CHAPTER 5

Database Queries

5.1 Structured Query Language (SQL)

SQL, sometimes pronounced as *See-Quel,* is a Standard Database Language which is used to deal with RDBMS to perform various operations on the data that exist in the tables, such as:

(a) Creation of Database

(b) Creation of Tables

(c) Inserting records in tables

(d) Updating (modifying) records in tables

(e) Deleting records from tables

(f) Altering (modifying) table structure (Rename table and field name; adding and deleting columns)

(g) Drop (delete) database and tables

All Relational databases such as Oracle, MySQL, MS SQL Server, PostgreSQL, DB2, Sybase, etc uses SQL.

A query is an operation that retrieves data from one or more tables.

SQL statements start with specific keywords like Create, Insert, Update, and Delete etc. based on the operation you want to perform and end with a semicolon (;). All such rules and guidelines to define an SQL statement are called standard Syntax. For example, below is a Query which will select all records from students table, where "SELECT * FROM" is the SQL standard syntax and Students is the table you have created to store student's information.

```
SELECT * FROM Students;
```

Various Syntax in SQL:

Create

ORACLE

CREATE DATABASE statement is used to create a database. Most of the time it's created during database installation, and post-installation you can just start using it. Otherwise, you must have to create a database after installation following some additional advance actions before you have an operational database. It's an advanced option, hence I am not going to explain more on it in this beginner's edition.

CREATE TABLE statement is used to create tables. Creating a table involves naming the table and defining its columns and data type. Integrity constraints like primary key, unique key, foreign key etc. can be defined for the columns while creating the table or you can define them later as well.

Syntax

Without Primary Key

```
CREATE TABLE table_name
(
column 1 datatype,
column 2 datatype,
column 3 datatype,
....
....
column n datatype
);
```

With Primary Key

```
CREATE TABLE table_name
(
column 1 datatype,
column 2 datatype,
column 3 datatype,
....
....
column n datatype,
CONSTRAINT constraint-name PRIMARY KEY (field-name)
);
```

Examples

Let us create a few tables. We will use the same tables for demonstrating the rest of the SQL statements.

I have created four tables for demonstration purpose. STUDENTS, SUBJECTS, FACULTY, and MARKS.

- **STUDENTS** is a Master table, which contains student's static data
- **SUBJECTS** is a Master table, which contains all subject details running in university along with their subject id.
- **FACULTY** is a Master table which contains faculty static data.
- **MARKS** table is a dynamic table, which contains all students' marks for all semesters along with corresponding faculty and few more details.

Here,

- Enrollment_No field is the Primary Key in **STUDENTS** table.
- **Subject_id** field is the Primary Key in **SUBJECTS** table.

- **Faculty_id** field is the Primary Key in **FACULTY** table.
- In Marks table, the **subject_id** field is the *Foreign Key,* referencing **subject_id** *Primary Key* of **SUBJECTS** table, and **faculty_id** field is another *Foreign Key* referencing **faculty_id** Primary Key of **FACULTY** table.

Note: As explained in Normalization Rules, Master table contains static data (with no duplicity), which don't change or has very less probability of change. For example, Student Name, Father Name, Contact No, Address, Date of Birth etc. The dynamic table contains data requires frequent manipulations. For example, Student Marks, Semester, Fee Submission Date, Fee Due Amount etc. Hence, Primary Key(s) mostly defined on Master table and Foreign Key(s) defined on the Dynamic table.

TABLE: STUDENTS

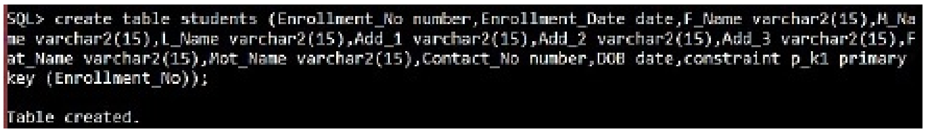
```
          Create table students
(
Enrollment_No number,
Enrollment_Date date,
F_Name varchar2(15),
M_Name varchar2(15)
,L_Name varchar2(15),
Add_1 varchar2(15)
,Add_2 varchar2(15),
Add_3 varchar2(15),
Fat_Name varchar2(15),
Mot_Name varchar2(15),
Contact_No number,
DOB date,
constraint p_k1 primary key (Enrollment_No)
          );
```

Where,

o **Enrollment_No**	: Enrollment number of student
o **Enrollment_Date**	: Enrollment date of student in College/University
o **F_Name**	: First Name of student
o **M_Name**	: Middle name of student
o **L_Name**	: Last Name of student
o **Add_1**	: Address part 1 of student
o **Add_2**	: Address part 2 of student
o **Add_3**	: Address part 3 of student
o **Fat_Name**	: Father's Name of student
o **Mot_Name**	: Mother's name of student
o **Contact_No**	: Contact no of student

- **DOB** : Date of Birth of student

And **Enrollment_No** is defined as the constraint *(Primary Key)* with constraint name **P k1**.



```
SQL> create table students (Enrollment_No number,Enrollment_Date date,F_Name varchar2(15),M_Na
me varchar2(15),L_Name varchar2(15),Add_1 varchar2(15),Add_2 varchar2(15),Add_3 varchar2(15),F
at_Name varchar2(15),Mot_Name varchar2(15),Contact_No number,DOB date,constraint p_k1 primary
key (Enrollment_No));

Table created.
```

TABLE: SUBJECTS

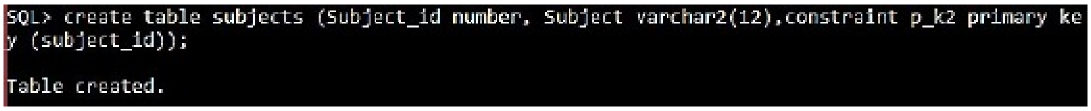
```
Create table subjects
(
Subject_id number,
Subject varchar2(12),
constraint p_k2 primary key (subject_id)
);
```

Where

Subject_id : Unique subject id allocated to each subject

Subject : Name of subject

And **subject_id** is defined as a constraint *(Primary Key)* with the name as **p_k2**.



```
SQL> create table subjects (Subject_id number, Subject varchar2(12),constraint p_k2 primary ke
y (subject_id));

Table created.
```

TABLE: FACULTY

```
Create table faculty
(
Faculty_id number,
F_Name varchar2(20),
M_Name varchar2(20),
L_Name varchar2(20),
DOJ date,
constraint p_k3 primary key (faculty_id)
);
```

Where

Faculty_id: Unique ID allocated to each faculty member just like enrollment number of student

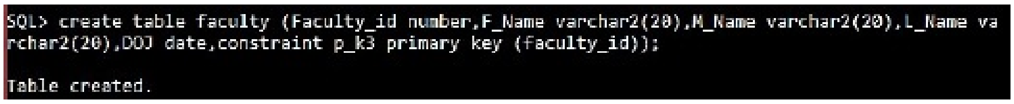
F_Name : First name of the faculty

M_Name : Middle name of faculty

L_Name : Last name of the faculty

DOJ : Data of Joining of faculty

And **faculty_id** is defined as the constraint *(Primary Key)* with constraint name **P k3**.



```
SQL> create table faculty (Faculty_id number,F_Name varchar2(20),M_Name varchar2(20),L_Name va
rchar2(20),DOJ date,constraint p_k3 primary key (faculty_id));

Table created.
```

TABLE: MARKS

```
Create table marks (
Enrollment_No number,
Subject_id number,
Semester varchar2(20),
Faculty_id number,
Marks number,
constraint p_k4 foreign key (subject_id) references
subjects(subject_id),
constraint p_k5 foreign key (faculty_id) references
faculty(faculty_id)
);
```

Where,

Enrollment_No : Enrollment Number of student

Subject_id : Subject id of the subject on which student enrolled (can be multiple entries for a student for multiple subjects)

Semester : Semester of Student (old as well as new this table will contain dynamic data, as soon as a student will promote in next semester, new semester data will be populated)

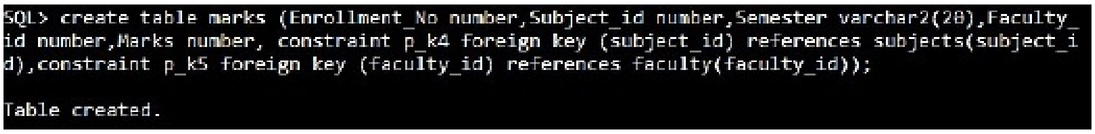
Faculty_id : Unique id of faculty

Marks : Marks obtained by the student in a particular subject (corresponding to subject id)

And,

(a) **subject_id** is defined as the constraint *(Foreign Key)* referencing **subject_id** *(Primary Key)* of Subjects table, with constraint name **P_k4**.

(b) **faculty_id** is another constraint *(Foreign Key)* referencing **faculty_id** *(Primary Key)* of Faculty table, with constraint name **P_k5**.



```
SQL> create table marks (Enrollment_No number,Subject_id number,Semester varchar2(20),Faculty_
id number,Marks number, constraint p_k4 foreign key (subject_id) references subjects(subject_i
d),constraint p_k5 foreign key (faculty_id) references faculty(faculty_id));

Table created.
```

MYSQL

Corresponding Mysql syntax and commands are as follows:

Unlike Oracle, in MySQL first you have to create a database using **CREATE** command and then you have to switch to that database to start working.

Create and Switch Database:

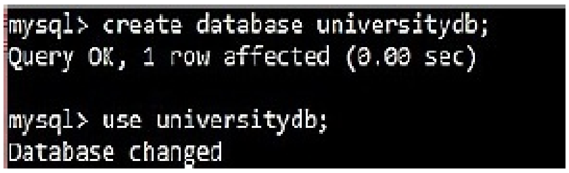
Syntax

```
CREATE DATABASE database_name;
Use database_name;
```

Example

```
CREATE DATABASE universitydb;
USE universitydb;
```

Above the first command will create a database with name universitydb, and second will switch to that database.



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

mysql> use universitydb;
Database changed
```

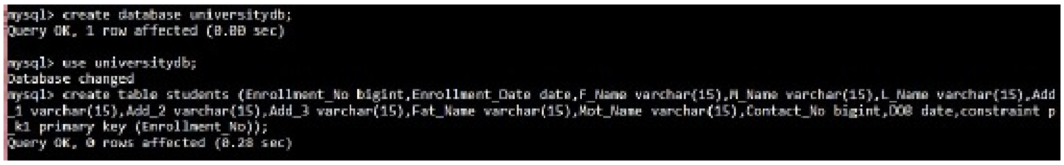
CREATE Syntax

It is same as Oracle. Only make sure to specify data types according to MySQL.

Examples

To create Students table:

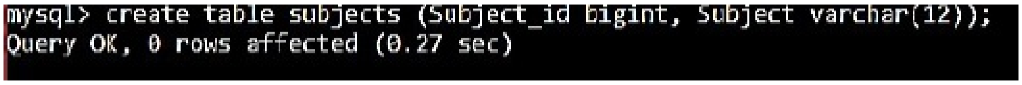
```
CREATE TABLE students
(
Enrollment_No bigint,
Enrollment_Date date,
F_Name varchar(15),
M_Name varchar(15),
L_Name varchar(15),
Add_1 varchar(15),
Add_2 varchar(15),
Add_3 varchar(15)
,Fat_Name varchar(15),
Mot_Name varchar(15),
Contact_No bigint,
DOB date,
constraint p_k1 primary key (Enrollment_No)
);
```



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

mysql> use universitydb;
Database changed
mysql> create table students (Enrollment_No bigint,Enrollment_Date date,F_Name varchar(15),M_Name varchar(15),L_Name varchar(15),Add
_1 varchar(15),Add_2 varchar(15),Add_3 varchar(15),Fat_Name varchar(15),Mot_Name varchar(15),Contact_No bigint,DOB date,constraint p
_k1 primary key (Enrollment_No));
Query OK, 0 rows affected (0.28 sec)
```

To create Subjects table:

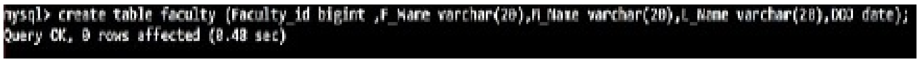
```
CREATE TABLE subjects
(
Subject_id bigint,
Subject varchar(12)
);
```



```
mysql> create table subjects (Subject_id bigint, Subject varchar(12));
Query OK, 0 rows affected (0.27 sec)
```

To create faculty table:

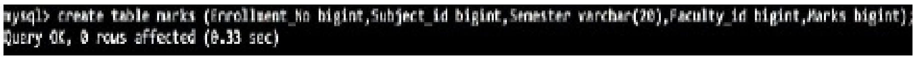
```
CREATE TABLE faculty
(
Faculty_id bigint ,
F_Name varchar(20),
M_Name varchar(20),
L_Name varchar(20),
DOJ date
);
```



```
mysql> create table faculty (Faculty_id bigint ,F_Name varchar(20),M_Name varchar(20),L_Name varchar(20),DOJ date);
Query OK, 0 rows affected (0.48 sec)
```

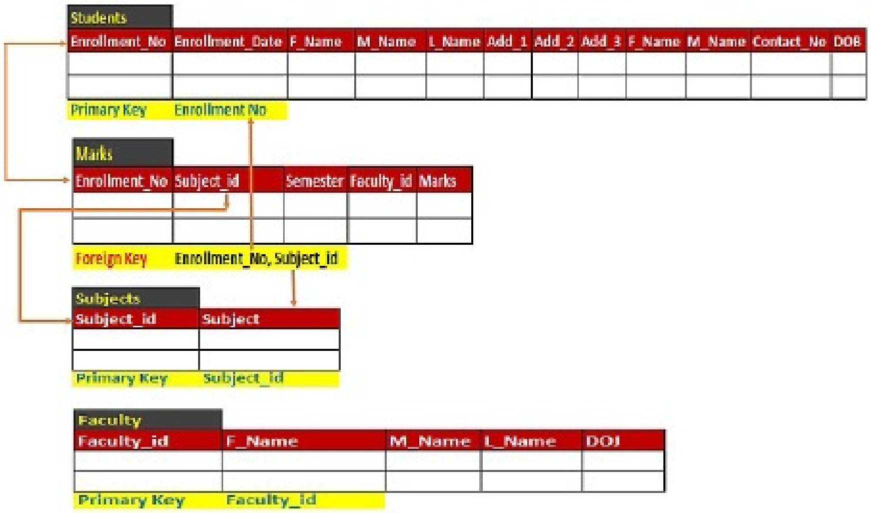
To create Marks table:

```
CREATE TABLE marks
(
Enrollment_No bigint,
Subject_id bigint,
Semester varchar(20),
Faculty_id bigint, Marks bigint
);
```



```
mysql> create table marks (Enrollment_No bigint,Subject_id bigint,Semester varchar(20),Faculty_id bigint,Marks bigint);
Query OK, 0 rows affected (0.33 sec)
```

Please refer the following figure to understand the overall relationship.



