Chapter 11: Database connectivity (MySQL)

To build the real world applications, connecting with the databases is the necessity for the programming languages. However, python allows us to connect our application to the databases like MySQL, SQLite, MongoDB, and many others.

We will discuss MySQL first and we will perform the database operations on the AWS platform. Then we will use python programming language to connect to the database using python-mysql drivers

Install Mysql on AWS

```
sudo wget http://repo.mysql.com/mysql-community-release-el7-5.noarch.rpm

sudo rpm -ivh mysql-community-release-el7-5.noarch.rpm

sudo yum install mysql-server -y

sudo service mysqld start

sudo systemctl enable mysqld


mysql -u root
```

Performing Database operations

Here is a simple query that asks the server to tell you its version number and the current date.

```
mysql> SELECT VERSION(), CURRENT_DATE;
mysql> select version(), current_date;
mysql> SeLeCt vErSiOn(), current_DATE;
```

This query illustrates several things about **mysql**:

- A query normally consists of an SQL statement followed by a semicolon. (There are some exceptions where a semicolon may be omitted. QUIT, mentioned earlier, is one of them. We'll get to others later.)
- When you issue a query, **mysql** sends it to the server for execution and displays the results, then prints another mysql> prompt to indicate that it is ready for another query.
- **mysql** displays query output in tabular form (rows and columns). The first row contains labels for the columns. The rows following are the query results. Normally, column labels are the names of the columns you fetch from database tables. If you're retrieving the value of an expression rather than a table column (as in the example just shown), mysql labels the column using the expression itself.
- **mysql** shows how many rows were returned and how long the query took to execute, which gives you a rough idea of server performance. These values are imprecise

because they represent wall clock time (not CPU or machine time), and because they are affected by factors such as server load and network latency.

Here is another query. It demonstrates that you can use mysql as a simple calculator:

```
mysql> SELECT SIN(PI()/4), (4+1)*5;
+------------------+---------+
| SIN(PI()/4)      | (4+1)*5 |
+------------------+---------+
| 0.70710678118655 |      25 |
+------------------+---------+
1 row in set (0.02 sec)
```

The queries shown thus far have been relatively short, single-line statements. You can even enter multiple statements on a single line. Just end each one with a semicolon:

```
SELECT VERSION(); SELECT NOW();
+------------+
| VERSION() |
+------------+
| 5.6.51    |
+------------+
1 row in set (0.00 sec)

+---------------------+
| NOW()               |
+---------------------+
| 2022-02-02 05:40:49 |
+---------------------+
1 row in set (0.00 sec)
```

Here is a simple multiple-line statement:

```
select
    -> user(),
    -> current_date;
+------------------------------------------------------+--------------+
| user()                                               | current_date |
+------------------------------------------------------+--------------+
| newuser@ip-172-31-10-241.ap-south-1.compute.internal | 2022-02-02   |
+------------------------------------------------------+--------------+
1 row in set (0.00 sec)
```

Creating and Selecting a Database

Database names are case-sensitive. so you must always refer to your database as bigdata not as Bigdata.
Creating a database does not select it for use; you must do that explicitly. To make bigdata the current database, use the "use" statement:

Creating a database in mysql

```
mysql -uroot
Create database empdb;
Use empdb;
```

```
SELECT DATABASE();
```

Creating a Table

```
CREATE TABLE employee (

  empid int(11) PRIMARY KEY,

  first_name varchar(20) DEFAULT NULL,

  last_name varchar(20) DEFAULT NULL,

  gender varchar(1) DEFAULT NULL,

  email varchar(20) DEFAULT NULL,

  dob varchar(12) DEFAULT NULL,

  age int(11) DEFAULT NULL,

  salary double DEFAULT NULL,

  deptid int(11) DEFAULT NULL

);
```

Listing a Table

```
mysql>show tables
```

Knowing the structure of the table

```
mysql> desc employee
```

Please refer to the datatypes at the following mysql website
https://dev.mysql.com/doc/refman/8.0/en/data-types.html

Loading Data into a Table

After creating your table, you need to populate it. Use either Load Data or insert statement. Because you are beginning with an empty table, an easy way to populate it is to create a text file containing a row for each of your employees, then load the contents of the file into the table with a single statement.
You could create a text file employee.txt containing one record per line, with values separated by tabs, and given in the order in which the columns were listed in the CREATE TABLE statement. If you want to add the data using comma separated values then add “fields terminated by ‘,’” at the end of the load statement.

```
LOAD DATA LOCAL INFILE 'employee.txt' into table employee;
```

Note: If you get an error like below follow the below procedure and then run the load statements
ERROR 3950 (42000): Loading local data is disabled; this must be enabled on both the client and server side

Step1: Login to mysql using the following command
`mysql --local-infile=1 -u root`

Step2 : set the following global varialble
`SET GLOBAL local_infile=1;`

When you want to add new records one at a time, the INSERT statement is useful. In its simplest form, you supply values for each column, in the order in which the columns were listed in the CREATE TABLE statement.

```
insert into employee values (100,"Sudheer",12345,1);
insert into employee values (200,"Suchitra",65432,2);
select * from employee;
```

Retrieving Information from a Table

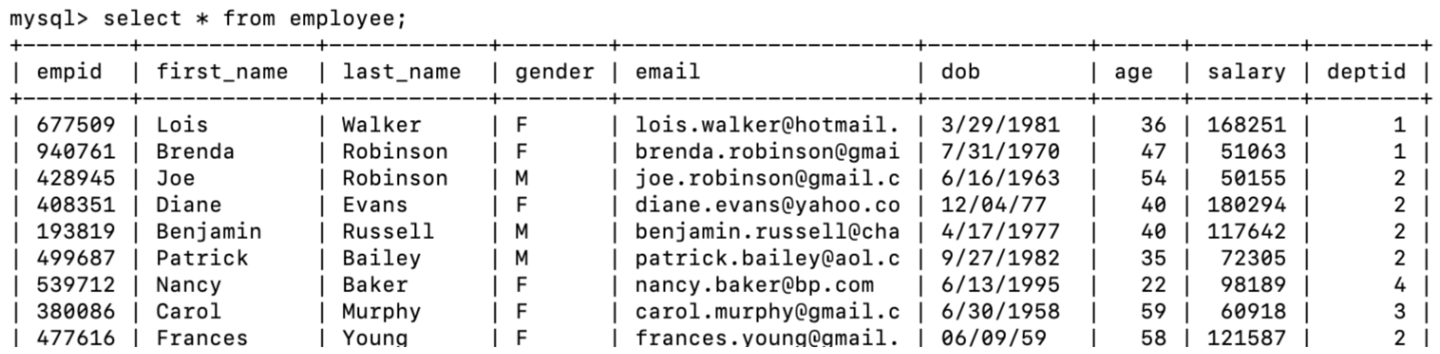
The SELECT statement is used to pull information from a table. The general form of the statement is:

```
SELECT what_to_select
FROM which_table
WHERE conditions_to_satisfy;
```

`what_to_select` indicates what you want to see. This can be a list of columns, or * to indicate "all columns." ***`which_table`*** indicates the table from which you want to retrieve data. The WHERE clause is optional. If it is present, ***`conditions_to_satisfy`*** specifies one or more conditions that rows must satisfy to qualify for retrieval.

