

# Implementation in MySQL

## ABOUT MYSQL

With over 10 million installations, MySQL is probably the most popular database management system for web servers. Developed in the mid-1990s, it's now a mature technology that powers many of today's most-visited Internet destinations. It's not only free to use but also extremely powerful and exceptionally fast. MySQL is also highly scalable, which means that it can grow with website.

The SQL in MySQL stands for Structured Query Language. It is the standard relational database language and includes features for defining the structure of the data, modifying data in the database and for specifying security constraints. Apart from MySQL, it is used in Oracle and Microsoft SQL Server.

There are three main ways to interact with MySQL: using a command line, via a web interface such as phpMyAdmin, and through a programming language like PHP. In this lab, we'll cover the command line interaction.

## USING MYSQL COMMAND LINE IN WINDOWS

If you installed the WAMP Server, you will be able to access the MySQL executable from the following directory:

- C:\wamp\bin\mysql\mysql5.6.17\bin

By default, the initial MySQL user will be root and will not have had a password set. To enter MySQL's command-line interface, select Start→Run, enter CMD into the Run box, and press Return. This will call up a Windows Command Prompt. From there, enter following:

- "C:\wamp\bin\mysql\mysql5.6.17\bin" -u root

This command tells MySQL to log you in as user root, without a password. You will now be logged into MySQL and can start entering commands as shown in Figure 5.1. Please note that your Wamp server must be running, otherwise MySQL command prompt will not show up.

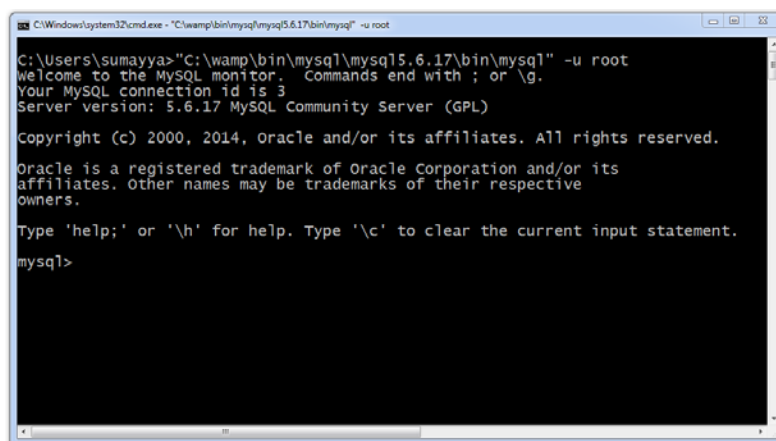


Figure 6.1 – MySQL Command Line Interface

## The Semicolon

MySQL uses semicolon to separate or end commands. If you forget to enter it, MySQL will issue a prompt and wait for you to do so. The required semicolon was made part of the syntax to let you enter multiple line commands, which can be convenient because some commands get quite long. It also allows you to issue more than one command at a time by placing a semicolon after each one. The interpreter gets them all in a batch when you press the Enter (or Return) key and executes them in order.

There are six different prompts in MySQL as shown in Table 6.1.

TABLE 6.1 MYSQL COMMAND PROMPTS

MySQL prompt	Meaning
mysql>	Ready and waiting for a command
->	Waiting for the next line of a command
'>	Waiting for the next line of a string started with a single quote
">	Waiting for the next line of a string started with a double quote
`>	Waiting for the next line of a string started with a backtick
/*>	Waiting for the next line of a comment started with /*

*Note: If you are partway through entering a command and decide you don't wish to execute it after all, whatever you do don't press Control-C! That will close the program. Instead, enter \c and press Return.*

## MYSQL COMMAND CATEGORIES

SQL statements can be grouped into four general categories: **Data definition language (DDL)**, **Data manipulation language (DML)**, **Transaction control language (TCL)**, and **Data control language (DCL)**. Each one of these is briefly discussed here.

### Data Definition Language (DDL)

The overall design of a database is called the database schema. A database schema is specified by a set of definitions that are expressed using a data definition language. It is used for structuring the database. We use the data definition language statements **CREATE**, **ALTER**, **DROP**, **TRUNCATE** to create new objects, alter the structure of existing objects, completely remove objects from system, or delete all rows permanently from the table leaving the structure of the table.

### Data Manipulation Language (DML)

These commands are the most frequently used SQL commands. They are used to query and manipulate existing objects like tables. DML commands are: **SELECT**, **UPDATE**, **INSERT**, and **DELETE**.

## Transaction Control Language (TCL)

The changes made to the database are known as transaction. Transactions can be made permanent to a database by commit. The various commands in TCL are: **COMMIT**, **SAVEPOINT**, and **ROLLBACK**.