Defined Relationship Between Tables

DESC or DESCRIBE

ORACLE

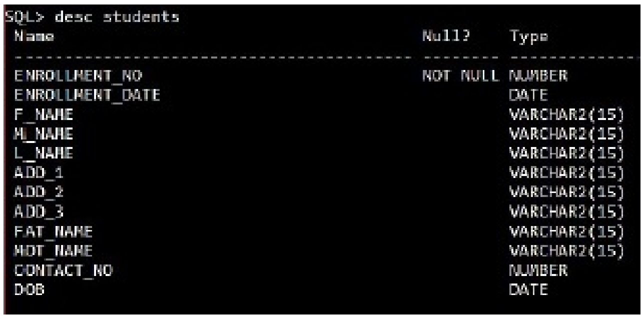
DESC or **DESCRIBE** is used to describe the structure of a table in the database.

Syntax

```
DESC table_name or DESCRIBE table_name
```

Example

```
DESC Students;
```



```
SQL> desc students
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 ENROLLMENT_NO                             NOT NULL NUMBER
 ENROLLMENT_DATE                                    DATE
 F_NAME                                             VARCHAR2(15)
 M_NAME                                             VARCHAR2(15)
 L_NAME                                             VARCHAR2(15)
 ADD_1                                              VARCHAR2(15)
 ADD_2                                              VARCHAR2(15)
 ADD_3                                              VARCHAR2(15)
 FAT_NAME                                           VARCHAR2(15)
 MOT_NAME                                           VARCHAR2(15)
 CONTACT_NO                                         NUMBER
 DOB                                                DATE
```

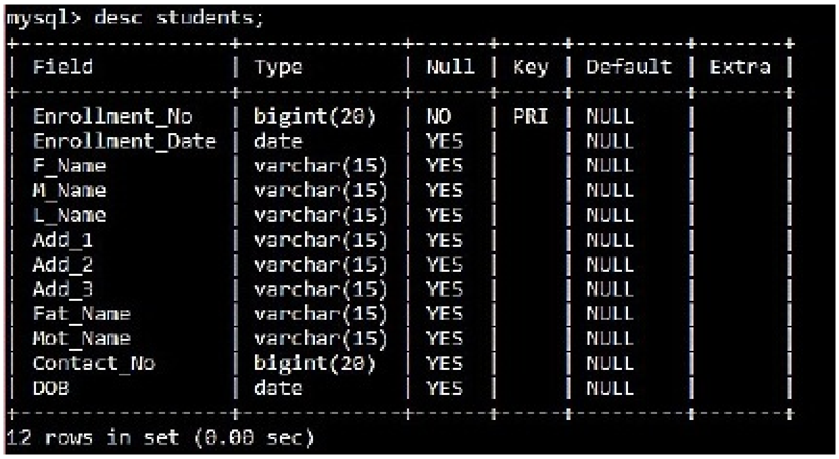
MYSQL

Syntax

Same as Oracle.

Examples

DESC students;



```
mysql> desc students;
+-----------------+-------------+------+-----+---------+-------+
| Field           | Type        | Null | Key | Default | Extra |
+-----------------+-------------+------+-----+---------+-------+
| Enrollment_No   | bigint(20)  | NO   | PRI | NULL    |       |
| Enrollment_Date | date        | YES  |     | NULL    |       |
| F_Name          | varchar(15) | YES  |     | NULL    |       |
| M_Name          | varchar(15) | YES  |     | NULL    |       |
| L_Name          | varchar(15) | YES  |     | NULL    |       |
| Add_1           | varchar(15) | YES  |     | NULL    |       |
| Add_2           | varchar(15) | YES  |     | NULL    |       |
| Add_3           | varchar(15) | YES  |     | NULL    |       |
| Fat_Name        | varchar(15) | YES  |     | NULL    |       |
| Mot_Name        | varchar(15) | YES  |     | NULL    |       |
| Contact_No      | bigint(20)  | YES  |     | NULL    |       |
| DOB             | date        | YES  |     | NULL    |       |
+-----------------+-------------+------+-----+---------+-------+
12 rows in set (0.00 sec)
```

INSERT

ORACLE

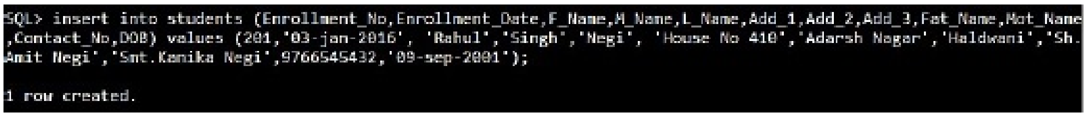
Insert is used to insert single or multiple values/records/data in a table.

Syntax

```
INSERT INTO table_name
(columnl, column2, ... column_n)
VALUES
(value1, value2, ... value_n );
```

Example

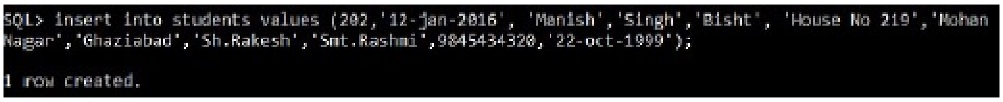
```
INSERT INTO students
(
Enrollment_No,Enrollment_Date,F_Name,M_Name,L_
Name,Add_1,Add_2,Add_3,Fat_Name,Mot_Name,Contact_No,DOB
)
VALUES
(
201,'03-jan-2016',    'Rahul','Singh','Negi',    'House    No    410','Adarsh
Nagar','Haldwani','Sh.Amit Negi','Smt.Kanika Negi',9766545432,'09-sep-2001'
);
```



```
SQL> insert into students (Enrollment_No,Enrollment_Date,F_Name,M_Name,L_Name,Add_1,Add_2,Add_3,Fat_Name,Mot_Name
,Contact_No,DOB) values (201,'03-jan-2016', 'Rahul','Singh','Negi', 'House No 410','Adarsh Nagar','Haldwani','Sh.
Amit Negi','Smt.Kanika Negi',9766545432,'09-sep-2001');

1 row created.
```

If you are inserting all column values, then it's not necessary to supply all column names.

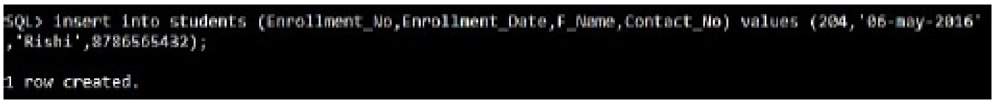
```
INSERT INTO students VALUES
(
202,'12-jan-2016',    'Manish','Singh','Bisht',    'House    No    219','Mohan
Nagar','Ghaziabad','Sh.Rakesh','Smt. Rashmi',9845434320,'22-oct-1999'
);
```



```
SQL> insert into students values (202,'12-jan-2016', 'Manish','Singh','Bisht', 'House No 219','Mohan
Nagar','Ghaziabad','Sh.Rakesh','Smt.Rashmi',9845434320,'22-oct-1999');

1 row created.
```

If you are inserting only a few column values, then it's mandatory to specify column names in the insert statement.

```
INSERT INTO students
(
Enrollment_No,Enrollment_Date,F_Name,Contact_No)     VALUES     (204,'06-may-
2016','Rishi',8786565432
);
```



```
SQL> insert into students (Enrollment_No,Enrollment_Date,F_Name,Contact_No) values (204,'06-may-2016'
,'Rishi',8786565432);

1 row created.
```

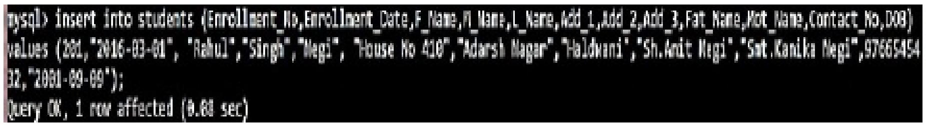
MYSQL

Syntax

Same as Oracle.

Examples

```
INSERT INTO students (
Enrollment_No,Enrollment_Date,F_Name,M_Name,L_
Name,Add_1,Add_2,Add_3,Fat_Name,Mot_Name,Contact_No,DOB
)
VALUES  (201,"2016-03-01",  "Rahul","Singh","Negi",  "House  No  410","Adarsh
Nagar","Haldwani","Sh.Amit Negi","Smt.Kanika Negi",9766545432,"2001-09-09"
);
```



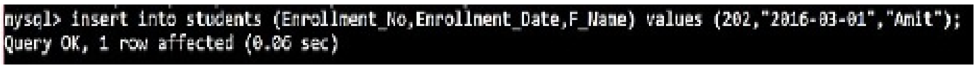
```
mysql> insert into students (Enrollment_No,Enrollment_Date,F_Name,M_Name,L_Name,Add_1,Add_2,Add_3,Fat_Name,Mot_Name,Contact_No,DOB)
values (201,"2016-03-01", "Rahul","Singh","Negi", "House No 410","Adarsh Nagar","Haldwani","Sh.Amit Negi","Smt.Kanika Negi",97665454
32,"2001-09-09");
Query OK, 1 row affected (0.08 sec)
```

Like Oracle, if you are inserting all field values, then you can exclude field names:

```
INSERT INTO students VALUES
(
202,'12-jan-2016',    'Manish','Singh','Bisht',    'House    No    219','Mohan
Nagar','Ghaziabad','Sh.Rakesh','Smt. Rashmi',9845434320,'22-oct-1999'
);
```

To insert few fields:

```
INSERT INTO students
(
Enrollment_No,Enrollment_Date,F_Name,Contact_No
)
VALUES (204,'06-may-2016','Rishi',8786565432
);
```



```
mysql> insert into students (Enrollment_No,Enrollment_Date,F_Name) values (202,"2016-03-01","Amit");
Query OK, 1 row affected (0.06 sec)
```

COMMIT

It is a transactional command used to save changes invoked by a transaction to the database.

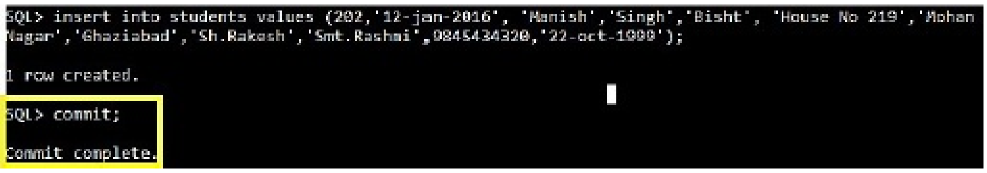
ORACLE

Syntax

Commit;

Example

For example, refer below figure where commit is executed after inserting a record in a database table. If you will not apply commit and close the database session, then data will not save.



```
SQL> insert into students values (202,'12-jan-2016', 'Manish','Singh','Bisht', 'House No 219','Mohan
Nagar','Ghaziabad','Sh.Rakesh','Smt.Rashmi',9845434320,'22-oct-1999');

1 row created.

SQL> commit;

Commit complete.
```

MYSQL

Same as Oracle.

ROLLBACK

Rollback is used to undo the work performed by the current transaction.

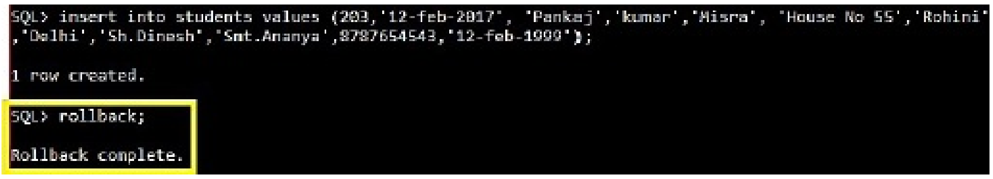
ORACLE

Syntax

Rollback;

Example

For example, in the following figure, all inserted values will be rolled back and not save in database table.



```
SQL> insert into students values (203,'12-feb-2017', 'Pankaj','kumar','Misra', 'House No 55','Rohini'
,'Delhi','Sh.Dinesh','Smt.Ananya',8787654543,'12-feb-1999');

1 row created.

SQL> rollback;

Rollback complete.
```

MYSQL

Same as Oracle.

SELECT

ORACLE

Select is used to retrieve records from database tables.

Select from a single table.

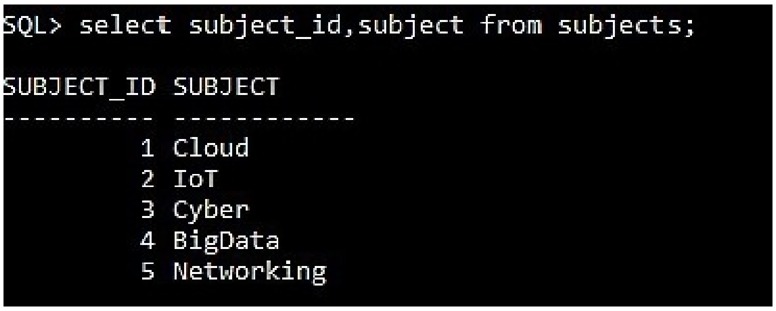
Syntax

```
SELECT column_name(s)
FROM
table_name
```

Example

Below query will select subject_id and subject from Subjects table.

```
Select
subject_id, subject
From
SUBJECTS;
```



```
SQL> select subject_id,subject from subjects;

SUBJECT_ID SUBJECT
---------- ------------
         1 Cloud
         2 IoT
         3 Cyber
         4 BigData
         5 Networking
```

Here, I have already inserted subject values in Subjects table using insert statements as mentioned below.

```
Insert into SUBJECTS values (1,   'Cloud');
Insert into SUBJECTS values (1,   'IoT');
Insert into SUBJECTS values (1,   'Cyber');
Insert into SUBJECTS values (1,   'BigData');
Insert into SUBJECTS values (1,   'Networking);
```

There are different wildcard characters which you can use with queries for a different purpose. I will explain it later in more details, but one of the very generic and basic wildcard character used with **SELECT** statement is an asterisk (*) which is used to retrieve all record from the table.

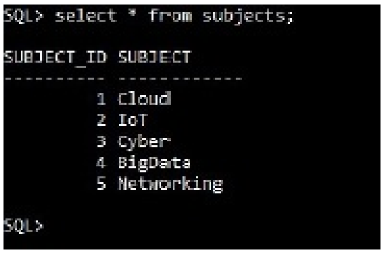
Syntax

```
SELECT *
FROM table_name;
```

Example

Following query will select all records from the subjects table.

```
SELECT * FROM subjects;
```



```
SQL> select * from subjects;

SUBJECT_ID SUBJECT
---------- ------------
         1 Cloud
         2 IoT
         3 Cyber
         4 BigData
         5 Networking

SQL>
```

Select values from multiple tables

To select Enrollment (from Students table), Subject (from Subjects table) and marks (from Marks table):

```
SELECT
Students.Enrollment_no,
Subjects.Subject,
Marks.Marks FROM Students, Subjects, Marks
WHERE Students.Enrollment_No=Marks.Enrollment_No AND
Subjects.Subject_Code=Marks.Subject_id;
```

Here, since we are selecting values from multiple tables by relating Primary and Foreign Keys, so we have to use table name as prefix before each field, because sometimes you have some field with the same name in multiple tables, and that time without using table name as prefix, database will confuse from which table you wanted to select value and will throw an error message.

