000, 16000, "john@bignet.com");

INSERT INTO employee_data (f_name, l_name, title, age, yos, salary, perks, email)
VALUES ("Edward", "Sakamuro", "Programmer", 25, 2, 75000, 14000,
"eddie@bignet.com");

INSERT INTO employee_data (f_name, l_name, title, age, yos, salary, perks, email)
VALUES ("Alok", "Nanda", "Programmer", 32, 3, 70000, 10000, "alok@bignet.com");

INSERT INTO employee_data (f_name, l_name, title, age, yos, salary, perks, email)
VALUES ("Hassan", "Rajabi", "Multimedia Programmer", 33, 3, 90000, 15000,
"hasan@bignet.com");

INSERT INTO employee_data (f_name, l_name, title, age, yos, salary, perks, email)
VALUES ("Paul", "Simon", "Multimedia Programmer", 43, 2, 85000, 12000,
"ps@bignet.com");

INSERT INTO employee_data (f_name, l_name, title, age, yos, salary, perks, email)
VALUES ("Arthur", "Hoopla", "Multimedia Programmer", 32, 1, 75000, 15000,
"arthur@bignet.com");

INSERT INTO employee_data (f_name, l_name, title, age, yos, salary, perks, email)
VALUES ("Kim", "Hunter", "Senior Web Designer", 32, 2, 110000, 20000,
"kim@bignet.com");

INSERT INTO employee_data (f_name, l_name, title, age, yos, salary, perks, email)
VALUES ("Roger", "Lewis", "System Administrator", 35, 2, 100000, 13000,
"roger@bignet.com");

INSERT INTO employee_data (f_name, l_name, title, age, yos, salary, perks, email)
VALUES ("Danny", "Gibson", "System Administrator", 34, 1, 90000, 12000,
"danny@bignet.com");

INSERT INTO employee_data (f_name, l_name, title, age, yos, salary, perks, email)
VALUES ("Mike", "Harper", "Senior Marketing Executive", 36, 2, 120000, 28000,
"mike@bignet.com");

INSERT INTO employee_data (f_name, l_name, title, age, yos, salary, perks, email)
VALUES ("Monica", "Sehgal", "Marketing Executive", 30, 3, 90000, 25000,
"monica@bignet.com");

INSERT INTO employee_data (f_name, l_name, title, age, yos, salary, perks, email)
VALUES ("Hal", "Simlai", "Marketing Executive", 27, 2, 70000, 18000,
"hal@bignet.com");

INSERT INTO employee_data (f_name, l_name, title, age, yos, salary, perks, email)
VALUES ("Joseph", "Irvine", "Marketing Executive", 27, 2, 72000, 18000,
"joseph@bignet.com");

INSERT INTO employee_data (f_name, l_name, title, age, yos, salary, perks, email)
VALUES ("Shahida", "Ali", "Customer Service Manager", 32, 3, 70000, 9000,
"shahida@bignet.com");

INSERT INTO employee_data (f_name, l_name, title, age, yos, salary, perks, email)
VALUES ("Peter", "Champion", "Finance Manager", 36, 4, 120000, 25000,
"peter@bignet.com");
```

<u>Inserting data into employee_data table with employee.dat file</u>

On Linux (From shell)

1. Exit from mysql by using “\q” to command prompt.
2. Go to the directory of the downloaded file. Issue the following command

```
mysql u49nxxxxdb < employee.dat –u u49nxxxx -p
```

3. Enter your password.

For Windows, one may use the command “source” in MySQL prompt.

On MySQL Prompt

Issue the following command to execute MySQL statements line by line

```
mysql> source employee.dat ;
```

Our table contains 21 entries (20 from employee.dat file and one from the INSERT statement we issued at the beginning).

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]

[Group By ...column_name]

[Order By ... column_name][asc/desc]

[Having ... conditions];
```

The conditions part of the statement is optional (we'll go through this later). Basically, you need 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.

```
mysql> SELECT f_name, l_name FROM employee_data;
+---------+------------+
| f_name  | l_name     |
+---------+------------+
| Manish  | Sharma     |
| John    | Hagan      |
...   ... ... ... ... ...
...   ... ... ... ... ...
| Shahida | Ali        |
| Peter   | Champion   |
+---------+------------+
21 rows in set (0.00 sec)
```

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

```
mysql> 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.

Assignments 1

1. **Write the complete SQL statement for creating a new database called employee**
2. **How would you list all the databases available on the system?**
3. **Which statement is used to list the information about a table? How do you use this statement?**
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. **Write the statement for listing data from salary, perks and yos (year of service) columns of employee_data table.**

Selecting data using 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 section we will look at the SQL Comparison Operators.

- **The = and != comparision operators**

```
mysql> 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 the first and last names of all employees who are programmers.

```
mysql> 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)
```

List the first and last names of all employees who are 32 years old.

```
mysql> 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)
```

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', the opposite of the equality operator.