## Data Control Language (DCL)

DCL statements are used for securing the database. DCL statements such as **GRANT** and **REVOKE** control access to database and affirm or revoke database transactions. It provides the user with privilege commands. The owner of the database can grant privileges or withdraw (revoke) privilege to other database users so that they can perform the operations accordingly on the tables.

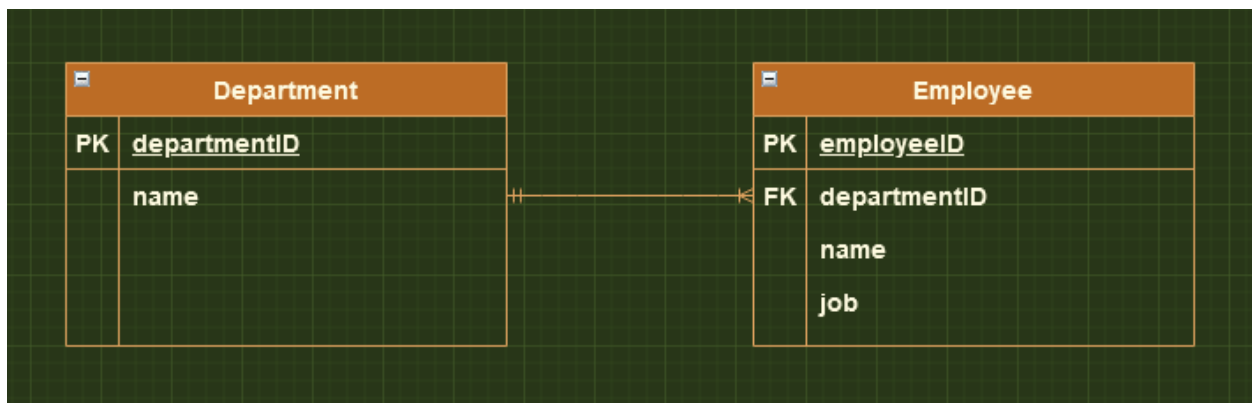


Figure 6.2 – Department-Employee Relational Schema

## USE OF MYSQL COMMANDS

Consider the department and employee Relational Schema given in Figure 6.2, we will create its MySQL implementation in this section.

**Command:** `create database sam_db;`

**Detail:** This command creates a new database sam\_db.

**Command:** `show databases;`

**Detail:** This command lists all the existing databases in mysql environment.

**Command:** `use sam_db;`

**Detail:** This command switches over from default mysql database to specified database. In current case, switched database is sam\_db.

**Command:** `show tables;`

**Detail:** This command lists all the existing tables in current database i.e. sam\_db. Since there are no tables yet in database, so empty set is shown.

**Command:** `create table department(`

`departmentID int not null auto_increment,`

`name varchar(30),`

`primary key(departmentID));`

**Detail:** This command creates a new table department in sam\_db database. It consists of two columns departmentID and name. The datatype of each column is specified along with its size. The primary key of table has been set to departmentID attribute. Note that auto\_increment option automatically increments the primary key if it's not specified during record entries in table.

**Command:** `create table employee(`

`employeeID int not null auto_increment,`

`name varchar(80),`

`job varchar(30),`

`departmentID int default 0,`

`primary key(employeeID),`

`foreign key(departmentID) references department(departmentID));`

**Detail:** This command creates a new table employee in sam\_db database. It consists of four columns employeeID, name, job, & departmentID where employeeID has been set as primary key and departmentID as foreign key.

**Command:** `show tables;`

**Detail:** This command shows all the existing tables in the sam\_db database. Since two tables i.e. department and employee are created in above commands, so these two are listed.

**Command:** `describe department;`

**Detail:** This command describes the table structure of department table. Description includes field or attribute names, their respective data types, primary key information, default value of any field and extra options (such as auto\_increment).

**Command:** `insert into department values`  
(10, 'Computer Systems Engineering'),  
(15, 'Electrical Engineering'),  
(20, 'Chemical Engineering'),  
(25, 'Mining Engineering');

**Detail:** This command populates multiple records in department table.

**Command:** `insert into employee values`  
(1, 'ABC', 'Lecturer', 10),  
(3, 'ACB', 'Lecturer', 10),  
(4, 'XYZ', 'Assistant Professor', 10),  
(5, 'CAB', 'Lecturer', 15);

**Detail:** This command populates multiple records in employee table.

**Command:** `select * from employee;`

**Detail:** This command selects records from employee table where \* means to select and show values from all the columns.