To match the relationship, map *Primary Key* field of a table with the corresponding *Foreign Key* field.

For example, in this query, we are selecting records from three tables, where:

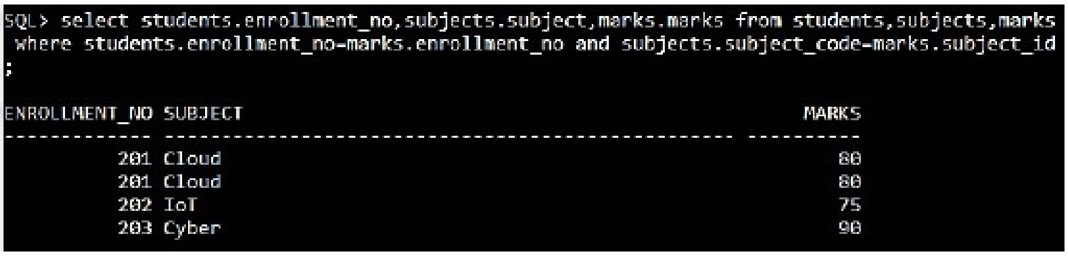
(a) **Enrollment_No** field in the Marks table is the *Foreign Key* of **Enrollment_No** field of

Students table, which is a *Primary Key*, hence we have used:

- **`Students.Enrollment_No=Marks.Enrollment_No`**

(b) Similarly, **Subject_id** field in the Marks table is the *Foreign Key* of **Subject_id** field of Subjects table, which is a *Primary Key*, hence we have used:

- **`Subjects.Subject_id = Marks.Subject_id;`**



```
SQL> select students.enrollment_no,subjects.subject,marks.marks from students,subjects,marks
 where students.enrollment_no=marks.enrollment_no and subjects.subject_code=marks.subject_id
;

ENROLLMENT_NO SUBJECT                                                MARKS
------------- ---------------------------------------------------- ----------
          201 Cloud                                                   80
          201 Cloud                                                   80
          202 IoT                                                     75
          203 Cyber                                                   90
```

MYSQL

Same as Oracle.

WHERE

ORACLE

WHERE clause is used to filter a particular record or set of records based on the specified condition.

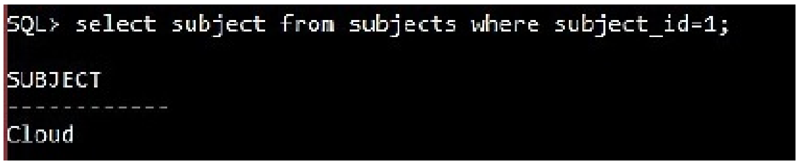
Syntax

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

Example

The following query will select a subject with code 1 from subjects table:

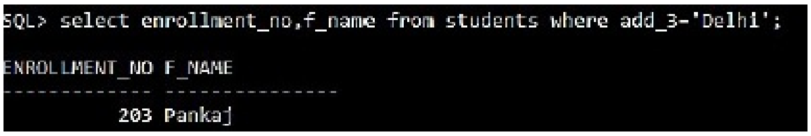
```
SELECT subject FROM subjects WHERE subject_id=1;
```



```
SQL> select subject from subjects where subject_id=1;

SUBJECT
------------
Cloud
```

Similarly, you can use the following query to select enrollment no. and name of all students from Delhi location.

```
SELECT enrollment_no,f_name FROM students WHERE add_3='Delhi';
```



```
SQL> select enrollment_no,f_name from students where add_3='Delhi';

ENROLLMENT_NO F_NAME
------------- ---------------
          203 Pankaj
```

MYSQL

Same as Oracle.

UPDATE

ORACLE

Update command is used to update existing record in a table.

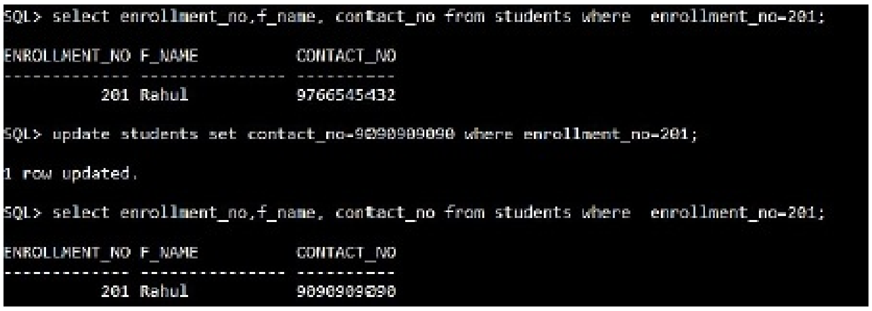
Syntax:

```
UPDATE table
SET
column 1 = expression 1,
column 2 = expression 2,
column n = expression n
WHERE conditions;
```

Example

Following command will update contact no. of a student with enrollment no 201 in subjects table.

```
UPDATE students SET contact_no=9090909090 WHERE enrollment_ no=201;
```



```
SQL> select enrollment_no,f_name, contact_no from students where  enrollment_no=201;

ENROLLMENT_NO F_NAME          CONTACT_NO
------------- --------------- ----------
          201 Rahul           9766545432

SQL> update students set contact_no=9090909090 where enrollment_no=201;

1 row updated.

SQL> select enrollment_no,f_name, contact_no from students where  enrollment_no=201;

ENROLLMENT_NO F_NAME          CONTACT_NO
------------- --------------- ----------
          201 Rahul           9090909090
```

MYSQL

Same as Oracle.

DELETE

DELETE command is used to delete existing record from a table.

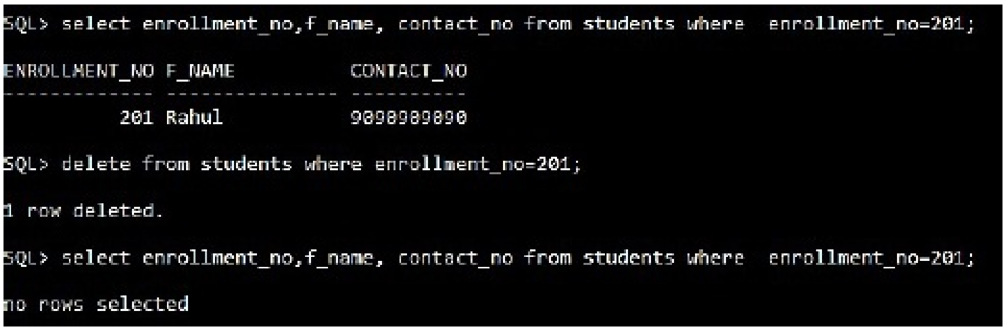
Syntax

DELETE FROM table
WHERE conditions;

Example

The following command will delete the record of a student with enrollment no 201 in students table.

```
DELETE FROM students WHERE enrollment_no=201;
```



```
SQL> select enrollment_no,f_name, contact_no from students where  enrollment_no=201;

ENROLLMENT_NO F_NAME          CONTACT_NO
------------- --------------- ----------
          201 Rahul           9898989890

SQL> delete from students where enrollment_no=201;

1 row deleted.

SQL> select enrollment_no,f_name, contact_no from students where  enrollment_no=201;

no rows selected
```

MYSQL

Same as Oracle.

ALTER

ORACLE

Alter command is used for multiple purposes such as add, delete, modify, delete (drop), rename a table. It's also used to add and delete (drop) constraints from a table.

Add new column

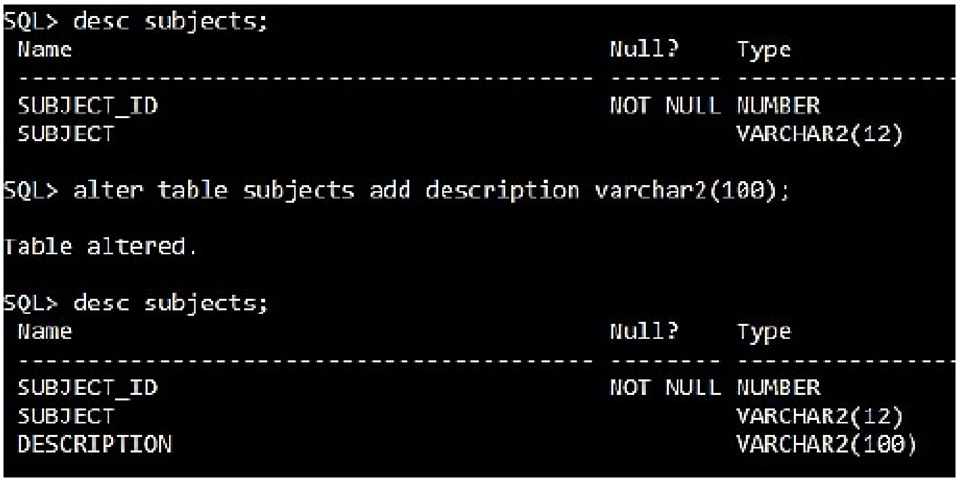
```
Syntax:
ALTER TABLE table_name ADD column_name datatype;;
```

Example

The following command will add a new column *description* in the subjects table;

```
ALTER TABLE subjects ADD description varchar2(100);
```

Example

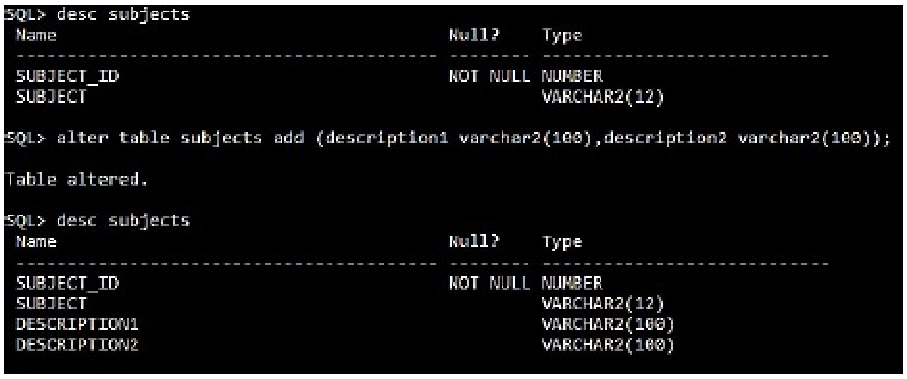


```
SQL> desc subjects;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 SUBJECT_ID                                NOT NULL NUMBER
 SUBJECT                                            VARCHAR2(12)

SQL> alter table subjects add description varchar2(100);

Table altered.

SQL> desc subjects;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 SUBJECT_ID                                NOT NULL NUMBER
 SUBJECT                                            VARCHAR2(12)
 DESCRIPTION                                        VARCHAR2(100)
```

Add multiple columns

```
ALTER TABLE subjects ADD (description1 varchar2(100), description2 varchar2(100));
```

Example



```
SQL> desc subjects
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 SUBJECT_ID                                NOT NULL NUMBER
 SUBJECT                                            VARCHAR2(12)

SQL> alter table subjects add (description1 varchar2(100),description2 varchar2(100));

Table altered.

SQL> desc subjects
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 SUBJECT_ID                                NOT NULL NUMBER
 SUBJECT                                            VARCHAR2(12)
 DESCRIPTION1                                       VARCHAR2(100)
 DESCRIPTION2                                       VARCHAR2(100)
```

Drop (Delete) Column

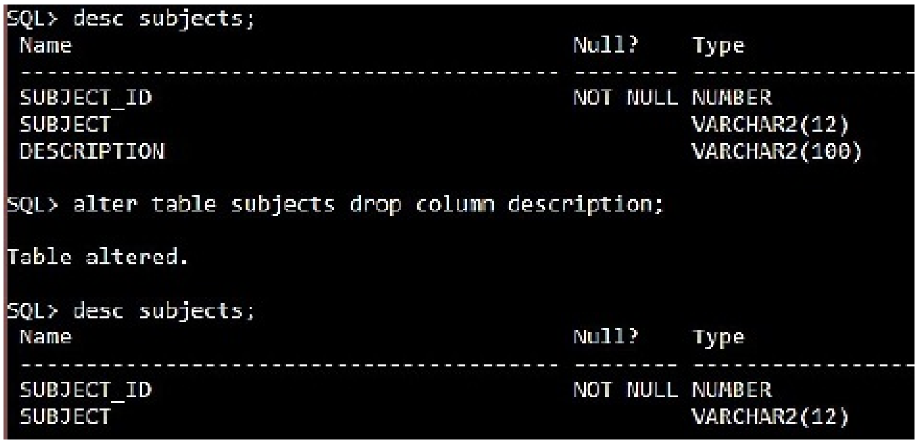
Syntax:

```
ALTER TABLE table_name
DROP column_name datatype;
```

Example:

The following command will drop the column *description* in the subjects table;

```
ALTER TABLE subjects DROP column description;
```



```
SQL> desc subjects;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 SUBJECT_ID                                NOT NULL NUMBER
 SUBJECT                                            VARCHAR2(12)
 DESCRIPTION                                        VARCHAR2(100)

SQL> alter table subjects drop column description;

Table altered.

SQL> desc subjects;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 SUBJECT_ID                                NOT NULL NUMBER
 SUBJECT                                            VARCHAR2(12)
```

Alter (Modify) a column

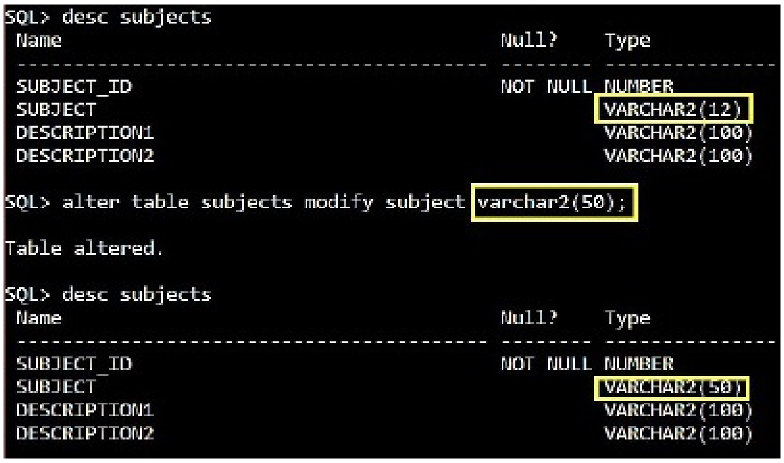
Syntax:

```
ALTER TABLE table_name
MODIFY column_name column_type;
```