- **The greater than and lesser than operators**

Let’s retrieve the first names of all employees who are older than 32.

```
mysql> 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?

```
mysql> 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.

```
mysql> SELECT f_name, l_name FROM employee_data WHERE yos < 3;
+--------+----------+
| f_name | l_name   |
+--------+----------+
| Mary   | Anchor   |
| Edward | Sakamuro |
... ... ... ... ...
| Hal    | Simlai   |
| Joseph | Irvine   |
+--------+----------+
10 rows in set (0.00 sec)
```

- **The <= and >= operators**

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

```
mysql> 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)
```

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

```
mysql> 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)
```

<u>Assignments 2</u>

1. **What will the following SELECT statement display?**

```
SELECT * from employee_data where salary <=100000;
```

2. **Write SELECT statement to extract the ids of employees who are more than 30 years of age.**
3. **Write SELECT statement to extract the first and last names of all web designers.**
4. **List all full name of employees (last name followed by first name) who hold the title of Marketing Executive.**

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

```
mysql> 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 want to display employees whose first names begin with the alphabet **J**? SQL allows for some pattern matching with string data. For example,

```
mysql> 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 will notice that we have replaced == 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. Let's list all the employees that have **Senior** in their titles.

```
mysql> SELECT f_name, l_name 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 as follow.

```
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 3

1. **What will the following statement display**

```
SELECT f_name, l_name, salary FROM employee_data
WHERE  f_name LIKE '%k%';
```

2. **List all employees whose last names begin with P.**
3. **Display the names of all employees in the marketing division.**
4. **List the last names and titles of all programmers.**

SQL primer - Logical Operators

In this section, 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 **AND**, **OR** and **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.

```
mysql> SELECT f_name, l_name FROM employee_data
    -> WHERE salary > 70000 AND salary < 90000;
```

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

```
mysql> SELECT l_name FROM employee_data
    -> WHERE l_name LIKE 'S%' OR l_name LIKE 'A%';
```

Here is a more complex example. This example lists the names and ages of employees whose last names begin with S or P, and who are less than 30 years of age.

```
mysql> 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 is meant to separate the various logical conditions and remove any ambiguity. The **NOT** operator helps in listing all non programmers. (Programmers include Senior programmers, Multimedia Programmers and Programmers).

```
mysql> 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               |
... ... ... ... ... ... ... ... ... ... ... ... ...
| Peter   | Champion | Finance Manager            |
+---------+----------+----------------------------+
12 rows in set (0.00 sec)
```

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

```
mysql> SELECT f_name, l_name FROM employee_data
    -> WHERE yos > 3 AND age > 30;
```

Assignments 4

1. **What is displayed by the following statement?**

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

2. **List the first and last names of all employees who draw less than or equal to $90000 and are not Programmers, Senior programmers or Multimedia programmers.**
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.**

IN and BETWEEN

In this section, we consider IN and BETWEEN operators. To list employees who are **Web Designers** or **System Administrators**, we use a SELECT statement:

```
mysql> SELECT f_name, l_name, title FROM employee_data WHERE
    -> title = 'Web Designer' OR title = 'System Administrator';
```

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

```
mysql> 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**.

```
mysql> SELECT f_name, l_name, title FROM employee_data
    -> WHERE title NOT IN ('Programmer', 'Marketing Executive');
```

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

```
mysql> SELECT f_name, l_name, age FROM employee_data
    -> WHERE age BETWEEN 32 AND 40;
```

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

```
mysql> SELECT f_name, l_name, salary FROM employee_data
    -> WHERE salary NOT BETWEEN 90000 AND 150000;
```

Assignments 5

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 is displayed by the following statement?**

```
SELECT f_name, l_name, title, age
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;
```

Ordering data

This section, we look 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).

```
mysql> SELECT l_name, f_name FROM employee_data ORDER BY l_name;
+------------+---------+
| l_name     | f_name  |
+------------+---------+
| Ali        | Shahida |
| Anchor     | Mary    |
| Champion   | Peter   |
... ... ... ... ... ...
... ... ... ... ... ...
| Pillai     | Ganesh  |
| Rajabi     | Hassan  |
| Sakamuro   | Edward  |
... ... ... ... ... ...
... ... ... ... ... ...
| Simon      | Paul    |
+------------+---------+
21 rows in set (0.00 sec)
```

Here are employees sorted by age.

```
mysql> SELECT l_name, f_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 |
... ... ... ... ... ... ... ...
... ... ... ... ... ... ... ...
| 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 employees in descending order of age, we'll use the statement below.