Selecting All Data

The simplest form of SELECT retrieves everything from a table:

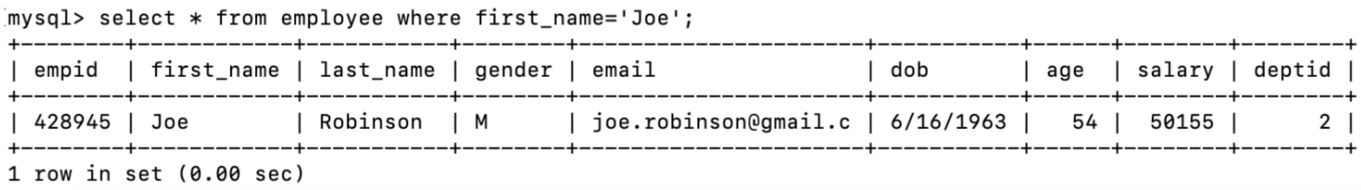


```
mysql> select * from employee;
+--------+-------------+------------+--------+-----------------------+------------+------+--------+--------+
| empid  | first_name  | last_name  | gender | email                 | dob        | age  | salary | deptid |
+--------+-------------+------------+--------+-----------------------+------------+------+--------+--------+
| 677509 | Lois        | Walker     | F      | lois.walker@hotmail.  | 3/29/1981  |   36 | 168251 |      1 |
| 940761 | Brenda      | Robinson   | F      | brenda.robinson@gmai  | 7/31/1970  |   47 |  51063 |      1 |
| 428945 | Joe         | Robinson   | M      | joe.robinson@gmail.c  | 6/16/1963  |   54 |  50155 |      2 |
| 408351 | Diane       | Evans      | F      | diane.evans@yahoo.co  | 12/04/77   |   40 | 180294 |      2 |
| 193819 | Benjamin    | Russell    | M      | benjamin.russell@cha  | 4/17/1977  |   40 | 117642 |      2 |
| 499687 | Patrick     | Bailey     | M      | patrick.bailey@aol.c  | 9/27/1982  |   35 |  72305 |      2 |
| 539712 | Nancy       | Baker      | F      | nancy.baker@bp.com    | 6/13/1995  |   22 |  98189 |      4 |
| 380086 | Carol       | Murphy     | F      | carol.murphy@gmail.c  | 6/30/1958  |   59 |  60918 |      3 |
| 477616 | Frances     | Young      | F      | frances.young@gmail.  | 06/09/59   |   58 | 121587 |      2 |
```

This form of SELECT uses *, which is shorthand for "select all columns." This is useful if you want to review your entire table, for example, after you've just loaded it with your initial data set.

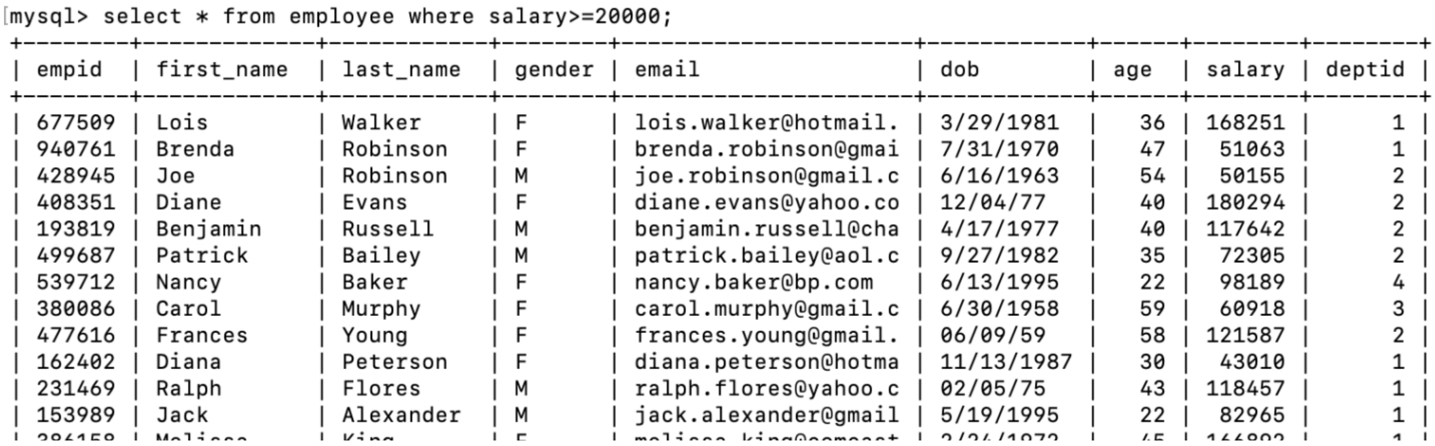
Selecting Particular Rows

You can select only particular rows from your table. For example, if you want to verify the user Ravi, select record like this:



```
mysql> select * from employee where first_name='Joe';
+--------+------------+-----------+--------+-----------------------+-----------+------+--------+--------+
| empid  | first_name | last_name | gender | email                 | dob       | age  | salary | deptid |
+--------+------------+-----------+--------+-----------------------+-----------+------+--------+--------+
| 428945 | Joe        | Robinson  | M      | joe.robinson@gmail.c | 6/16/1963 |   54 |  50155 |      2 |
+--------+------------+-----------+--------+-----------------------+-----------+------+--------+--------+
1 row in set (0.00 sec)
```