Example

Following command will modify the character length of the subject field from 12 to 50.

```
ALTER TABLE subjects MODIFY subject varchar2(50);
```



```
SQL> desc subjects
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 SUBJECT_ID                                NOT NULL NUMBER
 SUBJECT                                            VARCHAR2(12)
 DESCRIPTION1                                       VARCHAR2(100)
 DESCRIPTION2                                       VARCHAR2(100)

SQL> alter table subjects modify subject varchar2(50);

Table altered.

SQL> desc subjects
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 SUBJECT_ID                                NOT NULL NUMBER
 SUBJECT                                            VARCHAR2(50)
 DESCRIPTION1                                       VARCHAR2(100)
 DESCRIPTION2                                       VARCHAR2(100)
```

Rename column

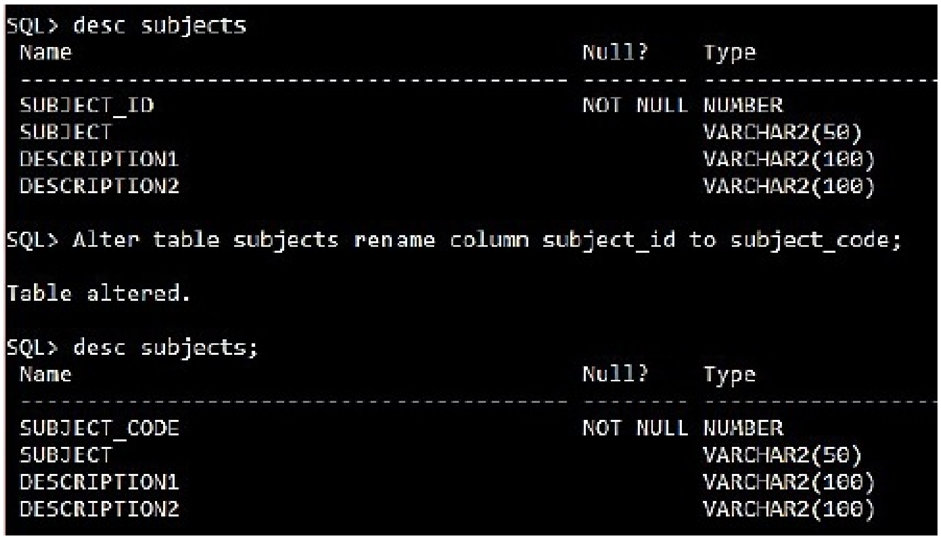
Syntax:

```
ALTER TABLE table_name
RENAME COLUMN old_name TO new_name;
```

Example

The following command will change column name **subject_id** to subject_ code.

```
ALTER TABLE subjects RENAME COLUMN subject_id TO subject_code;
```



```
SQL> desc subjects
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------------
 SUBJECT_ID                                NOT NULL NUMBER
 SUBJECT                                            VARCHAR2(50)
 DESCRIPTION1                                       VARCHAR2(100)
 DESCRIPTION2                                       VARCHAR2(100)

SQL> Alter table subjects rename column subject_id to subject_code;

Table altered.

SQL> desc subjects;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------------
 SUBJECT_CODE                              NOT NULL NUMBER
 SUBJECT                                            VARCHAR2(50)
 DESCRIPTION1                                       VARCHAR2(100)
 DESCRIPTION2                                       VARCHAR2(100)
```

Rename Table

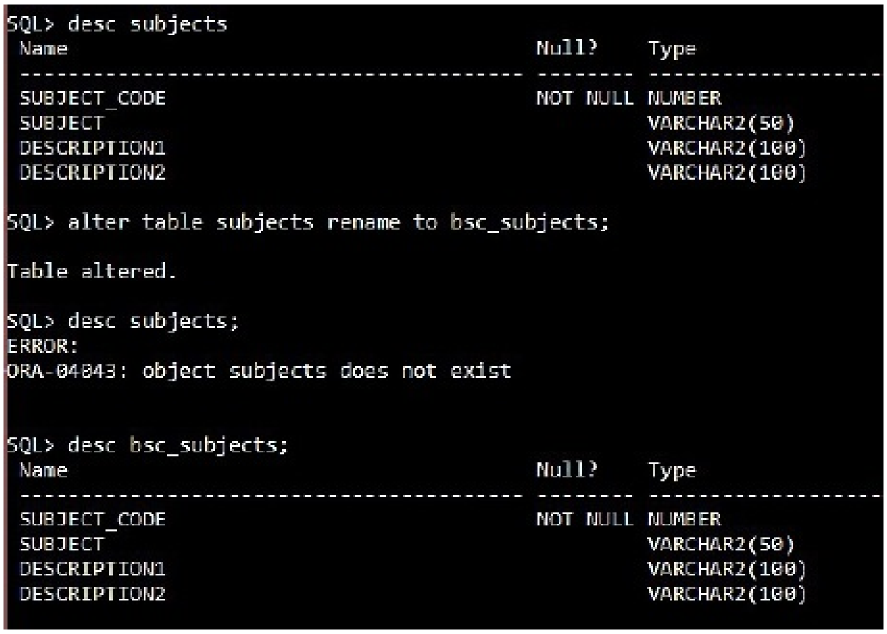
Syntax:

```
ALTER TABLE table_name
RENAME TO new_table_name;
```

Example

The following command will change the subjects table to **bsc_subjects**.

```
ALTER TABLE subjects RENAME TO bsc_subjects;
```



```
SQL> desc subjects
 Name                                      Null?    Type
 ----------------------------------------- -------- -------------------
 SUBJECT_CODE                              NOT NULL NUMBER
 SUBJECT                                            VARCHAR2(50)
 DESCRIPTION1                                       VARCHAR2(100)
 DESCRIPTION2                                       VARCHAR2(100)

SQL> alter table subjects rename to bsc_subjects;

Table altered.

SQL> desc subjects;
ERROR:
ORA-04043: object subjects does not exist


SQL> desc bsc_subjects;
 Name                                      Null?    Type
 ----------------------------------------- -------- -------------------
 SUBJECT_CODE                              NOT NULL NUMBER
 SUBJECT                                            VARCHAR2(50)
 DESCRIPTION1                                       VARCHAR2(100)
 DESCRIPTION2                                       VARCHAR2(100)
```

Add Constraint

If you have not defined primary or foreign key during table creation, or want to add later, then you can use Alter command as explained below.

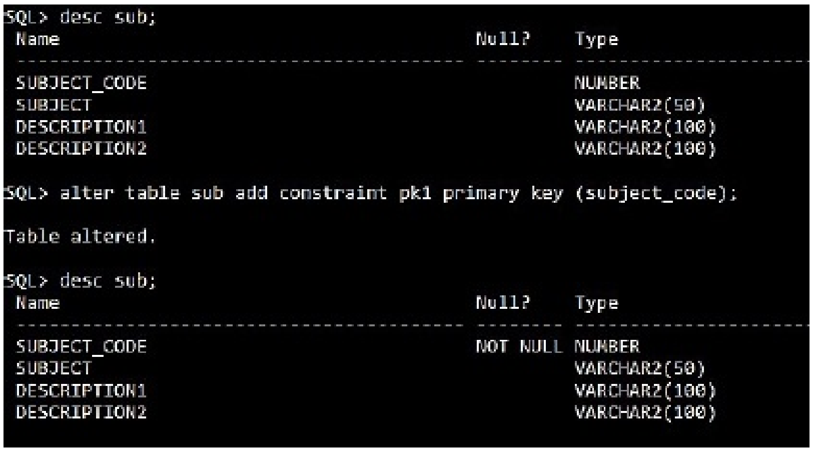
To add Primary Key

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name constraint_type (column);
```

Example

Following command will make **subject_code** as the primary key.

```
ALTER TABLE sub ADD CONSTRAINT pk1 primary key (subject_code);
```



```
SQL> desc sub;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------
 SUBJECT_CODE                                       NUMBER
 SUBJECT                                            VARCHAR2(50)
 DESCRIPTION1                                       VARCHAR2(100)
 DESCRIPTION2                                       VARCHAR2(100)

SQL> alter table sub add constraint pk1 primary key (subject_code);

Table altered.

SQL> desc sub;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------
 SUBJECT_CODE                              NOT NULL NUMBER
 SUBJECT                                            VARCHAR2(50)
 DESCRIPTION1                                       VARCHAR2(100)
 DESCRIPTION2                                       VARCHAR2(100)
```

To add Foreign Key

Syntax:

ALTER TABLE child_table_name ADD constraint constraint_name FOREIGN KEY (child_table_column_name) REFERENCES parent_table_ name (column_name);

Example

The following command will make a **subject_id** foreign key in table mark referencing primary key **subject_code** of table subjects.

```
ALTER TABLE mrk ADD constraint pk2 FOREIGN KEY (subject_id) REFERENCES subjects(subject_code);
```

MYSQL

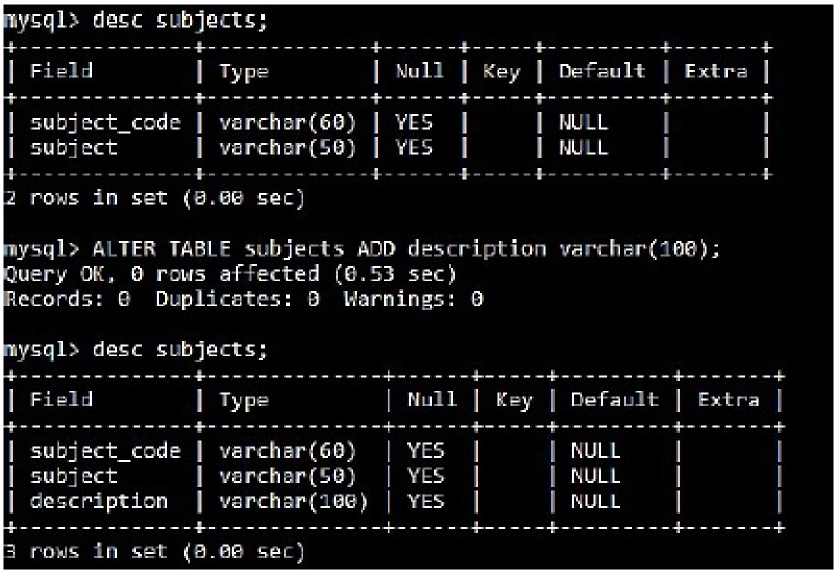
Add Column

Syntax

Same as Oracle. The only difference of specifying data type (Example - from Oracle varchar2 to Mysql varchar)

To add one column

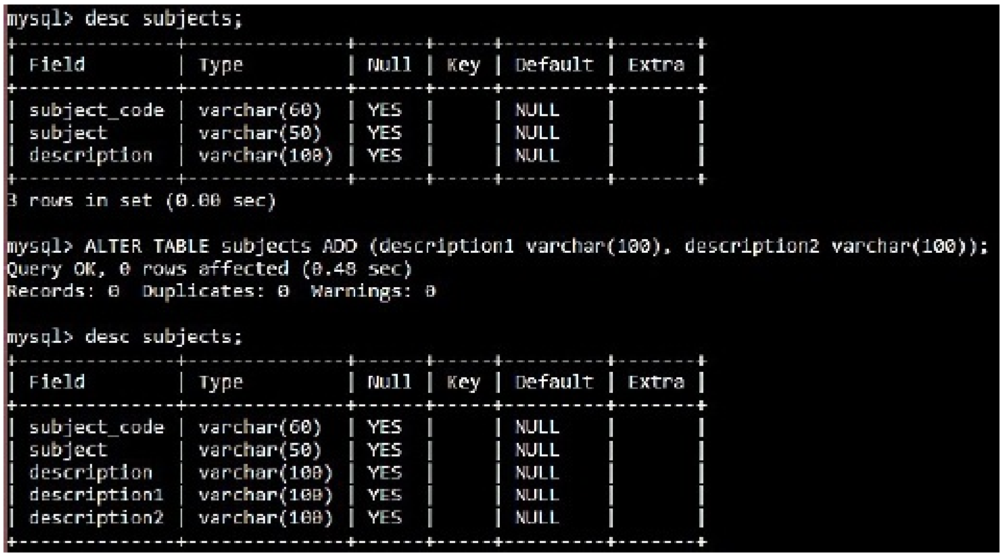
```
ALTER TABLE subjects ADD description varchar(100);
```



```
mysql> desc subjects;
+--------------+-------------+------+-----+---------+-------+
| Field        | Type        | Null | Key | Default | Extra |
+--------------+-------------+------+-----+---------+-------+
| subject_code | varchar(60) | YES  |     | NULL    |       |
| subject      | varchar(50) | YES  |     | NULL    |       |
+--------------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql> ALTER TABLE subjects ADD description varchar(100);
Query OK, 0 rows affected (0.53 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc subjects;
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| subject_code | varchar(60)  | YES  |     | NULL    |       |
| subject      | varchar(50)  | YES  |     | NULL    |       |
| description  | varchar(100) | YES  |     | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

To add multiple columns

```
ALTER TABLE subjects ADD (description1 varchar(100), description2 varchar(100));
```



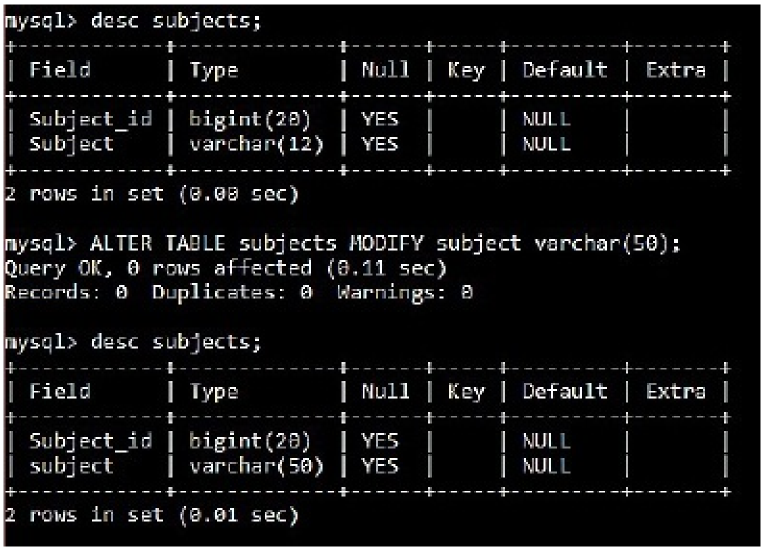
```
mysql> desc subjects;
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| subject_code | varchar(60)  | YES  |     | NULL    |       |
| subject      | varchar(50)  | YES  |     | NULL    |       |
| description  | varchar(100) | YES  |     | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> ALTER TABLE subjects ADD (description1 varchar(100), description2 varchar(100));
Query OK, 0 rows affected (0.48 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc subjects;
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| subject_code | varchar(60)  | YES  |     | NULL    |       |
| subject      | varchar(50)  | YES  |     | NULL    |       |
| description  | varchar(100) | YES  |     | NULL    |       |
| description1 | varchar(100) | YES  |     | NULL    |       |
| description2 | varchar(100) | YES  |     | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
```

Drop Column

Same as Oracle.