```
mysql> SELECT f_name FROM employee_data ORDER BY age DESC;
+---------+------------+------+
| f_name  | l_name     | age  |
+---------+------------+------+
| Paul    | Simon      |   43 |
| Peter   | Champion   |   36 |
| Mike    | Harper     |   36 |
| Roger   | Lewis      |   35 |
... ... ... ... ... ... ... ...
... ... ... ... ... ... ... ...
| Hal     | Simlai     |   27 |
| Anamika | Pandit     |   27 |
| Mary    | Anchor     |   26 |
| Edward  | Sakamuro   |   25 |
+---------+------------+------+
21 rows in set (0.00 sec)
```

Note: The ascending (ASC) order is the default

Assignments 6

1. **List all employees in descending order of their years of service.**
2. **What does the following statement display?**

```
SELECT emp_id, l_name, title, age
FROM employee_data
ORDER BY title DESC, age ASC;
```

3. **Display employees (last names followed by first names) who hold the title of either "Programmer" or "Web Designer" and sort their last names alphabetically.**

Limiting data retrieval

This section, we look 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.

```
mysql> SELECT f_name, lname 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.

```
mysql> SELECT f_name, lname, 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 |
+--------+----------+------+
5 rows in set (0.01 sec)
```

Similarly, we can list the two youngest employees.

```
mysql> SELECT f_name, lname, 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;
```

```
mysql> SELECT f_name, lname 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 7

1. **List the names of 3 youngest employees in the company.**
2. **Extract the next 5 entries starting with the 4th 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;
```

Distinct Keyword

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

```
mysql> SELECT title FROM employee_data;
+----------------------------+
| title                      |
+----------------------------+
| CEO                        |
| Senior Programmer          |
| Senior Programmer          |
| Web Designer               |
| Web Designer               |
 ... ... ... ... ... ... ...
 ... ... ... ... ... ... ...
| Marketing Executive        |
| Marketing Executive        |
| Customer Service Manager   |
| Finance Manager            |
+----------------------------+
21 rows in set (0.00 sec)
```

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

```
mysql> 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**.

```
mysql> SELECT DISTINCT age FROM employee_data ORDER BY age;
```

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

Assignments 8

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

Finding the minimum and maximum values

MySQL provides inbuilt functions to find the minimum and maximum values. SQL provides 5 ***aggregate functions***. They are:

1. **MIN()**: Minimum value
2. **MAX()**: Maximum value
3. **SUM()**: The sum of values
4. **AVG()**: The average values
5. **COUNT()**: Counts the number of rows

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

Minimum value

```
mysql> SELECT MIN(salary) FROM employee_data;
+-------------+
| MIN(salary) |
+-------------+
|       70000 |
+-------------+
1 row in set (0.00 sec)
```

Maximum value

```
mysql> SELECT MAX(salary) FROM employee_data;
+-------------+
| MAX(salary) |
+-------------+
|      200000 |
+-------------+
1 row in set (0.00 sec)
```

Assignments 9

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.**

Finding the average and sum

The **SUM()** aggregate function calculates the total of values in a column. You are required to give the column name, which should be placed inside parenthesis. Let's see how much *Bignet* spends on salaries.

```
mysql> 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.

```
mysql> 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?

```
mysql> 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 arithmetic expressions.

The **AVG()** aggregate function is employed for calculating averages of data in columns.

```
mysql> 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.

```
mysql> SELECT AVG(salary) FROM employee_data;
```

Assignments 10

1. **Display the sum of ages of employees.**
2. **How would you calculate the total of years of service the employees 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;
```

Naming Columns

MySQL allows you to name the displayed columns. Then you can use more descriptive terms. This is achieved with **AS**.

```
mysql> SELECT AVG(salary) AS 'Average Salary' FROM employee_data;
+----------------+
| Average Salary |
+----------------+
|     95095.2381 |
+----------------+
1 row in set (0.00 sec)
```

Such *pseudo names* make the results clearer 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:

```
mysql> SELECT (SUM(perks)/SUM(salary) * 100) AS 'Perk Percentage'
    -> FROM employee_data;

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

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.