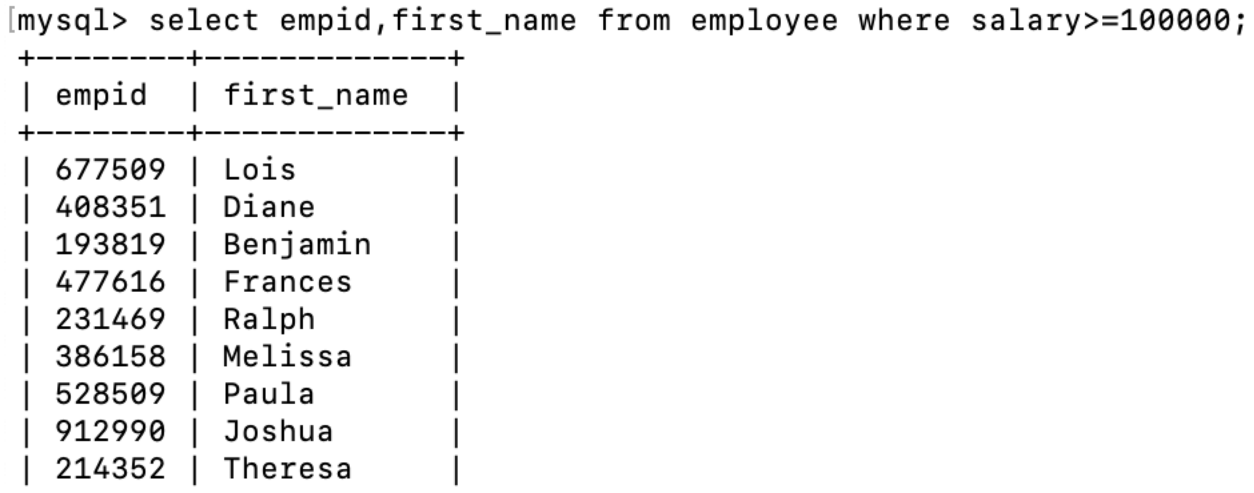
You can also filter the rows based on the salary as below



```
mysql> select * from employee where salary>=20000;
+--------+-------------+------------+--------+----------------------+------------+------+--------+--------+
| empid  | first_name  | last_name  | gender | email                | dob        | age  | salary | deptid |
+--------+-------------+------------+--------+----------------------+------------+------+--------+--------+
| 677509 | Lois        | Walker     | F      | lois.walker@hotmail. | 3/29/1981  |   36 | 168251 |      1 |
| 940761 | Brenda      | Robinson   | F      | brenda.robinson@gmai | 7/31/1970  |   47 |  51063 |      1 |
| 428945 | Joe         | Robinson   | M      | joe.robinson@gmail.c | 6/16/1963  |   54 |  50155 |      2 |
| 408351 | Diane       | Evans      | F      | diane.evans@yahoo.co | 12/04/77   |   40 | 180294 |      2 |
| 193819 | Benjamin    | Russell    | M      | benjamin.russell@cha | 4/17/1977  |   40 | 117642 |      2 |
| 499687 | Patrick     | Bailey     | M      | patrick.bailey@aol.c | 9/27/1982  |   35 |  72305 |      2 |
| 539712 | Nancy       | Baker      | F      | nancy.baker@bp.com   | 6/13/1995  |   22 |  98189 |      4 |
| 380086 | Carol       | Murphy     | F      | carol.murphy@gmail.c | 6/30/1958  |   59 |  60918 |      3 |
| 477616 | Frances     | Young      | F      | frances.young@gmail. | 06/09/59   |   58 | 121587 |      2 |
| 162402 | Diana       | Peterson   | F      | diana.peterson@hotma | 11/13/1987 |   30 |  43010 |      1 |
| 231469 | Ralph       | Flores     | M      | ralph.flores@yahoo.c | 02/05/75   |   43 | 118457 |      1 |
| 153989 | Jack        | Alexander  | M      | jack.alexander@gmail | 5/19/1995  |   22 |  82965 |      1 |
```

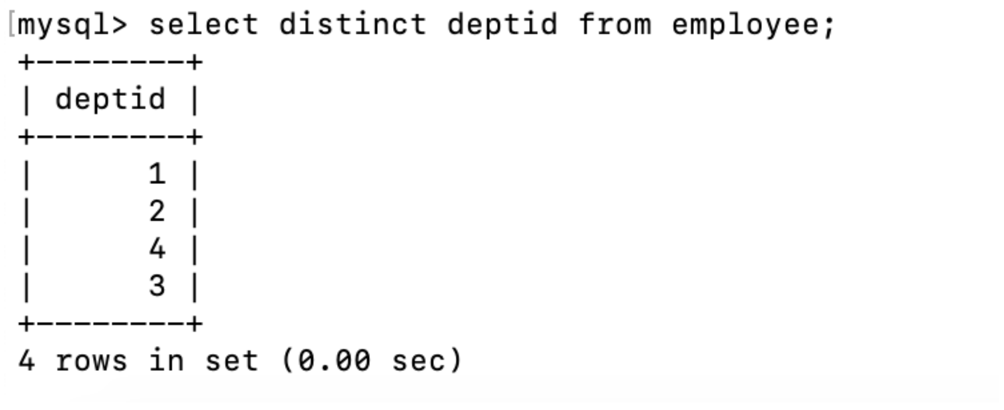
Selecting Particular Columns

If you do not want to see entire rows from your table, just name the columns in which you are interested, separated by commas.



```
mysql> select empid,first_name from employee where salary>=100000;
+--------+-------------+
| empid  | first_name  |
+--------+-------------+
| 677509 | Lois        |
| 408351 | Diane       |
| 193819 | Benjamin    |
| 477616 | Frances     |
| 231469 | Ralph       |
| 386158 | Melissa     |
| 528509 | Paula       |
| 912990 | Joshua      |
| 214352 | Theresa     |
```

If some of the column values appear more than once, retrieve each unique output record just once by adding the keyword `DISTINCT`:


```
[mysql> select distinct deptid from employee;
+--------+
| deptid |
+--------+
|      1 |
|      2 |
|      4 |
|      3 |
+--------+
4 rows in set (0.00 sec)
```

Constraints in MySQL

Constraints are the rules that we can apply on the type of data in a table. That is, we can specify the limit on the type of data that can be stored in a particular column in a table using constraints.
The available constraints in SQL are:

- **NOT NULL**: This constraint tells that we cannot store a null value in a column. That is, if a column is specified as NOT NULL then we will not be able to store null in this particular column any more.
- **UNIQUE**: This constraint when specified with a column, tells that all the values in the column must be unique. That is, the values in any row of a column must not be repeated.
- **PRIMARY KEY**: A primary key is a field which can uniquely identify each row in a table. And this constraint is used to specify a field in a table as primary key.
- **FOREIGN KEY**: A Foreign key is a field which can uniquely identify each row in a another table. And this constraint is used to specify a field as Foreign key.
- **CHECK**: This constraint helps to validate the values of a column to meet a particular condition. That is, it helps to ensure that the value stored in a column meets a specific condition.
- **DEFAULT**: This constraint specifies a default value for the column when no value is specified by the user.

How to specify constraints?

We can specify constraints at the time of creating the table using CREATE TABLE statement. We can also specify the constraints after creating a table using ALTER TABLE statement.
Syntax:
Below is the syntax to create constraints using CREATE TABLE statement at the time of creating the table.

```
CREATE TABLE sample_table

(

column1 data_type(size) constraint_name,

column2 data_type(size) constraint_name,

column3 data_type(size) constraint_name,

....

);
```

sample_table: Name of the table to be created.
data_type: Type of data that can be stored in the field.
constraint_name: Name of the constraint. for example- NOT NULL, UNIQUE, PRIMARY KEY etc.

1. NOT NULL –
If we specify a field in a table to be NOT NULL. Then the field will never accept null value. That is, you will be not allowed to insert a new row in the table without specifying any value to this field.

2. UNIQUE –
This constraint helps to uniquely identify each row in the table. i.e. for a particular column, all the rows should have unique values. We can have more than one UNIQUE columns in a table.

3. PRIMARY KEY –
Primary Key is a field which uniquely identifies each row in the table. If a field in a table as primary key, then the field will not be able to contain NULL values as well as all the rows should have unique values for this field. So, in other words we can say that this is combination of NOT NULL and UNIQUE constraints.
A table can have only one field as primary key.

4. FOREIGN KEY –
Foreign Key is a field in a table which uniquely identifies each row of a another table. That is, this field points to primary key of another table. This usually creates a kind of link between the tables.

Orders

O_ID	ORDER_NO	C_ID
1	2253	3
2	3325	3
3	4521	2
4	8532	1

Customers

C_ID	NAME	ADDRESS
1	RAMESH	DELHI
2	SURESH	NOIDA
3	DHARMESH	GURGAON

As we can see clearly that the field C_ID in Orders table is the primary key in Customers table, i.e. it uniquely identifies each row in the Customers table. Therefore, it is a Foreign Key in Orders table.