Modify Column

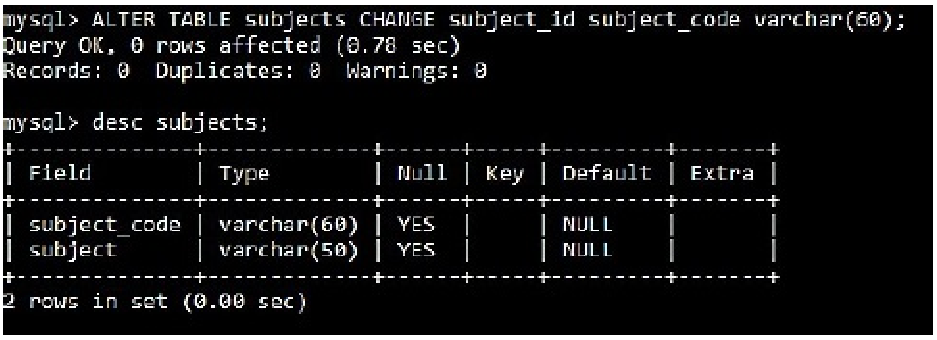
```
ALTER TABLE subjects MODIFY subject varchar(50);
```



```
mysql> desc subjects;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| Subject_id | bigint(20)  | YES  |     | NULL    |       |
| Subject    | varchar(12) | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)

mysql> ALTER TABLE subjects MODIFY subject varchar(50);
Query OK, 0 rows affected (0.11 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc subjects;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| Subject_id | bigint(20)  | YES  |     | NULL    |       |
| subject    | varchar(50) | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+
2 rows in set (0.01 sec)
```

Rename Column

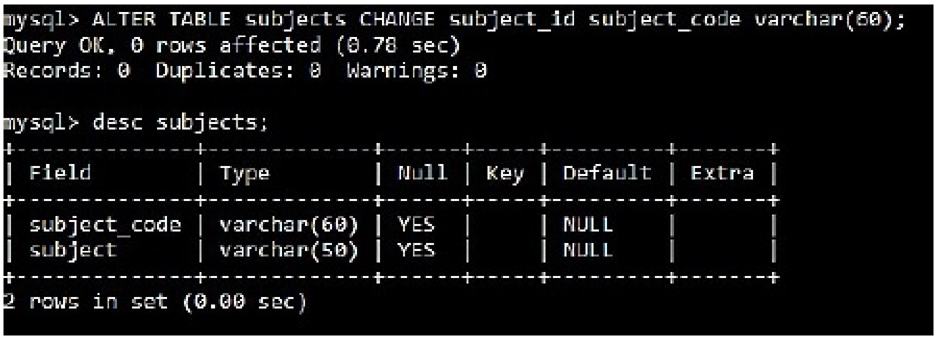
```
ALTER TABLE subjects CHANGE subject_id subject_code varchar(60);
```



```
mysql> ALTER TABLE subjects CHANGE subject_id subject_code varchar(60);
Query OK, 0 rows affected (0.78 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc subjects;
+--------------+-------------+------+-----+---------+-------+
| Field        | Type        | Null | Key | Default | Extra |
+--------------+-------------+------+-----+---------+-------+
| subject_code | varchar(60) | YES  |     | NULL    |       |
| subject      | varchar(50) | YES  |     | NULL    |       |
+--------------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

Rename Table

```
RENAME TABLE subjects TO bsc_subjects;
```



```
mysql> ALTER TABLE subjects CHANGE subject_id subject_code varchar(60);
Query OK, 0 rows affected (0.78 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc subjects;
+--------------+-------------+------+-----+---------+-------+
| Field        | Type        | Null | Key | Default | Extra |
+--------------+-------------+------+-----+---------+-------+
| subject_code | varchar(60) | YES  |     | NULL    |       |
| subject      | varchar(50) | YES  |     | NULL    |       |
+--------------+-------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

Add Constraints

It is used to add constraints (primary key, foreign key etc.) in the table.

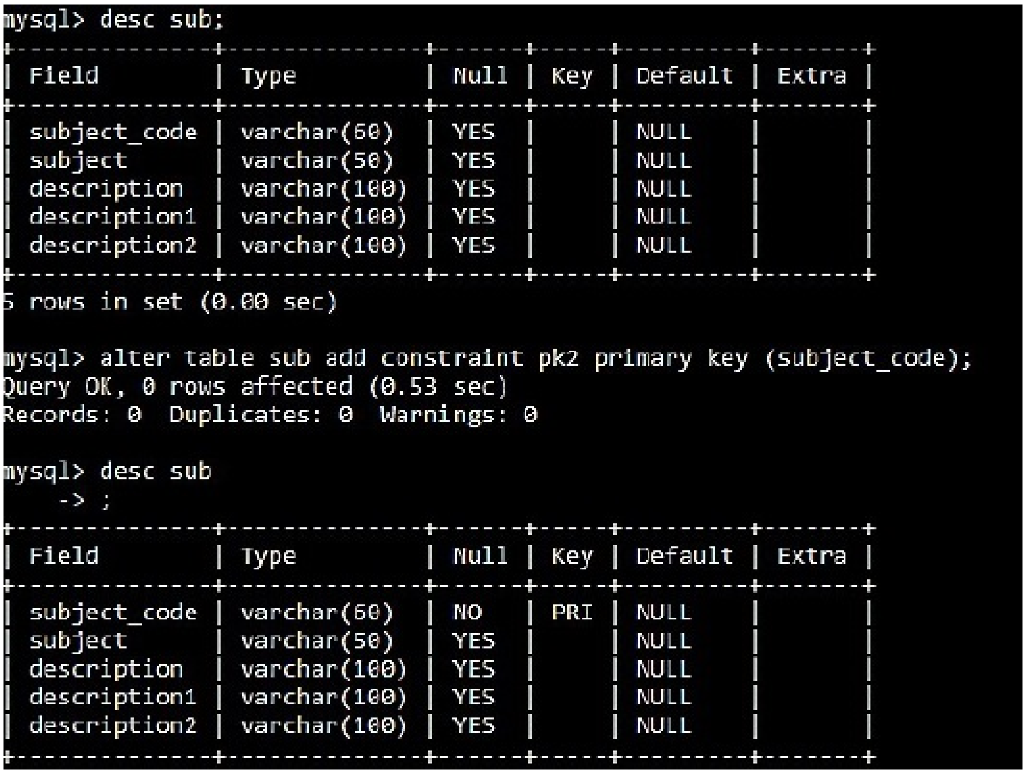
To add Primary Key

Syntax

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name constraint_type (column);
```

Example

```
Alter TABLE sub ADD CONSTRAINT pk2 PRIMARY KEY (subject_code);
```



```
mysql> desc sub;
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| subject_code | varchar(60)  | YES  |     | NULL    |       |
| subject      | varchar(50)  | YES  |     | NULL    |       |
| description  | varchar(100) | YES  |     | NULL    |       |
| description1 | varchar(100) | YES  |     | NULL    |       |
| description2 | varchar(100) | YES  |     | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
5 rows in set (0.00 sec)

mysql> alter table sub add constraint pk2 primary key (subject_code);
Query OK, 0 rows affected (0.53 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> desc sub
    -> ;
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| subject_code | varchar(60)  | NO   | PRI | NULL    |       |
| subject      | varchar(50)  | YES  |     | NULL    |       |
| description  | varchar(100) | YES  |     | NULL    |       |
| description1 | varchar(100) | YES  |     | NULL    |       |
| description2 | varchar(100) | YES  |     | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
```

DROP

The drop is used for deleting a table from the database. If the table has some referential integrities (Primary and Foreign Keys defined), then you have to specify Cascade Constraints otherwise you would not be able to drop table. Drop move table along with all data to oracle recycle bin and you still have some scope to recover table. In case there is some sensitive information which you want to erase permanently or don't want to put in recycle bin, then you can use Purge option with Drop.

ORACLE

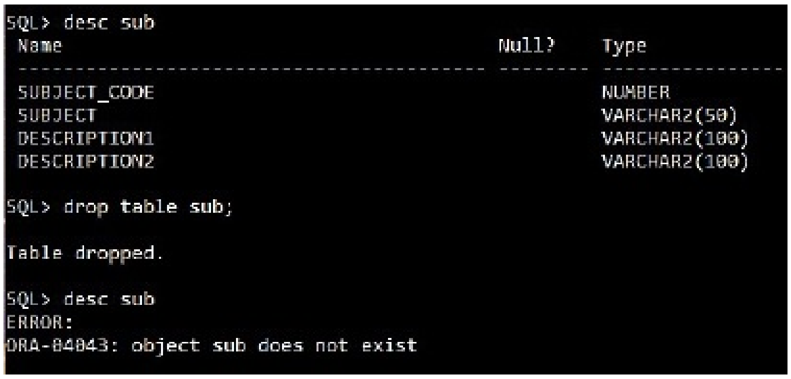
Syntax

```
DROP TABLE table_name
<CASCADE CONSTRAINTS>
<PURGE>;
```

Example

Following command will drop table sub.

```
DROP TABLE sub;
```



```
SQL> desc sub
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------
 SUBJECT_CODE                                       NUMBER
 SUBJECT                                            VARCHAR2(50)
 DESCRIPTION1                                       VARCHAR2(100)
 DESCRIPTION2                                       VARCHAR2(100)

SQL> drop table sub;

Table dropped.

SQL> desc sub
ERROR:
ORA-04043: object sub does not exist
```

Below command will drop table even in case of referential integrities:

```
DROP TABLE sub CASCADE CONSTRAINTS;
```

Following command will completely drop table from the database. That means it will not go to recycle bin and you will not be able to recover it later.

```
DROP TABLE sub PURGE;
```

MYSQL

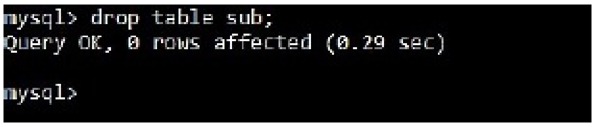
Syntax

```
DROP TABLE table_name;
```

Example

Following command will drop table sub.

```
DROP TABLE sub;
```



```
mysql> drop table sub;
Query OK, 0 rows affected (0.29 sec)

mysql>
```

TRUNCATE

Truncate is used to delete all the records from a table. It is similar to run **DELETE** command without **WHERE** clause, which will delete all records from the table, but the only difference is once executed, truncate statement can't be rollback.

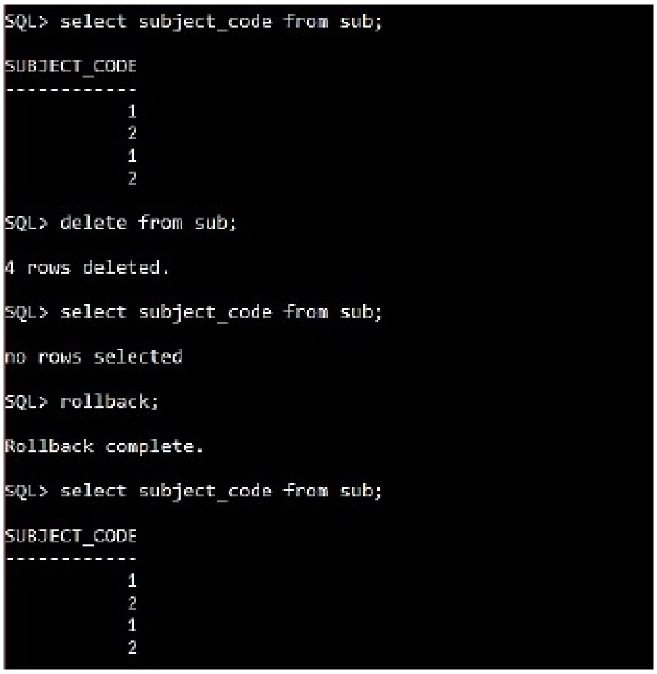
ORACLE

Syntax:

```
TRUNCATE TABLE table_name;
```

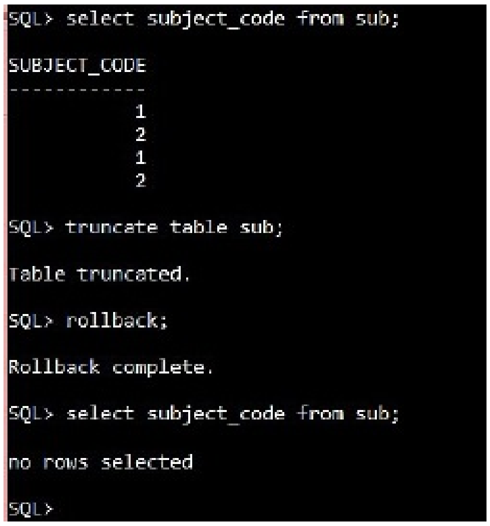
Example

With Delete, you can be able to recover using rollback.



```
SQL> select subject_code from sub;

SUBJECT_CODE
------------
           1
           2
           1
           2

SQL> delete from sub;

4 rows deleted.

SQL> select subject_code from sub;

no rows selected

SQL> rollback;

Rollback complete.

SQL> select subject_code from sub;

SUBJECT_CODE
------------
           1
           2
           1
           2
```

But you are not able to rollback with Truncate.



```
SQL> select subject_code from sub;

SUBJECT_CODE
------------
           1
           2
           1
           2

SQL> truncate table sub;

Table truncated.

SQL> rollback;

Rollback complete.