```
mysql> SELECT COUNT(*) FROM employee_data;

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

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

```
mysql> SELECT COUNT(*) FROM employee_data
    -> WHERE title = 'Programmer';

+----------+
| COUNT(*) |
+----------+
|        4 |
+----------+
1 row in set (0.00 sec)
```

The 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

```
mysql> SELECT title FROM employee_data GROUP BY title;
```

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.

```
mysql> SELECT title, COUNT(*) 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 (0.01 sec)
```

For the command above, MySQL first groups different titles and then executes count on each group. "GROUP BY" can also be applied to any other aggregate functions e.g., sum(), min().

Sorting the data

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

```
mysql> 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 11

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.**

HAVING Clause

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

```
mysql> 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. A short explanation is that the WHERE clause is used for filtering rows. However, AVG(salary) is not a value of any row. That is, it is an aggregated value resulted from summarizing the data of many rows. Here, the **HAVING** clause comes to our rescue.

```
mysql> 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 12

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

4 More about MySQL

The MySQL SELECT command is 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

```
mysql> SELECT version();
+-------------------------+
| version()               |
+-------------------------+
| 5.0.51b-community-nt-log |
+-------------------------+
1 row in set (0.00 sec)
```

Displaying the current date and time

```
mysql> SELECT now();
+---------------------+
| now()               |
+---------------------+
| 2012-08-18 17:12:34 |
+---------------------+

1 row in set (0.00 sec)
```

Displaying the current Day, Month and Year

```
mysql> SELECT DAYOFMONTH(CURRENT_DATE);
+--------------------------+
| DAYOFMONTH(CURRENT_DATE) |
+--------------------------+
|                       18 |
+--------------------------+
1 row in set (0.01 sec)
```

```
mysql> SELECT MONTH(CURRENT_DATE);
+---------------------+
| MONTH(CURRENT_DATE) |
+---------------------+
|                   8 |
+---------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT YEAR(CURRENT_DATE);
+--------------------+
| YEAR(CURRENT_DATE) |
+--------------------+
|               2012 |
+--------------------+
1 row in set (0.00 sec)
```

Displaying Text Strings

```
mysql> 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**.

```
mysql> SELECT 'Manish Sharma' AS Name;
+---------------+
| Name          |
+---------------+
| Manish Sharma |
+---------------+
1 row in set (0.00 sec)
```

Evaluating Expressions

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

Concatenation

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

```
mysql> 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)
```

Assignments 13

1. **Use the SELECT command to evaluate 4 X 4 X 4 and name the column "Cube of 4".**
2. **Display your name with SELECT.**

5 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.

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

MOD(x, y)
Display the remainder of x divided by y, the same as the modulus operator.

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

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

```
mysql> SELECT ABS(-4.05022);
+---------------+
| ABS(-4.05022) |
+---------------+
|       4.05022 |
+---------------+
1 row in set (0.00 sec)
```

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

```
mysql> SELECT SIGN(-34.22);
+--------------+
| SIGN(-34.22) |
+--------------+
|           -1 |
+--------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT SIGN(0);
+---------+
| SIGN(0) |
+---------+
|       0 |
+---------+
1 row in set (0.00 sec)
```

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

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

SQRT(x)
Calculate the square root of x.

```
mysql> 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.

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

```
mysql> SELECT ROUND(4.5002);
+---------------+
| ROUND(4.5002) |
+---------------+
|             5 |
+---------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT ROUND(-12.773);
+----------------+
| ROUND(-12.773) |
+----------------+
|            -13 |
+----------------+
1 row in set (0.00 sec)
```

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

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

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

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

CEILING(x)
Return the smallest integer that is great than or equal to x.

```
mysql> SELECT CEILING(23.544);
+-----------------+
| CEILING(23.544) |
+-----------------+
|              24 |
+-----------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT CEILING(-18.4);
+----------------+
| CEILING(-18.4) |
+----------------+
|            -18 |
+----------------+
1 row in set (0.00 sec)
```

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

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

6 MySQL Update Functions

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

```
UPDATE table_name
  SET column_name1 = value1,
      column_name2 = value2, ...
[WHERE conditions];
```

Obviously, like other SQL commands, you can type 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.