Syntax:

```
CREATE TABLE Orders

(

O_ID int NOT NULL,

ORDER_NO int NOT NULL,

C_ID int,

PRIMARY KEY (O_ID),

FOREIGN KEY (C_ID) REFERENCES Customers(C_ID)

)
```

5. CHECK –

Using the CHECK constraint we can specify a condition for a field, which should be satisfied at the time of entering values for this field.

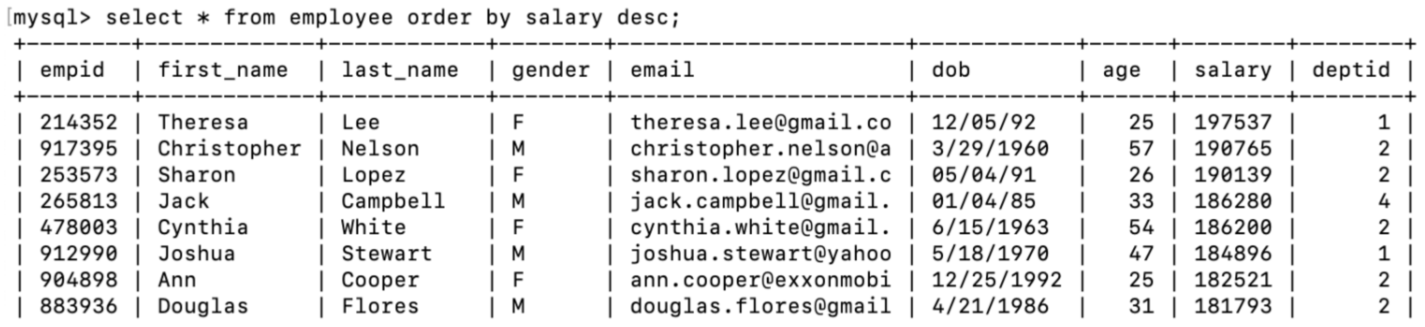
6. DEFAULT –

This constraint is used to provide a default value for the fields. That is, if at the time of entering new records in the table if the user does not specify any value for these fields then the default value will be assigned to them.
For example, the query will create a table named Student and specify the default value for the field AGE as 18.

```
CREATE TABLE Student

(

ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,

AGE int DEFAULT 18

);
```

Sorting Rows

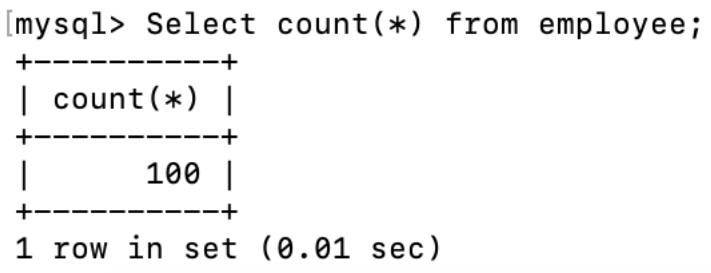
You may have noticed in the preceding examples that the result rows are displayed in no particular order. It is often easier to examine query output when the rows are sorted in some meaningful way. To sort a result, use an ORDER BY clause.



```
[mysql> select * from employee order by salary desc;
+--------+-------------+------------+--------+----------------------+------------+------+--------+--------+
| empid  | first_name  | last_name  | gender | email                | dob        | age  | salary | deptid |
+--------+-------------+------------+--------+----------------------+------------+------+--------+--------+
| 214352 | Theresa     | Lee        | F      | theresa.lee@gmail.co | 12/05/92   |   25 | 197537 |      1 |
| 917395 | Christopher | Nelson     | M      | christopher.nelson@a | 3/29/1960  |   57 | 190765 |      2 |
| 253573 | Sharon      | Lopez      | F      | sharon.lopez@gmail.c | 05/04/91   |   26 | 190139 |      2 |
| 265813 | Jack        | Campbell   | M      | jack.campbell@gmail. | 01/04/85   |   33 | 186280 |      4 |
| 478003 | Cynthia     | White      | F      | cynthia.white@gmail. | 6/15/1963  |   54 | 186200 |      2 |
| 912990 | Joshua      | Stewart    | M      | joshua.stewart@yahoo | 5/18/1970  |   47 | 184896 |      1 |
| 904898 | Ann         | Cooper     | F      | ann.cooper@exxonmobi | 12/25/1992 |   25 | 182521 |      2 |
| 883936 | Douglas     | Flores     | M      | douglas.flores@gmail | 4/21/1986  |   31 | 181793 |      2 |
```

Counting Rows

To count the number of rows in a table use the following statement



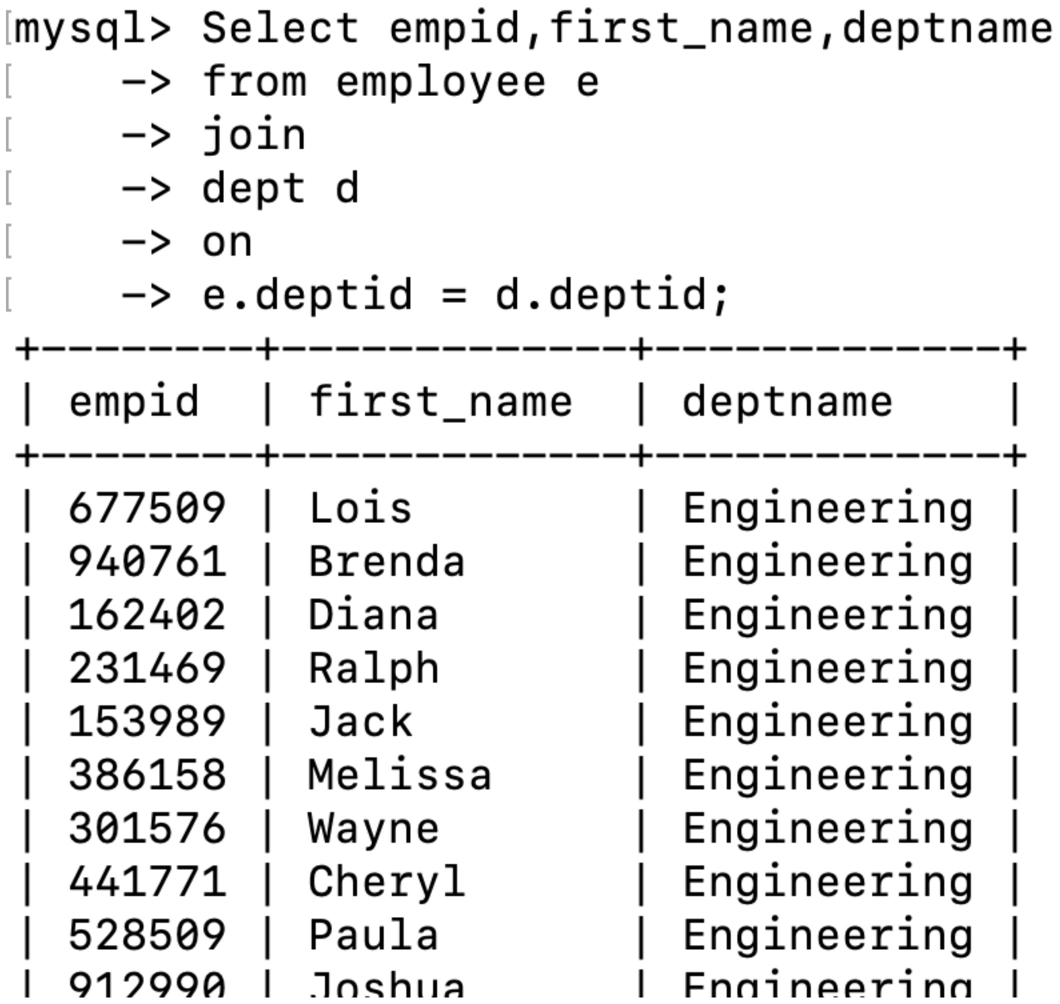
```
[mysql> Select count(*) from employee;
+----------+
| count(*) |
+----------+
|      100 |
+----------+
1 row in set (0.01 sec)
```