**Command:** `select * from employee where job = 'Lecturer';`

**Detail:** This command selects all the records from employee table whose job title is 'Lecturer'.

**Command:** `select * from department where departmentID > 15;`

**Detail:** This command selects those records from department table whose departmentID is greater than 15.

**Command:** `select * from department where departmentID = 10 or departmentID = 20;`

**Detail:** This command shows departmental record of those departments whose departmentID is either 10 or 20.

**Command:** `select employee.name, department.name  
from employee, department,  
where employee.departmentID = department.departmentID;`

**Detail:** There are many departments in an organization and a given department contains many employees. We are interested to find employee name and its respective department name, where an employee works. In employee table, department information is shown by departmentID that is a foreign key. Thus, to find accurate department name, only those records must be listed where departmentID in both table matches. This is done by condition specified after where keyword.

**Command:** `select employee.name as empName, department.name as deptName  
from employee, department,  
where employee.departmentID = department.departmentID;`

**Detail:** This command shows the use of alias feature of MySQL. The given command is same as the above one except that column names are specified by alias names i.e. employee name is represented by empName and department name is represented by deptName.

**Command:** `delete from department  
where departmentID=25;`

**Detail:** This command deletes a record from department table, whose departmentID is 25.

**Command:** `select * from department;`

**Detail:** This command selects records from department table where \* means to select and show values from all the columns. Note that the records shown don't contain department having 25 as it is deleted in above command.

**Command:** `select job from employee;`

**Detail:** This command selects and shows content of 'job' column in employee table.

**Command:** `select distinct job from employee;`

**Detail:** To select different jobs and not the repeated ones, given command is used. Distinct keyword selects only unique jobs present in job column.

**Command:** `select count(job) from employee;`

**Detail:** This command counts the total number of entries in job column of employee table.

**Command:** `select count(distinct job) from employee;`

**Detail:** To count the number of unique jobs only, distinct keyword is used with count function. The result of this command is count of unique jobs in job column.

**Command:** `select count(*) job,  
from employee  
group by job;`

**Detail:** To determine how many employees are hired for a specific job title, following command is used. It finds the count for all the unique jobs.

**Command:** `select count(*) job,  
from employee  
group by job  
having count(*)=1;`

**Detail:** This command is the same as above one except the 'having' keyword. Here we are interested to find that *job title*, whose count equals 1 (i.e. find a job for which only one employee is hired?). If such job exists whose employee count is one, then it is returning the count; otherwise an empty set is returned.

**Command:** `select *`  
`from employee`  
`order by job asc;`

**Detail:** By default, records are sorted according to primary key in ascending order. But it can be changed. Given command lists all the entries in employee table, such that sorting order of the records is specified by 'job' field.

**Command:** `select *`  
`from employee`  
`limit 3;`

**Detail:** To select specific number of records from a given table, limit keyword can be used. The given command selects and displays three records from employee table

**Command:** `update employee`  
`set job='Lab Engineer'`  
`where employeeID=3;`

**Detail:** This command updates a record from employee table, sets job to 'Lab Engineer' from 'Lecturer' where employeeID is 3.

So, in this section we have covered the following commands: **create database, show databases, use, create table, show tables, describe, insert, update, delete, and select.**

## INTEGRITY CONSTRAINTS USING MYSQL COMMANDS

In this section, we'll apply integrity constraints in created database tables using MySQL commands.

**Step 1:** `alter table employee`  
`drop foreign key employee_ibfk_1;`

**Detail:** This command drops the foreign key from the employee table.



**Step 2:**      `alter table employee`  
`add foreign key (departmentID) references department(departmentID)`  
`on update cascade;`

**Detail:**      This command applies the foreign key on the employee table with integrity constraint update cascade. This means that if the foreign key is updated in the parent table i.e. department then it'll be reflected in child table i.e. employee. Similarly, other integrity constraints i.e. on update restrict, on update set null, and on update set default can be applied.

**Step 3:**      `update department`  
`set departmentID=12`  
`where departmentID=10;`

**Detail:**      This command updates a record from department table, sets departmentID to 12 from 10 where departmentID is 10.

**Step 4:**      `select * from department;`

**Detail:**      This command lists all the entries in the department table. Note that now the departmentID starts from 12 instead of 10 due to previous command.

**Step 5:**      `select * from employee;`

**Detail:**      This command lists all the entries in the employee table. Note that all the employees of Computer Systems Engineering have foreign key value of 12 instead of 10 due to update cascade integrity constraint.

**Command:**   `alter table employee`  
`add foreign key (departmentID) references department(departmentID)`  
`on update set null;`

**Detail:**      This command applies the foreign key on the employee table with integrity constraint update set null. This means that if the foreign key is updated in the parent table i.e. department then it'll be set to null in the child table i.e. employee. Step 1, 3, 4, and 5 are the same as above. Only step has been changed.