```
mysql> 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:
```

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

```
mysql> SELECT salary, perks FROM employee_date
    -> 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.

```
mysql> 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 careful.

Assignments 14

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.**

7 MySQL Date Data Type (I)

Till now we've dealt with text (varchar) and numbers (int) data types. To understand **date** type, we'll create one more table. Create *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

```
CREATE TABLE employee_per (e_id int unsigned not null primary key, address
varchar(60), phone int, p_email varchar(60), birth_date DATE, sex ENUM('M', 'F'),
m_status ENUM('Y','N'), s_name varchar(40), children int);

INSERT INTO employee_per (e_id, address, phone, p_email, birth_date, sex, m_status,
s_name) values (1, '202, Holder Street', 7176167, 'nettish@hotmail.com', '1972-03-
16', 'M', 'Y', 'Anamika Sharma');

INSERT INTO employee_per (e_id, address, phone, p_email, birth_date, sex, m_status,
s_name, children) values (2, '1232 Marker Hotel Road', 5553312,
'johnny4@hotmail.com', '1968-04-02', 'M', 'Y', 'Jane Donner', 3);

INSERT INTO employee_per (e_id, address, phone, p_email, birth_date, sex, m_status,
s_name, children) values (3, '90 Potter Avenue', 4321211, 'gpillai@youremail.com',
'1968-09-22', 'M', 'Y', 'Sandhya Pillai', 2);

INSERT INTO employee_per (e_id, address, phone, p_email, birth_date, sex, m_status,
s_name) values (4, '202, Holder Street', 7176167, 'twinkleinmyeyes@hotmail.com',
'1972-08-09', 'F', 'Y', 'Manish Sharma');

INSERT INTO employee_per (e_id, address, phone, p_email, birth_date, sex, m_status)
values (5, 'Apartment #8, Fuhrer Building, Cobb Street', 8973242,
'holychild@heavenlymail.com', '1974-10-13', 'F', 'N');

INSERT INTO employee_per (e_id, address, phone, p_email, birth_date, sex, m_status)
values (6, '46 Elm Street', '6666666', 'killeratlarge@elmmail.com', '1969-12-31',
'M', 'N');

INSERT INTO employee_per (e_id, address, phone, p_email, birth_date, sex, m_status,
s_name, children) values (7, '432 Mercury Avenue', 7932232, 'macmohan@hotmail.com',
'1966-8-20', 'M', 'Y', 'Mary Shelly', '3');

INSERT INTO employee_per (e_id, address, phone, p_email, birth_date, sex, m_status)
values (8, '88 Little Tokyo', 5442994, 'eddies@givememail.com', '1975-01-12', 'M',
'N');

INSERT INTO employee_per (e_id, address, phone, p_email, birth_date, sex, m_status,
s_name, children) values (9, '64 Templeton Road', 4327652,
'nandy@physicalemail.com', '1968-05-19', 'M', 'Y', 'Manika Nanda', '1');

INSERT INTO employee_per (e_id, address, phone, p_email, birth_date, sex, m_status)
values (10, '134 Metro House, Handenson Street', 5552376, 'rajabihn@hotmail.com',
'1967-07-06', 'M', 'N');

INSERT INTO employee_per (e_id, address, phone, p_email, birth_date, sex, m_status,
s_name, children) values (11, '1 Graceland, Aaron Avenue', 5433879,
'soundofsilence@boxer.net', '1957-11-04', 'M', 'Y', 'Muriel Lovelace', '4');

INSERT INTO employee_per (e_id, address, phone, p_email, birth_date, sex, m_status,
s_name, children) values (12, '97 Oakland Road', 5423311,
'kingarthur@roundtable.org', '1968-02-15', 'M', 'Y', 'Rina Brighton', 3);

INSERT INTO employee_per (e_id, address, phone, p_email, birth_date, sex, m_status,
s_name, children) values (13, '543 Applegate Lane', 3434343, 'levy@coolmail.com',
'1968-09-03', 'F', 'Y', 'Matt Shikari', '2');
```

```
INSERT INTO employee_per (e_id, address, phone, p_email, birth_date, sex, m_status)
values (14, '765 Flasher Street', 7432433, 'tinkertone@email.com', '1965-04-28',
'M', 'N');