SQL> select subject_code from sub;

no rows selected

SQL>
```

MYSQL

Same as Oracle.

GRANT

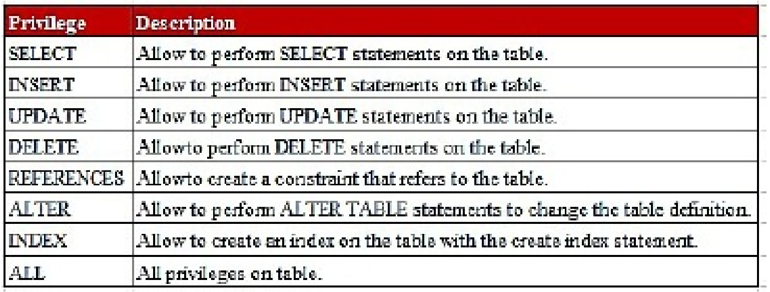
Grant is an Access Control command. You can use it to Grant access on various database objects *(Tables, Views,* etc) to users. For example, if you have different teams and you want to create different users for different teams or want to assign different privileges to different users such as read, write, delete, update etc. then you can restrict and control it easily with Grant command. Privileges can be any combination of **SELECT**, **INSERT**, **UPDATE**, **DELETE**, **REFERENCES**, **ALTER**, **INDEX**, or **ALL**.

ORACLE

Syntax

```
GRANT <Privileges> ON <object> TO <user>;
```

Where privileges could be one or combinations from the following screenshot:

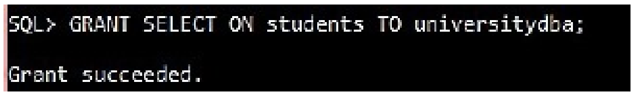


Privilege	Description
SELECT	Allow to perform SELECT statements on the table.
INSERT	Allow to perform INSERT statements on the table.
UPDATE	Allow to perform UPDATE statements on the table.
DELETE	Allowto perform DELETE statements on the table.
REFERENCES	Allowto create a constraint that refers to the table.
ALTER	Allow to perform ALTER TABLE statements to change the table definition.
INDEX	Allow to create an index on the table with the create index statement.
ALL	All privileges on table.

Example 1

Run the following command to grant select privileges on students table to universitydba user:

```
GRANT SELECT ON students TO universitydba;
```

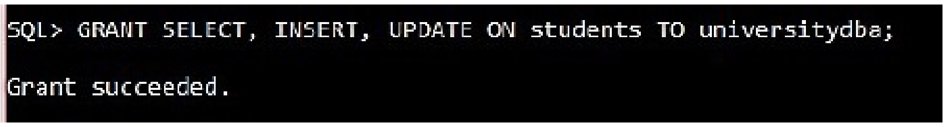


```
SQL> GRANT SELECT ON students TO universitydba;

Grant succeeded.
```

Example 2

Run the following command to grant **SELECT**, **INSERT** and **UPDATE** privileges on a table called subjects to a username universitydba

```
GRANT SELECT, INSERT, UPDATE ON students TO universitydba;
```

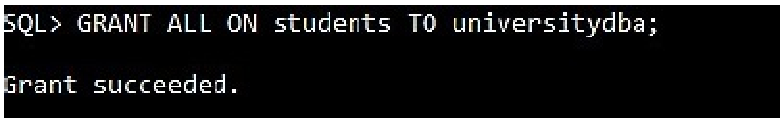


```
SQL> GRANT SELECT, INSERT, UPDATE ON students TO universitydba;

Grant succeeded.
```

Example 3

Run the following command to grant all permission to user universitydba:

```
GRANT ALL ON students TO universitydba;
```



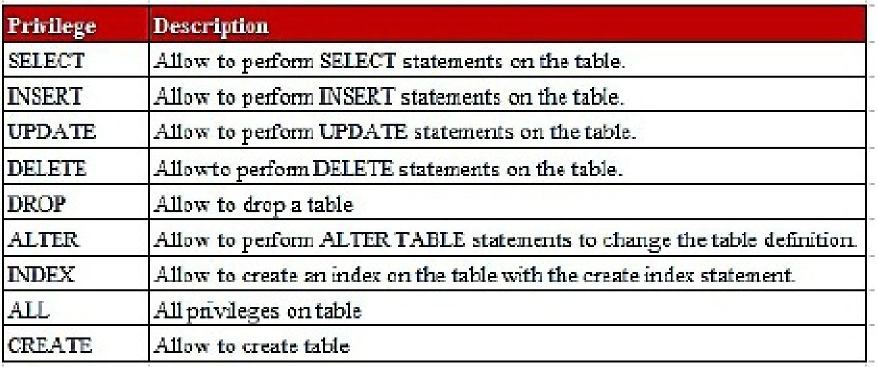
```
SQL> GRANT ALL ON students TO universitydba;

Grant succeeded.
```

MYSQL

Syntax

```
GRANT <Privileges> ON <object> TO <user>;
```

Where privileges could be one or combinations from the following table:

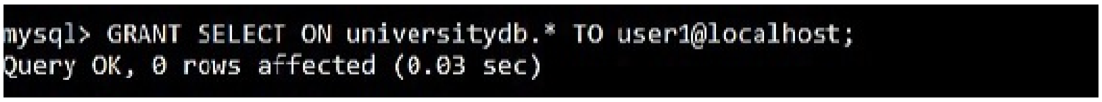


Privilege	Description
SELECT	Allow to perform SELECT statements on the table.
INSERT	Allow to perform INSERT statements on the table.
UPDATE	Allow to perform UPDATE statements on the table.
DELETE	Allowto perform DELETE statements on the table.
DROP	Allow to drop a table
ALTER	Allow to perform ALTER TABLE statements to change the table definition.
INDEX	Allow to create an index on the table with the create index statement.
ALL	All privileges on table
CREATE	Allow to create table

Example 1

Run the following command to grant **SELECT** privileges on all objects to a username user1 on the universitydb database:

```
GRANT SELECT ON universitydb.* TO user1@localhost;
```

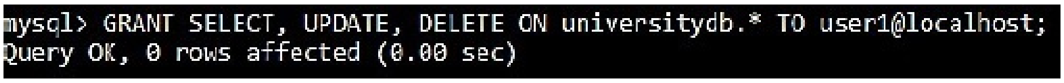


```
mysql> GRANT SELECT ON universitydb.* TO user1@localhost;
Query OK, 0 rows affected (0.03 sec)
```

Example 2

Run below command to grant **SELECT**, **UPDATE** and **DELETE** privileges on all objects to a username user1 on the universitydb database.

```
GRANT SELECT, UPDATE, DELETE ON universitydb.* TO user1@
localhost;
```

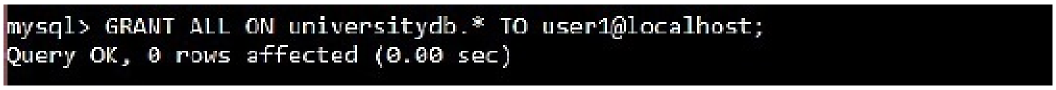


```
mysql> GRANT SELECT, UPDATE, DELETE ON universitydb.* TO user1@localhost;
Query OK, 0 rows affected (0.00 sec)
```

Example 3

Run the following command to grant all permission to a username user1 on the universitydb database.

```
GRANT ALL ON universitydb.* TO user1@localhost;
```

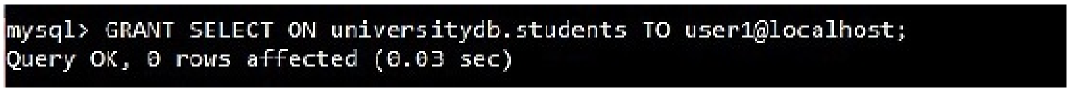


```
mysql> GRANT ALL ON universitydb.* TO user1@localhost;
Query OK, 0 rows affected (0.00 sec)
```

Example 4

Run the following command to grant **SELECT** privileges on students table to a username *user1* on the *universitydb* database

```
GRANT SELECT ON universitydb.students TO user1@localhost;
```



```
mysql> GRANT SELECT ON universitydb.students TO user1@localhost;
Query OK, 0 rows affected (0.03 sec)
```

Use the following command to list privileges of a user:

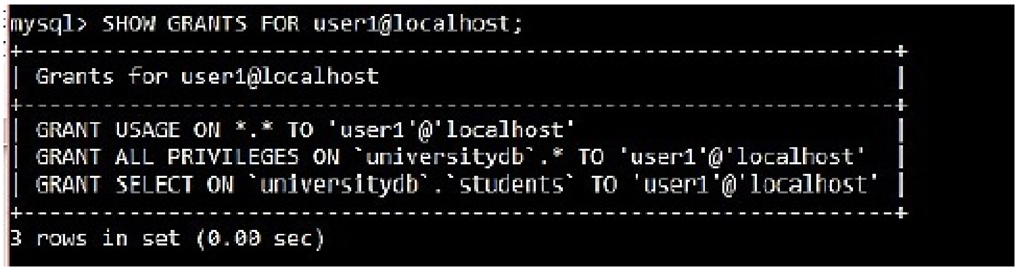
Syntax

```
SHOW GRANTS FOR <user>@<db_host>
```

Example

Run the following command to list the privileges of user with the name userl.

```
SHOW GRANTS FOR user1@localhost;
```



```
mysql> SHOW GRANTS FOR user1@localhost;
+-----------------------------------------------------------------+
| Grants for user1@localhost                                      |
+-----------------------------------------------------------------+
| GRANT USAGE ON *.* TO 'user1'@'localhost'                       |
| GRANT ALL PRIVILEGES ON `universitydb`.* TO 'user1'@'localhost' |
| GRANT SELECT ON `universitydb`.`students` TO 'user1'@'localhost' |
+-----------------------------------------------------------------+
3 rows in set (0.00 sec)
```

REVOKE

You can use this command to revoke the privileges granted with **GRANT** command.

ORACLE

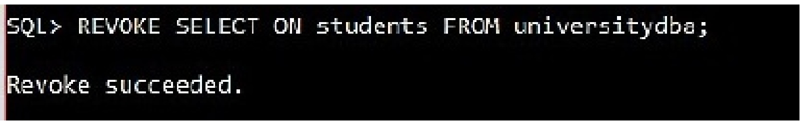
Syntax

```
REVOKE <Privileges> ON <object> FROM <user>;
```

Example 1

Run the following command to revoke a **SELECT** privilege from user **universitydba**.

```
REVOKE SELECT ON students FROM universitydba;
```

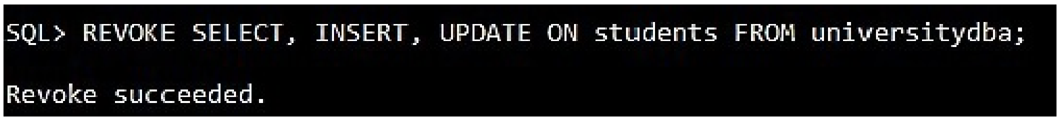


```
SQL> REVOKE SELECT ON students FROM universitydba;

Revoke succeeded.
```

Example 2

Run the following command to revoke **SELECT**, **INSERT,** and **UPDATE** privileges from user universitydba.

```
REVOKE SELECT, INSERT, UPDATE ON students FROM universitydba;
```

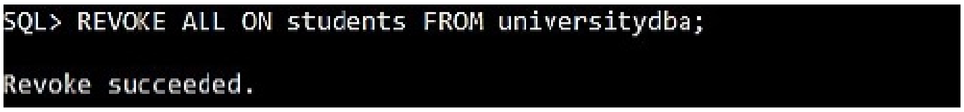


```
SQL> REVOKE SELECT, INSERT, UPDATE ON students FROM universitydba;

Revoke succeeded.
```

Example 3

Run the following command to revoke **ALL** privileges from user **universitydba**.

```
REVOKE ALL ON students FROM universitydba;
```



```
SQL> REVOKE ALL ON students FROM universitydba;

Revoke succeeded.
```

MYSQL

Syntax

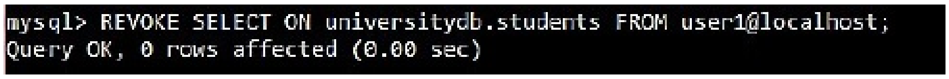
Same as Oracle.

```
REVOKE <Privileges> ON <object> FROM <user>;
```

Example

Run the following command to revoke **SELECT** privileges on the **universitydb** database from user **user1**.

```
REVOKE SELECT ON universitydb.students FROM user1@localhost;
```

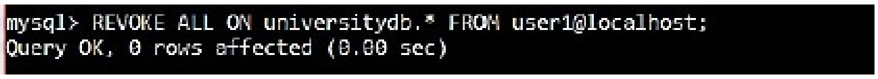


```
mysql> REVOKE SELECT ON universitydb.students FROM user1@localhost;
Query OK, 0 rows affected (0.00 sec)
```

Example 2

Run the following command to revoke **ALL** privileges on the **universitydb** database from user **for nurturing me with their immense knowledge.**.

```
REVOKE ALL ON universitydb.* FROM user1@localhost;
```



```
mysql> REVOKE ALL ON universitydb.* FROM user1@localhost;
Query OK, 0 rows affected (0.00 sec)
```

Distinct

The distinct clause is used with the only **SELECT** statement and it's used to remove duplicate values from the query output.

Syntax:

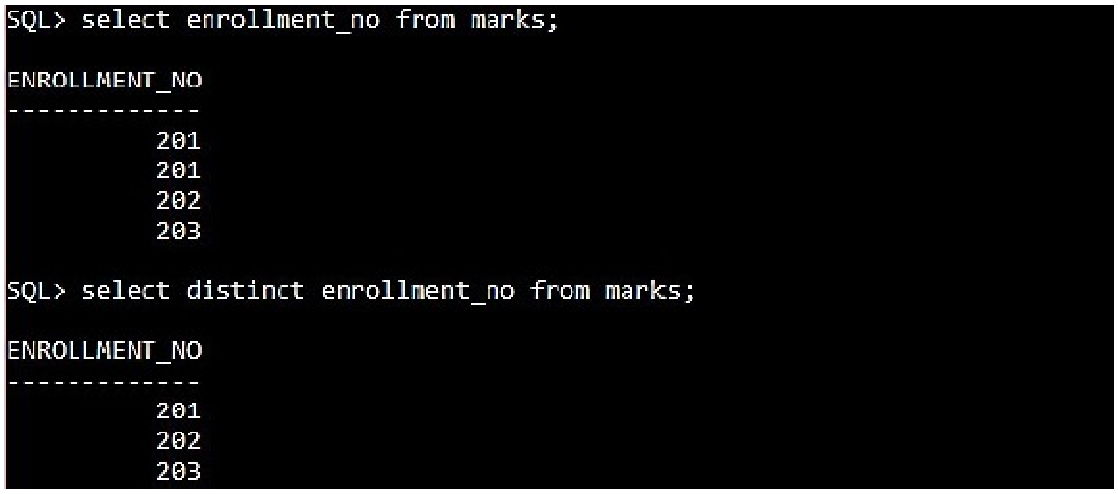
```
SELECT DISTINCT expressions
FROM tables
WHERE conditions;
```

Example

ORACLE

To select distinct enrollment numbers from marks table:

```
SELECT DISTINCT enrollment_no FROM marks;
```



```
SQL> select enrollment_no from marks;

ENROLLMENT_NO
-------------
          201
          201
          202
          203

SQL> select distinct enrollment_no from marks;

ENROLLMENT_NO
-------------
          201
          202
          203
```

MYSQL

Same as Oracle.

SQL Indexes

Indexes are used to make retrieval of data from tables faster. Without an index, a table is just like a phone directory without index page. As soon as pages will start increasing in the phone book, without index directory, it will make complicated and time taking to find a particular number. Similarly, as soon as data will start increasing in the table, it will make time taking to queries to retrieve records without an index. In another way, you can define it as a performance tuning method to make retrieval of data faster.

You can define an index on one or more number of columns.

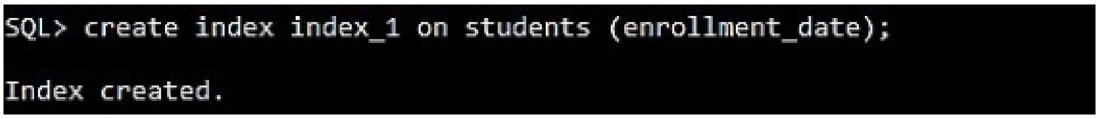
Syntax:

```
CREATE INDEX index_name
ON table_name (column 1, column 2, ... column n)
```

Examples

To create an index on **Enrollment_date** column:

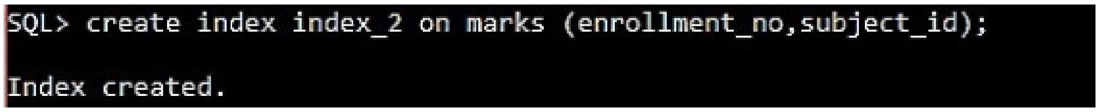
```
CREATE INDEX index_1 on STUDENTS (enrollment_date);
```



```
SQL> create index index_1 on students (enrollment_date);

Index created.
```

To create an index on two columns:

```
CREATE INDEX index_2 ON marks (enrollment_no,subject_id);
```



```
SQL> create index index_2 on marks (enrollment_no,subject_id);

Index created.
```

MYSQL

Same as Oracle.

SQL Alias

Alias is used to define a temporary name for columns and tables, especially to give meaningful names to columns in query output and to shorten the long length queries, where table name need to specify multiple times.

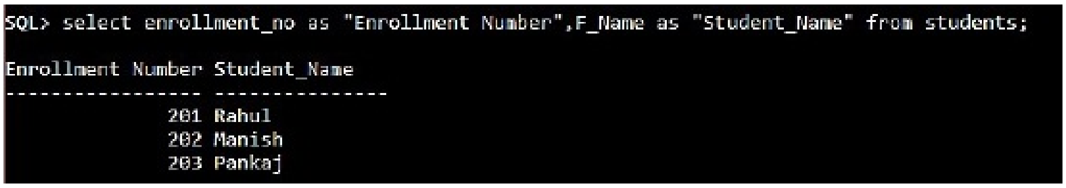
Syntax:

```
For Column: Column_Name Alias_Name
For Table: Table_Name Alias_Name
```

Example

At column level:

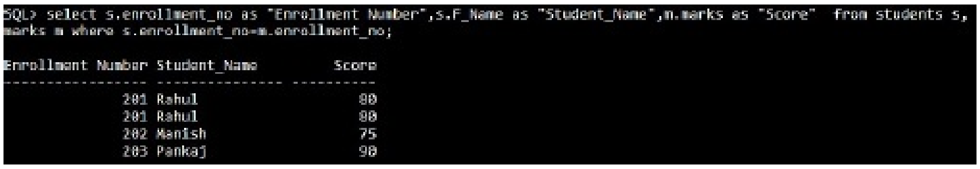
```
SELECT enrollment_no AS "Enrollment Number",F_Name AS "Student Name" FROM students;
```



```
SQL> select enrollment_no as "Enrollment Number",F_Name as "Student_Name" from students;

Enrollment Number Student_Name
----------------- ---------------
              201 Rahul
              202 Manish
              203 Pankaj
```

At table level:

SELECT s.enrollment_no AS “Enrollment Number”,s.F_Name AS “Student_ Name”,m.marks AS “Score” FROM STUDENTS s,marks m WHERE s.enroUment_no=m.enroUment_no;



```
SQL> select s.enrollment_no as "Enrollment Number",s.F_Name as "Student_Name",m.marks as "Score"  from students s,
marks m where s.enrollment_no=m.enrollment_no;

Enrollment Number Student_Name        Score
----------------- --------------- ----------
              201 Rahul                   80
              201 Rahul                   80
              202 Manish                  75
              203 Pankaj                  90
```

MYSQL

Same as Oracle.

SQL Sequences

Sequences are database objects used in tables to auto-generate sequential numbers. It's useful if you want to define a unique identity key (which will act as the primary key) for each record in a table however used rarely.

Oracle

```
Syntax
CREATE SEQUENCE sequence_name
MINVALUE value
MAXVALUE value
START WITH value
INCREMENT BY value
CACHE value;
```

There are few more options available in Syntax but those are required at advanced level learning.

INCREMENT BY

It tells the system how to increment the sequence. If it is positive, the values will be ascending and if it is negative, the values will be descending.

START WITH

It tells the system which integer to start with.

MINVALUE

It tells the system how low the sequence can go.

MAXVALUE

It tells the system, highest value that will be allowed.

CACHE

Caches the specified number of sequence values into the buffers to increase performance. The default value is 20 and the maximum value is max value-in value.

Example

ORACLE

```
CREATE SEQUENCE student_identification_no
MINVALUE 1
START WITH 1
INCREMENT BY 1
CACHE 20;
```

This will create a sequence student_identification_no which will start with 1 and will increment in sequence like 1,2,3.... etc. for each inserted record.

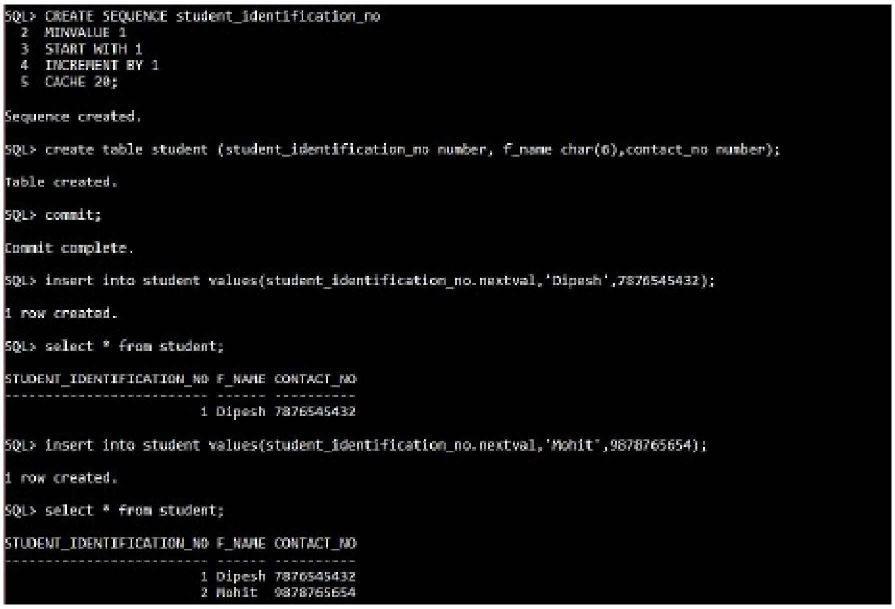
Drop Sequence

You can drop a sequence with a **DROP** statement.

```
DROP SEQUENCE sequence_name;
```

Use Sequence

You can use a sequence with **INSERT** command in the table.



```
SQL> CREATE SEQUENCE student_identification_no
  2  MINVALUE 1
  3  START WITH 1
  4  INCREMENT BY 1
  5  CACHE 20;

Sequence created.

SQL> create table student (student_identification_no number, f_name char(6),contact_no number);

Table created.

SQL> commit;

Commit complete.

SQL> insert into student values(student_identification_no.nextval,'Dipesh',7876545432);

1 row created.

SQL> select * from student;

STUDENT_IDENTIFICATION_NO F_NAME CONTACT_NO
------------------------- ------ ----------
                        1 Dipesh 7876545432

SQL> insert into student values(student_identification_no.nextval,'Mohit',9878765654);

1 row created.

SQL> select * from student;

STUDENT_IDENTIFICATION_NO F_NAME CONTACT_NO
------------------------- ------ ----------
                        1 Dipesh 7876545432
                        2 Mohit  9878765654
```

MYSQL

In MYSQL, we can create a column containing a sequence number using **AUTO_INCREMENT** option. Similar to Oracle, it's used to create a unique column in the table to act as *Primary Key.*

Syntax

```
CREATE TABLE table_name
(
columnl datatype NOT NULL AUTO_INCREMENT,
column2 datatype
   ...
   ...
column2 datatype
);
```

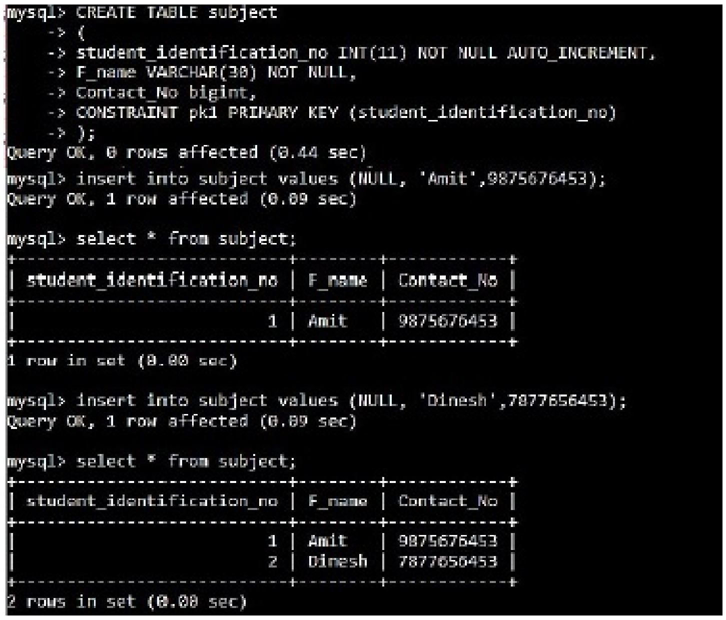
Example

```
CREATE TABLE subject
(
student_identification_no INT(11) NOT NULL AUTO_INCREMENT,
F_name VARCHAR(30) NOT NULL,
Contact_No bigint,
CONSTRAINT pk1 PRIMARY KEY (student_identification_no)
);
```

This will create a table subject with student_identification_no as a sequence number.

To insert a value just specify **NULL** in **student_identification_no** columns.

```
INSERT INTO subject VALUES (NULL, 'Amit',9875676453);
```



```
mysql> CREATE TABLE subject
    -> (
    -> student_identification_no INT(11) NOT NULL AUTO_INCREMENT,
    -> F_name VARCHAR(30) NOT NULL,
    -> Contact_No bigint,
    -> CONSTRAINT pk1 PRIMARY KEY (student_identification_no)
    -> );
Query OK, 0 rows affected (0.44 sec)
mysql> insert into subject values (NULL, 'Amit',9875676453);
Query OK, 1 row affected (0.09 sec)

mysql> select * from subject;
+---------------------------+--------+------------+
| student_identification_no | F_name | Contact_No |
+---------------------------+--------+------------+
|                         1 | Amit   | 9875676453 |
+---------------------------+--------+------------+
1 row in set (0.00 sec)

mysql> insert into subject values (NULL, 'Dinesh',7877656453);
Query OK, 1 row affected (0.09 sec)

mysql> select * from subject;
+---------------------------+--------+------------+
| student_identification_no | F_name | Contact_No |
+---------------------------+--------+------------+
|                         1 | Amit   | 9875676453 |
|                         2 | Dinesh | 7877656453 |
+---------------------------+--------+------------+
2 rows in set (0.00 sec)
```

SQL Views

A view is a virtual table, which provides access to a subset of the column from one or more table. A view created by querying one or more tables where the output of query stored as a view. It looks like a table but it does not take any space physically. It's just like a shortcut in windows, where shortcut always refers to the actual source file or directory.

Oracle

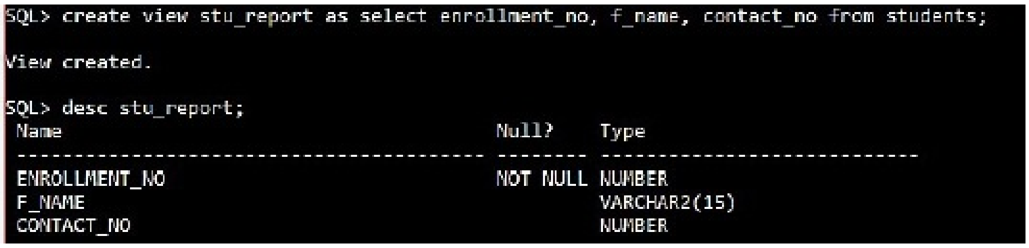
Syntax

```
CREATE VIEW view_name AS
SELECT columns
FROM tables
WHERE conditions;
```

Example

To create a view with name **stu_report** by selecting **enrollment_no**, **f_ name**, **contact_no** from students table:

CREATE VIEW stu_report AS SELECT enrollment_no, f_name, contact_no FROM students;



```
SQL> create view stu_report as select enrollment_no, f_name, contact_no from students;

View created.