Using More Than one Table

If you want to track which employee belongs to which **depa**rtment and the details of the department you can use a separate table called dept and store all the department related details there. You can retrieve the data from both the tables by using a JOIN statement.

```
create table dept(deptid int primary key, deptname varchar(20));

insert into dept values(1,"Engineering");
insert into dept values(2,"Testing");
insert into dept values(3,"Production");
insert into dept values(4,"HR");
```

Display the dept name of all the employees



```
[mysql> Select empid,first_name,deptname
[    -> from employee e
[    -> join
[    -> dept d
[    -> on
[    -> e.deptid = d.deptid;
+--------+-------------+-------------+
| empid  | first_name  | deptname    |
+--------+-------------+-------------+
| 677509 | Lois        | Engineering |
| 940761 | Brenda      | Engineering |
| 162402 | Diana       | Engineering |
| 231469 | Ralph       | Engineering |
| 153989 | Jack        | Engineering |
| 386158 | Melissa     | Engineering |
| 301576 | Wayne       | Engineering |
| 441771 | Cheryl      | Engineering |
| 528509 | Paula       | Engineering |
| 912990 | Joshua      | Engineering |
```

The MySQL UPDATE Statement

The `UPDATE` statement is used to modify the existing records in a table.

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Example: If you want to update the employee table with deptid=2 for empid=1.
Note: If where condition is not specified, all the records will be updated

```
UPDATE employee
SET deptid = 2
WHERE empid=1;
```

MySQL DELETE Statement

The `DELETE` statement is used to delete existing records in a table.

```
DELETE FROM table_name WHERE condition;
```

Example: If you want to delete the employee table record with empid=1.
Note: If where condition is not specified, all the records will be deleted

```
DELETE FROM employee WHERE empid=1;
```

The MySQL LIMIT Clause

The `LIMIT` clause is used to specify the number of records to return.

The `LIMIT` clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

LIMIT Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
LIMIT number;
Select empid,name from employee limit 2;
```

MySQL MIN() and MAX() Functions

The `MIN()` function returns the smallest value of the selected column.

The `MAX()` function returns the largest value of the selected column.

MIN() Syntax

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

MAX() Syntax

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

MySQL COUNT(), AVG() and SUM() Functions

The `COUNT()` function returns the number of rows that matches a specified criterion.

COUNT() Syntax

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

The `AVG()` function returns the average value of a numeric column.

AVG() Syntax

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

The `SUM()` function returns the total sum of a numeric column.

SUM() Syntax

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

groupBy() Syntax

The GROUP BY Statement in SQL is used to arrange identical data into groups with the help of some functions. i.e if a particular column has same values in different rows then it will arrange these rows in a group.

Important Points:

GROUP BY clause is used with the SELECT statement.

In the query, GROUP BY clause is placed after the WHERE clause.

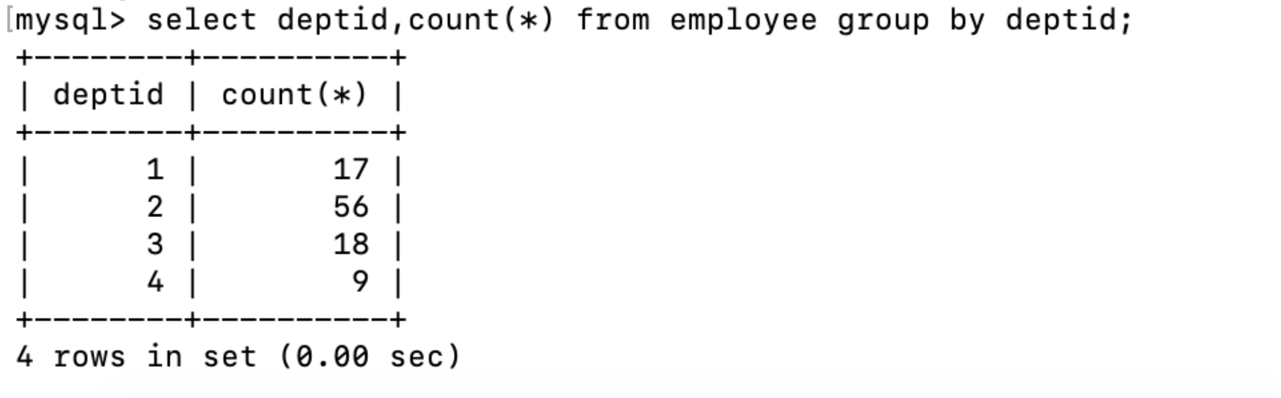
In the query, GROUP BY clause is placed before ORDER BY clause if used any.

Syntax:

```
SELECT column1, function_name(column2)

FROM table_name

WHERE condition

GROUP BY column1, column2

ORDER BY column1, column2;
```

```
function_name: Name of the function used for example, SUM() , AVG().
table_name: Name of the table.
condition: Condition used.
```

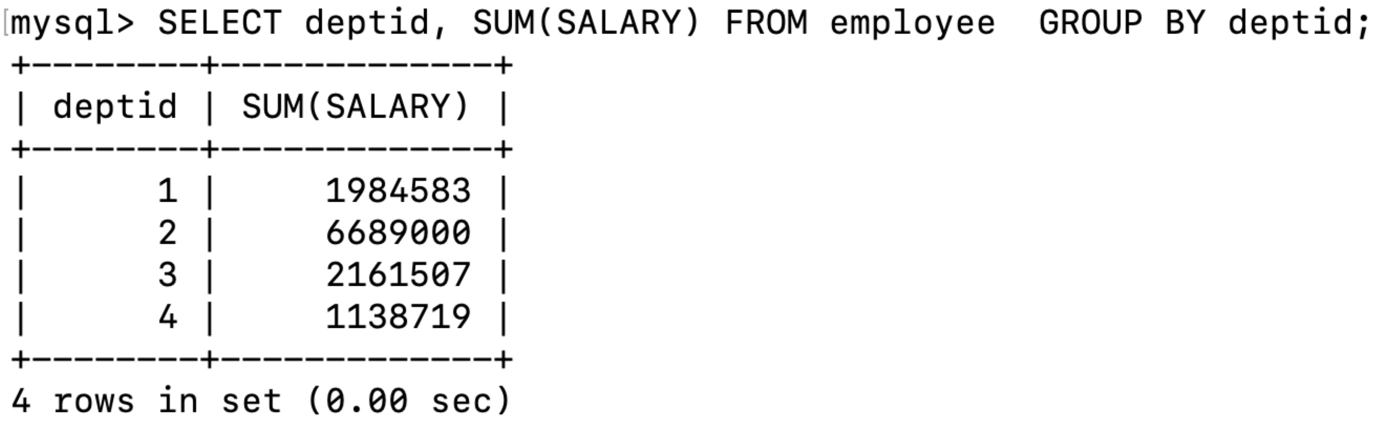
Example: You can group all the records based on the department like below:



```
mysql> select deptid,count(*) from employee group by deptid;
+--------+----------+
| deptid | count(*) |
+--------+----------+
|      1 |       17 |
|      2 |       56 |
|      3 |       18 |
|      4 |        9 |
+--------+----------+
4 rows in set (0.00 sec)
```

The above query counts the no of employees in each department.

Find the sum of salary by each department

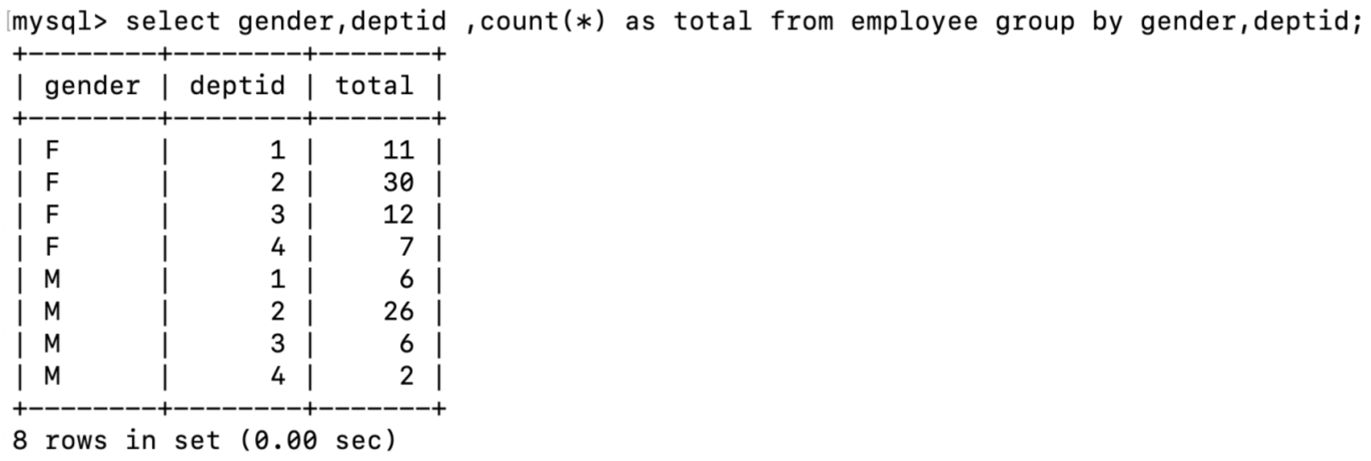


```
mysql> SELECT deptid, SUM(SALARY) FROM employee  GROUP BY deptid;
+--------+-------------+
| deptid | SUM(SALARY) |
+--------+-------------+
|      1 |     1984583 |
|      2 |     6689000 |
|      3 |     2161507 |
|      4 |     1138719 |
+--------+-------------+
4 rows in set (0.00 sec)
```

Group By multiple columns: Group by multiple column is say for example, **GROUP BY column1, column2**. This means to place all the rows with same values of both the columns **column1** and **column2** in one group. Consider the below query:

Example:

Find out how many Females and Males are there in each department?



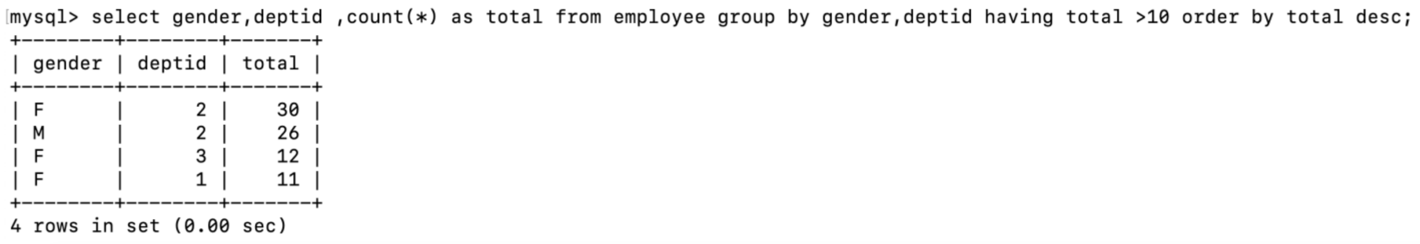
```
mysql> select gender,deptid ,count(*) as total from employee group by gender,deptid;
+--------+--------+-------+
| gender | deptid | total |
+--------+--------+-------+
| F      |      1 |    11 |
| F      |      2 |    30 |
| F      |      3 |    12 |
| F      |      4 |     7 |
| M      |      1 |     6 |
| M      |      2 |    26 |
| M      |      3 |     6 |
| M      |      4 |     2 |
+--------+--------+-------+
8 rows in set (0.00 sec)
```

We know that WHERE clause is used to place conditions on columns but what if we want to place conditions on groups?
This is where HAVING clause comes into use. We can use HAVING clause to place conditions to decide which group will be the part of final result-set. Also we can not use the aggregate functions like SUM(), COUNT() etc. with WHERE clause. So we have to use HAVING clause if we want to use any of these functions in the conditions.

Syntax:
```
SELECT column1, function_name(column2)

FROM table_name

WHERE condition

GROUP BY column1, column2

HAVING condition

ORDER BY column1, column2;
```

function_name: Name of the function used for example, SUM() , AVG().
table_name: Name of the table.
condition: Condition used.

Example: In the above query, if we want to see the total only if the number is > 10 ?

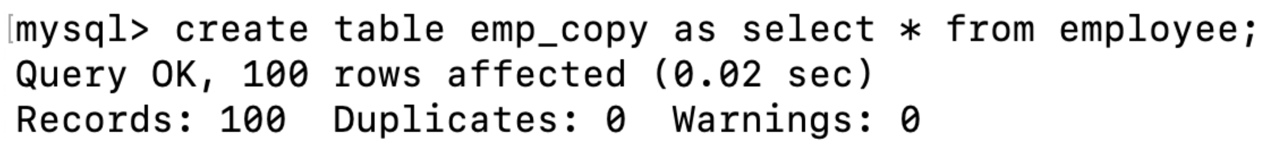


```
mysql> select gender,deptid ,count(*) as total from employee group by gender,deptid having total >10 order by total desc;
+--------+--------+-------+
| gender | deptid | total |
+--------+--------+-------+
| F      |      2 |    30 |
| M      |      2 |    26 |
| F      |      3 |    12 |
| F      |      1 |    11 |
+--------+--------+-------+
4 rows in set (0.00 sec)
```

Clone a table structure
Create table <New_Table> like <Original table>

CREATE TABLE AS SELECT(CTAS)

If you want to create a new table with a query from an existing table, use CTAS.