INSERT INTO employee_per (e_id, address, phone, p_email, birth_date, sex, m_status,
s_name, children) values (15, '98 Gunfoundry', 6500787, 'danny@foolhardy.com',
'1966-06-23', 'M', 'Y', 'Betty Cudly', 3);

INSERT INTO employee_per (e_id, address, phone, p_email, birth_date, sex, m_status,
s_name, children) values (16, '#5 Comely Homes', 5432132, 'mikeharper@coldmail.com',
'1964-03-06', 'M', 'Y', 'Stella Stevens', 2);

INSERT INTO employee_per (e_id, address, phone, p_email, birth_date, sex, m_status,
s_name, children) values (17, '652 Devon Building, 6th Jake Avenue', 5537885,
'mona@darling.com', '1970-04-18', 'F', 'Y', 'Edgar Alan', 1);

INSERT INTO employee_per (e_id, address, phone, p_email, birth_date, sex, m_status)
values (18, 'Apartment #9, Together Towers', 5476565, 'odessey2000@hotmail.com',
'1973-10-09', 'M', 'N');

INSERT INTO employee_per (e_id, address, phone, p_email, birth_date, sex, m_status)
values (19, 'Apartment #9, Together Towers', 5476565, 'jirvine@hotteremail', '1973-
1-20', 'M', 'N');

INSERT INTO employee_per (e_id, address, phone, p_email, birth_date, sex, m_status)
values (20, '90 Comfy Town', 7528326, 'helper@more.org', '1968-01-25', 'F', 'N');

INSERT INTO employee_per (e_id, address, phone, p_email, birth_date, sex, m_status,
s_name, children) values (21, '4329 Eucalyptus Avenue', 4254863,
'moneymatters@coldcash.com', '1964-06-13', 'M', 'Y', 'Ruby Richer', 2);
```

Inserting data into employee_per table with employee_per.dat file

On Windows

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

```
mysql dbname < employee_per.dat
```

On Linux

1. Move to the directory of the downloaded file. Issue the following command

```
mysql dbname < employee_per.dat –u username -p
```

2. Enter your password.

A new table, **employee_per,** will be created. The details of the table can be displayed with DESCRIBE (or desc) 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 (0.00 sec)
```

Notice that column **birth_date** has **date** as column type.

e_id:	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, **M**ale or **F**emale
m_status:	Marital status: **Y**es or **N**o.
s_name:	Name of Spouse (NULL if employee is unmarried)
children:	Number of children (NULL if employee is unmarried)

Characteristics of Date

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.

Operations on Date

Date column type allows for several operations such as sorting, testing conditions using comparison operators etc.

Using = or != operators.

```
mysql> 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

```
mysql> 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 ranges

```
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)
```

The following command gets the same result as above.

```
mysql> SELECT e_id, birth_date FROM employee_per
    -> WHERE birth_date >= '1969-01-01' AND birth_date <= '1974-01-01';
```

Assignments 15

1. **List all employee ids and birth dates that were born before 1965.**
2. **Display Ids and birth dates of employees born in and between 1970 and 1972.**

Using Date to sort data

```
mysql> 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 |
 ... ... ... ... ...
|   19 | 1973-01-20 |
|   18 | 1973-10-09 |
|    5 | 1974-10-13 |
|    8 | 1975-01-12 |
+------+------------+
21 rows in set (0.00 sec)
```

Selecting data using Date

Here is how to select employees born in March.

```
mysql> 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.

```
mysql> 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.

```
mysql> 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)
```

```
mysql> 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)
```

Using current date

We had seen in the session on 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.

```
mysql> SELECT e_id, birth_date FROM employee_per
    -> WHERE MONTH(birth_date) = MONTH(CURRENT_DATE);
+------+------------+
| e_id | birth_date |
+------+------------+
|   10 | 1967-07-06 |
+------+------------+
1 rows in set (0.00 sec)
```

Assignments 16

1. **List ids, birth dates and emails of employees born in April.**
2. **Display Ids, birth dates and spouse names of employees born in 1969 and sort the entries on the basis of their spouse names.**
3. **List the employee 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.**

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. Comparisons for NULL can be done with **IS NULL** or **IS NOT NULL**.

```
mysql> 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 17

1. **List the ids and spouse names of all employees who are married.**
2. **Change the above sql statement 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 of all employees.**

8 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 at least one column in the two tables contains 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.