```
mysql> create table emp_copy as select * from employee;
Query OK, 100 rows affected (0.02 sec)
Records: 100  Duplicates: 0  Warnings: 0
```

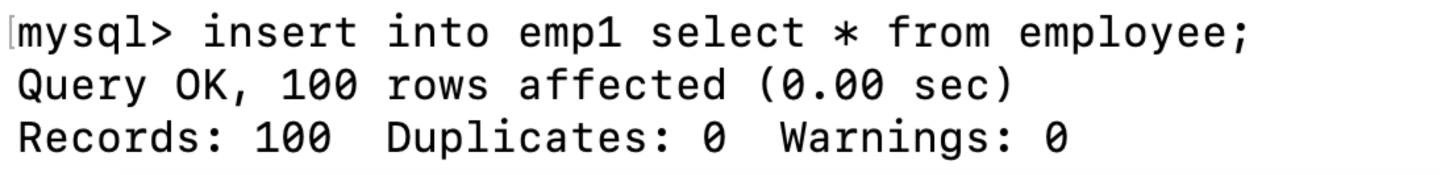
The new table will be an exact copy of the original table with same schema and data. The new table schema will depend on what columns you select from the original table. The new table data depends on the output of the select query.
Note: If a table with a same name already exists, mysql will throw an error.

INSERT INTO AS SELECT

If you want to copy data from an original table in to a new table whose schema is already created, use the Insert Into … select clause

```
mysql> CREATE TABLE emp1 (
       empid int(11) DEFAULT NULL,
       first_name varchar(20) DEFAULT NULL,
       last_name varchar(20) DEFAULT NULL,
       gender varchar(1) DEFAULT NULL,
       email varchar(20) DEFAULT NULL,
       dob varchar(12) DEFAULT NULL,
       age int(11) DEFAULT NULL,
       salary double DEFAULT NULL,
       deptid int(11) DEFAULT NULL
     );
```

Example:



```
mysql> insert into emp1 select * from employee;
Query OK, 100 rows affected (0.00 sec)
Records: 100  Duplicates: 0  Warnings: 0
```

Note: If a table with a same name doesn't exist, mysql will throw an error.

The MySQL DROP TABLE Statement

The DROP TABLE statement is used to drop an existing table in a database.

Syntax

```
DROP TABLE table_name;
```

MySQL TRUNCATE TABLE

The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

Syntax

```
TRUNCATE TABLE table_name;
```

MySQL ALTER TABLE Statement

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

ALTER TABLE - ADD Column

To add a column in a table, use the following syntax:

```
ALTER TABLE table_name
ADD column_name datatype;
```

The following SQL adds an "Email" column to the "Customers" table:

Example

```
ALTER TABLE Customers
ADD Email varchar(255);
```

ALTER TABLE - DROP COLUMN

To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

The following SQL deletes the "Email" column from the "Customers" table:

Example

```
ALTER TABLE Customers
DROP COLUMN Email;
```

ALTER TABLE - MODIFY COLUMN type

To change the data type of a column in a table, use the following syntax:

```
ALTER TABLE table_name
MODIFY COLUMN column_name datatype;
```

MySQL CREATE VIEW Statement

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.

A view is created with the `CREATE VIEW` statement.

CREATE VIEW Syntax

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Note: A view always shows up-to-date data! The database engine recreates the view, every time a user queries it.

```
Create view highlypaidemps as select * from employee where salary > 100000;

Select * from highlypaidemps;
```

SQL commands are divided into four subgroups, DDL, DML, DCL, and TCL.

DDL

DDL is short name of **Data Definition Language,** which deals with database schemas and descriptions, of how the data should reside in the database.

- CREATE to create a database and its objects like (table, index, views, store procedure, function, and triggers)
- ALTER - alters the structure of the existing database
- DROP - delete objects from the database
- TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed
- COMMENT - add comments to the data dictionary
- RENAME - rename an object

DML

DML is short name of **Data Manipulation Language** which deals with data manipulation and includes most common SQL statements such SELECT, INSERT, UPDATE, DELETE, etc., and it is used to store, modify, retrieve, delete and update data in a database.

- SELECT - retrieve data from a database
- INSERT- insert data into a table
- UPDATE - updates existing data within a table
- DELETE - Delete all records from a database table
- MERGE - UPSERT operation (insert or update)
- CALL - call a PL/SQL or Java subprogram
- EXPLAIN PLAN - interpretation of the data access path
- LOCK TABLE - concurrency Control

DCL

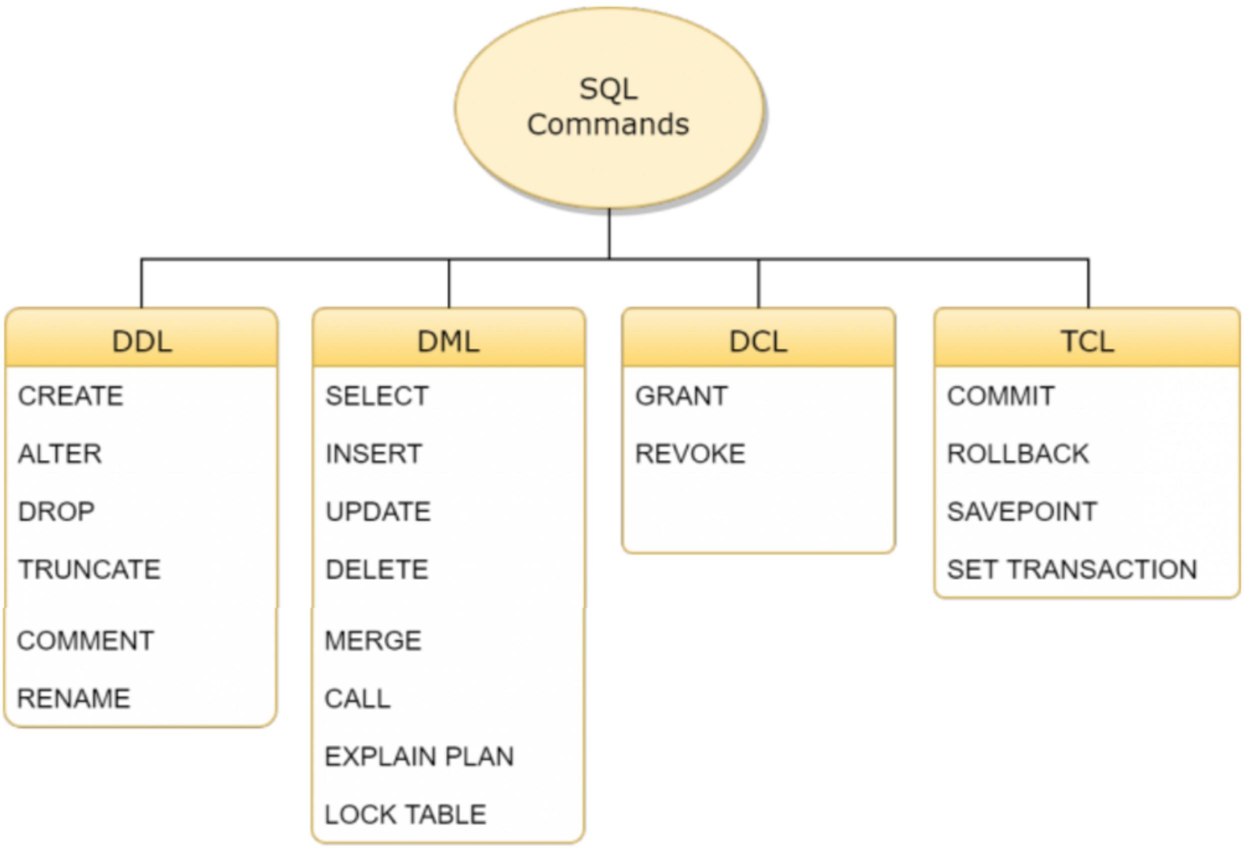
DCL is short name of **Data Control Language** which includes commands such as GRANT and mostly concerned with rights, permissions and other controls of the database system.

- GRANT - allow users access privileges to the database
- REVOKE - withdraw users access privileges given by using the GRANT command

TCL

TCL is short name of Transaction Control Language which deals with a transaction within a database.

- COMMIT - commits a Transaction
- ROLLBACK - rollback a transaction in case of any error occurs
- SAVEPOINT - to rollback the transaction making points within groups
- SET TRANSACTION - specify characteristics of the transaction



Exercises

1. Write a SQL statement to create a simple table, countries including columns country_id (Primary key)(Number),country_name(text) and region_id(Text).

2. Write insert statements to insert data in to the above table with some sample data.
   1,India,Asia
   2,South Africa, Africa
   3,USA, America
   4, Newzealand, Australia
   5, Bangladesh,Asia
   6, Pakistan,Asia

3. Write an SQL statement to create a duplicate dup_countries table, with the structure of table countries.
4. Write an SQL statement to create a table by name asia_countries which should include only Asian countries and schema similar to countries table.

5. Write a query to display the names (First Name, Last Name) using alias name “First_Name", "Last_Name" of the employees table

6. Write a query to get the number of employees working in the company

7. Write a query to display the Full Name as a column (First Name+Last Name) (Use employees table)

8. Write a query to get unique department ID from employee table

9. Write a query to get all employee details from the employee table order by first name, descending

10. Write a query to get the names (first_name, last_name), salary, PF of all the employees (PF is calculated as 15% of salary)
11. Write a query to get the maximum salary in each department.

12. Write a query to get the total salaries payable to employees in each department

13. Write a query to get the Average, maximum and minimum salary from employees table

14. Write a query to get the average salary and number of employees in each department.

15. Write a query to get the number different continents in the countries table.
16. Write a query get all first name from employees table in upper case.

17. Write a query to select first 10 records from a table.

18. Create a new dept table with columns deptid(number, PK), name(text) and enter some sample data. Use deptids as 1,2,3,4 and department names as Engineering,QA,Production,HR

19. Create a query which return the employee first name, salary and the name of the department that the employee belongs to.

Create a new database called salesdb and create a table product as shown below.

productID INT(PK)	productCode CHAR(3)	name VARCHAR(30)	quantity INT	price DECIMAL(10,2)
1001	PEN	Pen Red	5000	1.23
1002	PEN	Pen Blue	8000	1.25
1003	PEN	Pen Black	2000	1.25
1004	PEC	Pencil 2B	10000	0.48
1005	PEC	Pencil 2H	8000	0.49

Create a table suppliers as shown below in salesdb

Table: suppliers		
supplierID INT(PK)	**name VARCHAR(3)**	**phone CHAR(8)**
501	ABC Traders	88881111
502	XYZ Company	88882222
503	QQ Corp	88883333

add a new column supplierID(FK) into the products table.
Update the product table by setting the supplierid to values as 501,501,502,503,502 respectively

Write a query to list the product name, price and the supplier name from the above tables

Write a query to list the number of PEN and PEC available in the products table

Database Integration with Python

Install mysql.connector

To connect the python application with the MySQL database, we must import the mysql.connector module in the program.

The mysql.connector is not a built-in module that comes with the python installation. We need to install it to get it working.

```
pip3 install mysql-connector
```

Database Connection

Following are the steps to connect the python application to the database.

1. Import mysql.connector module
2. Create the connection object.
3. Create the cursor object
4. Execute the query

Creating the connection

To create a connection between the MySQL database and the python application, the connect() method of mysql.connector module is used.

Pass the database details like HostName, username, and the database password in the method call. The method returns the connection object.

The syntax to use the connect() is given below.

```
conn= mysql.connector.connect(host = <host-
name> , user = <username> , passwd = <password> )
```

Example: Creating a database connection in Python

```
import mysql.connector

#Create the connection object
myconn = mysql.connector.connect(host = "localhost", user = "newuser",passwd = "password",data
base="bigdata")

#printing the connection object
print(myconn)
```

Here, we must notice that we can specify the database name in the connect() method if we want to connect to a specific database.

Creating a cursor object

The cursor object can be defined as an abstraction specified in the Python DB-API 2.0. It facilitates us to have multiple separate working environments through the same connection to the database. We can create the cursor object by calling the 'cursor' function of the connection object. The cursor object is an important aspect of executing queries to the databases.

The syntax to create the cursor object is given below.

```
<my_cur>  = myconn.cursor()
```

```
import mysql.connector
#Create the connection object
myconn = mysql.connector.connect(host = "172.31.10.241", user = "newuser",passwd = "passwo
rd", database = "bigdata")

#printing the connection object
print(myconn)

#creating the cursor object
cur = myconn.cursor()

print(cur)
```

```
try:
    cur.execute("create databases salesdb")
    cur.execute("show databases")
except:
   myconn.rollback()
for x in cur:
   print(x)
myconn.close()
```

Creating a table using python in mysql

```
import mysql.connector
#Create the connection object
myconn = mysql.connector.connect(host = "172.31.10.241", user = "newuser",passwd = "password",
database = "bigdata")

#printing the connection object
print(myconn)

#creating the cursor object
cur = myconn.cursor()

try:

cur.execute("create table orders(oid int  primary key,
 product_name varchar(20), cust_id int)")
except:
  myconn.rollback()
  myconn.close()
```

Insert Data using python

```
import mysql.connector
#Create the connection object
myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "", database = "salesdb")
#creating the cursor object
cur = myconn.cursor()
sql = "insert into orders(oid, product_name, cust_id) values (%s, %s, %s)"
```

```
#The row values are provided in the form of tuple
val = [(1, "Mobile",100),( 2, "TV",101)]

try:
    #inserting the values into the table
    cur.executemany(sql,val)

    #commit the transaction
    myconn.commit()

except:
    myconn.rollback()

print(cur.rowcount,"record inserted!")
myconn.close()
```

Reading Data using python

```
import mysql.connector
#Create the connection object
myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "", database = "salesdb")
#creating the cursor object
cur = myconn.cursor()
try:
    #Reading the Employee data
    cur.execute("select * from orders")

    #fetching the rows from the cursor object
    result = cur.fetchall()
    #printing the result

    for x in result:
        print(x);
```

```
except:
    myconn.rollback()
    myconn.close()
```

The fetchone() method

The fetchone() method is used to fetch only one row from the table. The fetchone() method returns the next row of the result-set.

```
     #fetching the first row from the cursor object
       result = cur.fetchone()

       #printing the result
       print(result)

    except:
       myconn.rollback()

    myconn.close()
```