```
mysql> 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     |
 ... ... ... ... ... ... ... ... ...
| 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 rows where the *emp_id* and *e_id* are same. Without putting "**emp_id = e_id**" in the WHERE clause, all possible row combinations of the two tables will be returned (also known as a cross join).

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**.

```
mysql> 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  |
... ... ... ... ... ... ... ... ...
 ... ... ... ... ... ... ... ... ...
| Mike Harper     | Stella Stevens  |
| Monica Sehgal   | Edgar Alan      |
| Peter Champion  | Ruby Richer     |
+-----------------+-----------------+
13 rows in set (0.00 sec)
```

Caution!! **The following commands in Section 8 and 9 are used to delete your data, tables and databases. It is not necessary to practice these commands; otherwise your data will be lost.**

9 Delete & Drop entries from MySQL

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.

```
mysql> DELETE from employee_data
    -> WHERE emp_id = 10;

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

10 Dropping tables

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

```
mysql> 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.

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

Now, this table is completely deleted and will not be shown with SHOW TABLES.

11 MySQL database Column Types

The three major types of column types used in MySQL are

1. Integer
2. Text
3. Date

Choosing a column data type is very important in order to achieve speed, effective storage and retrieval.

Numeric Column Types

In addition to **int** (Integer data type), MySQL also provides floating-point and double precision numbers. Each integer type can 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)
- **TIME**: hh:mm:ss (Hours:Minutes:Seconds)
- **DATETIME**: YYYY-MM-DD hh:mm:ss (Date and time separated by a space character)
- **TIMESTAMP**: YYYYMMDDhhmmss
- **YEAR**: YYYY (4 digit year)

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
- **MEDIUMTEXT**: medium size text, case insensitive
- **LONGTEXT**: really long text, case insensitive
- **TINYBLOB**: Blob means a **B**inary **L**arge **OB**ject. 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**: **Enum**eration data type has fixed values and the column can take only one value from the given set. The values are placed in parenthesis following ENUM declaration. For example, the marital status column we encountered in *employee_per* table is of an ENUM.

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

Practice

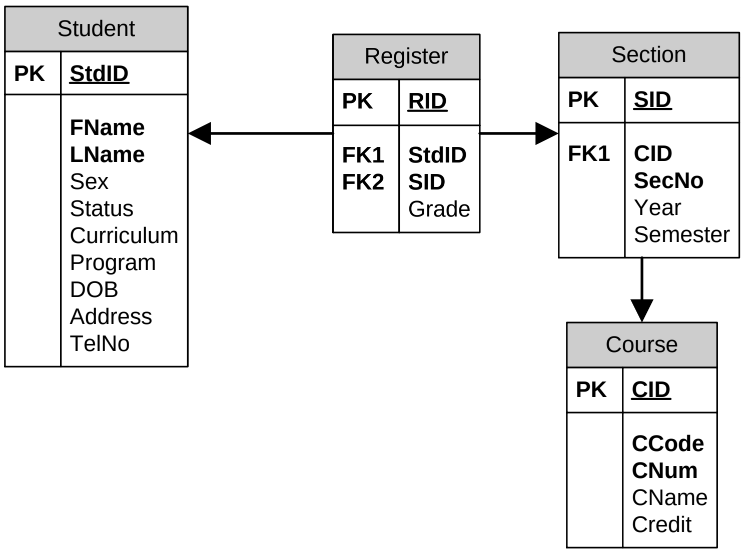

Figure 1: Entity-Relationship Diagram for a Small Registration Database

The SQL commands for creating the table and insert the data are already provided for you. You can import the SQL commands to your database by using the above command as introduced in manual ***or*** by using these commands after you are at the *mysql prompt*.

```
mysql> source createtbls.sql
mysql> LOAD DATA LOCAL INFILE 'Student.txt' INTO TABLE Student;
mysql> LOAD DATA LOCAL INFILE 'Course.txt' INTO TABLE Course;
mysql> LOAD DATA LOCAL INFILE 'Section.txt' INTO TABLE Section;
mysql> LOAD DATA LOCAL INFILE 'Register.txt' INTO TABLE Register;
```

1. Write a query to show a list of IT students and order by their first name.
2. Write a query to show a list of students who do not live in London.
3. Write a query to calculate average GPA for each semester for each student.
4. Write a query to count the number of courses each student took and got grade.
5. Write a query to find the maximum grade that each student used to get.
6. Write a query to show a list of students with the order of their average GPA.
7. Write a query to show a list of students who do not register any course until now.
8. Write a query to show the age of each student.