SQL> desc stu_report;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 ENROLLMENT_NO                             NOT NULL NUMBER
 F_NAME                                             VARCHAR2(15)
 CONTACT_NO                                         NUMBER
```

It's just a view, hence no DML is allowed to execute on it.

MYSQL

Same as Oracle.

CHAPTER 7

Introduction to Database Joins

7.1. SQLJoins

SQL join is a method to relate and retrieve data from two or more tables.

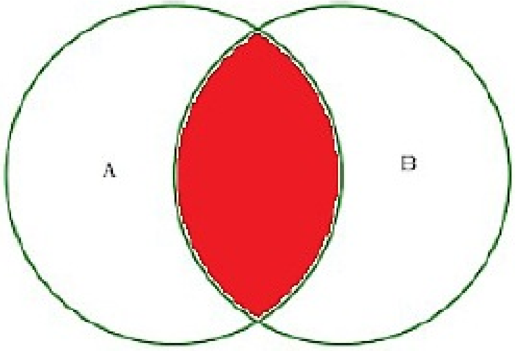
There are four major types of joins in the database as explained below:

Note

- When you join two or more tables, then adopt the standard practice to specify table name as a prefix in front of the fields you are selecting. For example, if you are selecting **roll_no** from students table, then write **students.roll_no**.
- Instead of full table name make a habit to use an alias name for tables to shorten the query. For example, if selecting **roll_no** from students table, then you can use “s” as an alias for the student's table, and for field selection, you can write **s.roll_no** instead of **students.roll_no**. Please refer next section to understand in detail with examples.

7.1.1 Inner Join (Also called SIMPLE JOIN)

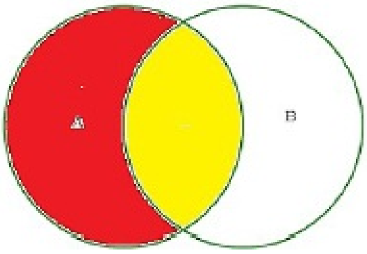
If there are two tables, **Table A** and **Table B**, then Inner Outer Join will return all records from **Table A** and **Table B** where join condition will meet.



Inner Join

7.1.2 Left Outer Join (Also called LEFT JOIN)

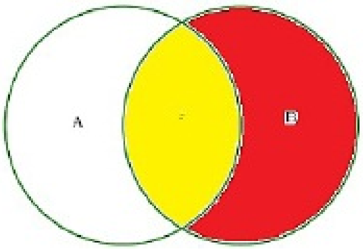
If there are two tables in **Table A** and **Table B**, then Left Outer Join will return all the records from **Table A** along with records from **Table B** where join condition will meet.



Left Outer Join

7.1.3 Right Outer Join (Also called RIGHT JOIN)

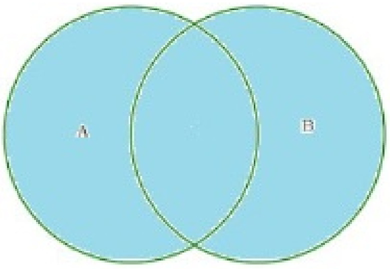
If there are two tables in **Table A** and **Table B**, then right outer join will return all records from **Table B** along with records from **Table A** where join condition will meet.



Right Outer Join

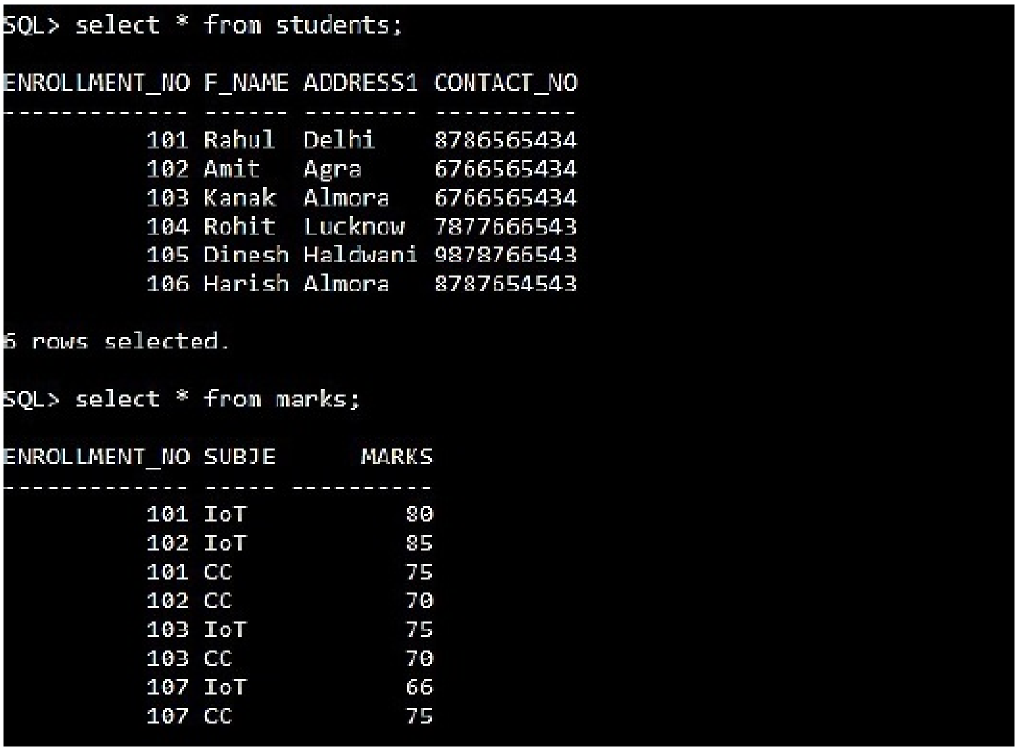
7.1.4 Full Outer Join (Also called FULL JOIN)

If there are two tables in **Table A** and **Table B**, then Full Outer Join will return all records from **Table A** and **Table B** regardless of whether the join condition met or not.



Full Outer Join

Let's try to understand with examples. Consider we have two sets of data in two tables, Students (Consider it as Table A) and Marks (Consider it as Table B) in our RDBMS. Both tables are related to keys, where **Enrollment id** is the *Primary Key* in **Students** table and *Foreign Key* in **Marks** table.

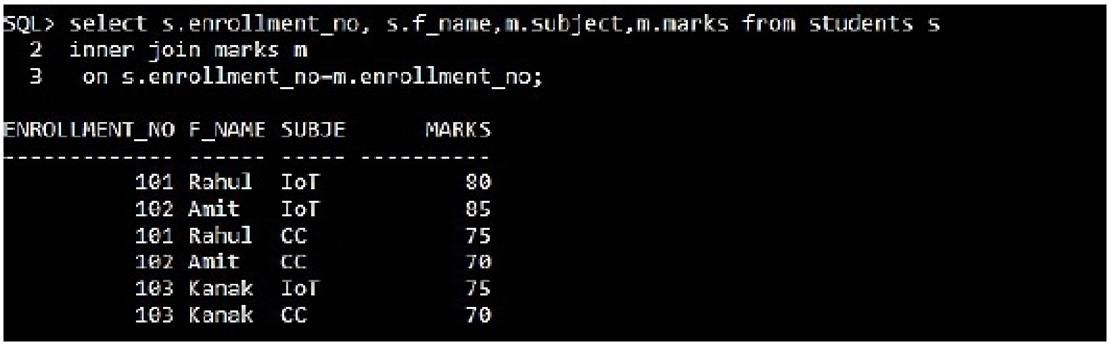


```
SQL> select * from students;

ENROLLMENT_NO F_NAME ADDRESS1 CONTACT_NO
------------- ------ -------- ----------
          101 Rahul  Delhi    8786565434
          102 Amit   Agra     6766565434
          103 Kanak  Almora   6766565434
          104 Rohit  Lucknow  7877666543
          105 Dinesh Haldwani 9878766543
          106 Harish Almora   8787654543

6 rows selected.

SQL> select * from marks;

ENROLLMENT_NO SUBJE      MARKS
------------- ----- ----------
          101 IoT           80
          102 IoT           85
          101 CC            75
          102 CC            70
          103 IoT           75
          103 CC            70
          107 IoT           66
          107 CC            75
```

7.1.5 Inner Join (Simple Join)

SELECT s.enrollment_no, s.f_name,m.subject,m.marks FROM students s
INNER JOIN marks m
ON s.enrollment_no=m.enrollment_no;

Output

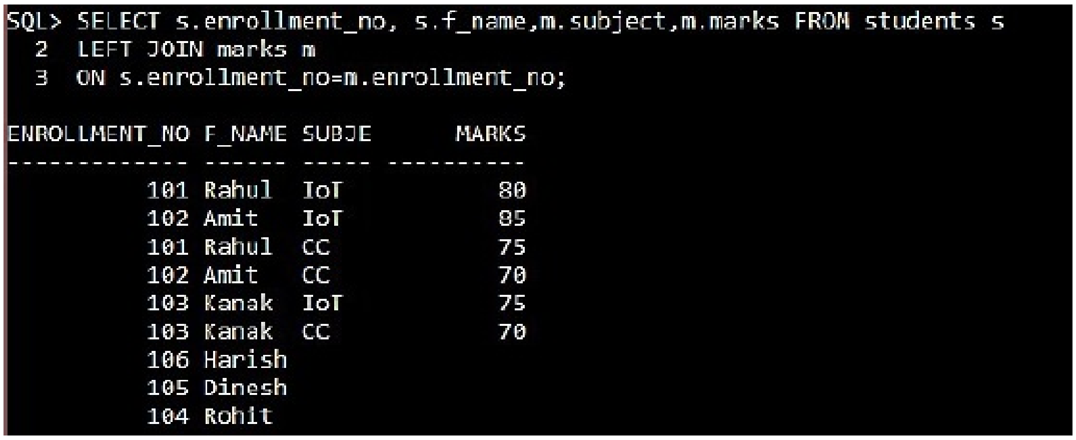


```
SQL> select s.enrollment_no, s.f_name,m.subject,m.marks from students s
  2  inner join marks m
  3   on s.enrollment_no=m.enrollment_no;

ENROLLMENT_NO F_NAME SUBJE      MARKS
------------- ------ ----- ----------
          101 Rahul  IoT           80
          102 Amit   IoT           85
          101 Rahul  CC            75
          102 Amit   CC            70
          103 Kanak  IoT           75
          103 Kanak  CC            70
```

7.1.6 Left Outer Join (Left Join)

```
SELECT s.enrollment_no, s.f_name,m.subject,m.marks FROM
students s
LEFT JOIN marks m
ON s.enrollment_no=m.enrollment_no;
```

Output

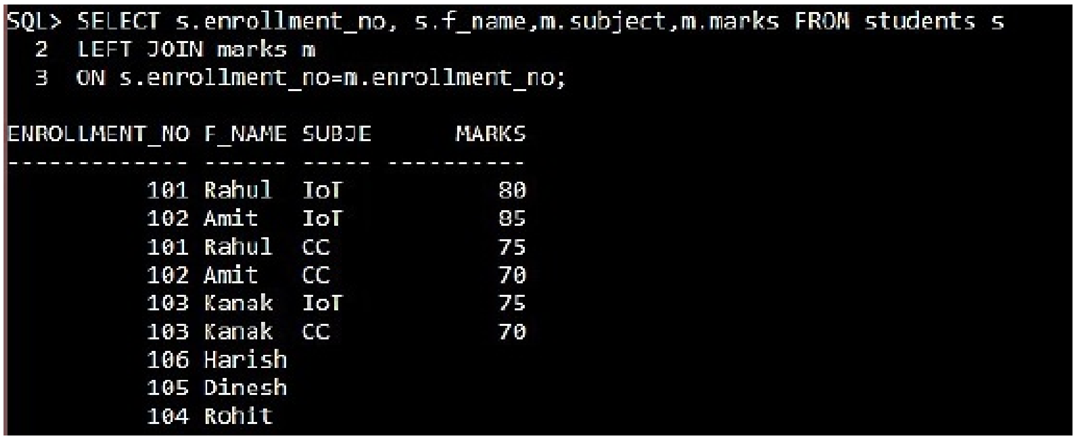


```
SQL> SELECT s.enrollment_no, s.f_name,m.subject,m.marks FROM students s
  2  LEFT JOIN marks m
  3  ON s.enrollment_no=m.enrollment_no;

ENROLLMENT_NO F_NAME SUBJE      MARKS
------------- ------ ----- ----------
          101 Rahul  IoT           80
          102 Amit   IoT           85
          101 Rahul  CC            75
          102 Amit   CC            70
          103 Kanak  IoT           75
          103 Kanak  CC            70
          106 Harish
          105 Dinesh
          104 Rohit
```

7.1.7 Right Outer Join (Also called Right Join)

```
SELECT s.enrollment_no, s.f_name,m.subject,m.marks FROM
students s
RIGHT JOIN marks m
ON s.enrollment_no=m.enrollment_no;
```

Output

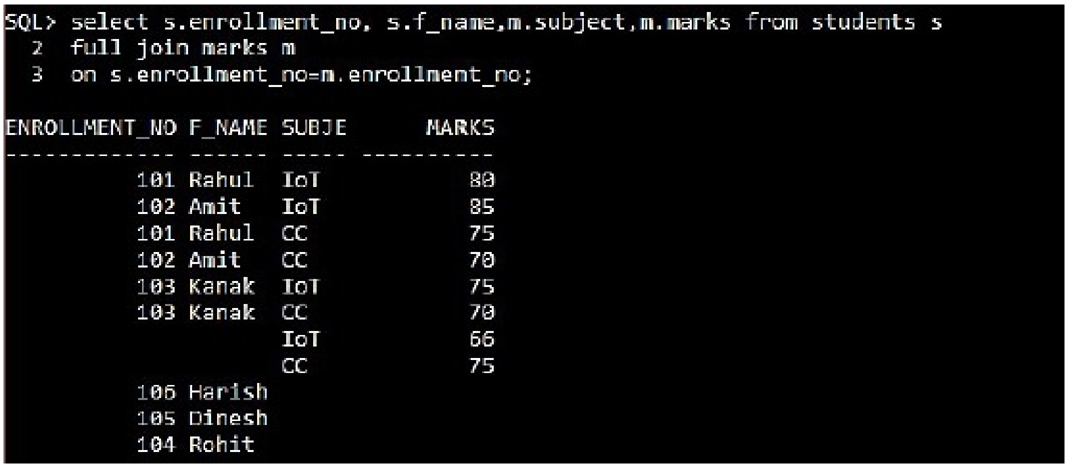


```
SQL> SELECT s.enrollment_no, s.f_name,m.subject,m.marks FROM students s
  2  LEFT JOIN marks m
  3  ON s.enrollment_no=m.enrollment_no;

ENROLLMENT_NO F_NAME SUBJE      MARKS
------------- ------ ----- ----------
          101 Rahul  IoT           80
          102 Amit   IoT           85
          101 Rahul  CC            75
          102 Amit   CC            70
          103 Kanak  IoT           75
          103 Kanak  CC            70
          106 Harish
          105 Dinesh
          104 Rohit
```

7.1.8 Full Outer Join (Also called Full Join)

```
SELECT s.enrollment_no, s.f_name,m.subject,m.marks FROM
students s
FULL JOIN marks m
on s.enrollment_no=m.enrollment_no;
```

Output



```
SQL> select s.enrollment_no, s.f_name,m.subject,m.marks from students s
  2  full join marks m
  3  on s.enrollment_no=m.enrollment_no;

ENROLLMENT_NO F_NAME SUBJE      MARKS
------------- ------ ----- ----------
          101 Rahul  IoT           80
          102 Amit   IoT           85
          101 Rahul  CC            75
          102 Amit   CC            70
          103 Kanak  IoT           75
          103 Kanak  CC            70
                     IoT           66
                     CC            75
          106 Harish
          105 Dinesh
          104 Rohit
```

CHAPTER 8

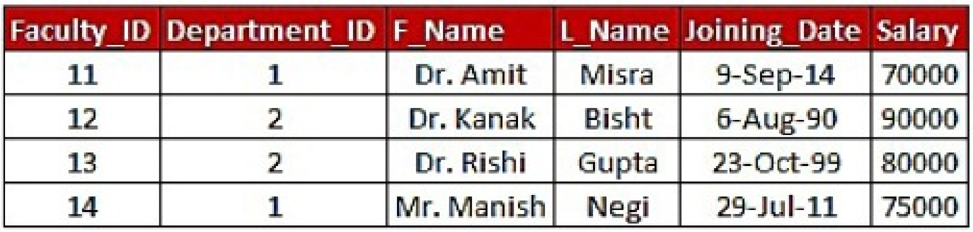
Aggregate functions, Subqueries and Users

8.1 Aggregate Functions

Aggregate functions apply on multiple rows in the group and return a single value for each group. For example, if you want to know the highest salary in each department, then you can use Max function by applying grouping (multiple rows for each department) on departments.

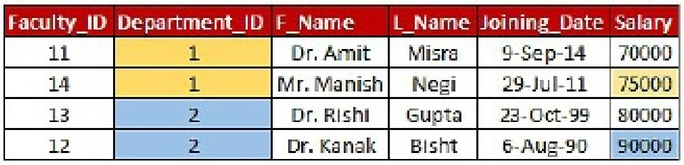
Aggregate functions commonly used with GROUP BY clause which divides rows into groups. For example, if you want to know the highest salary paid in each department, then you have to perform the department wise grouping first and its GROUP BY clause who actually does this. After that it applies the aggregate function supplied in the query such as MAX, MIN, AVG etc. and retrieve relevant results and values.

For example, let's consider that you have the following table with faculty data.



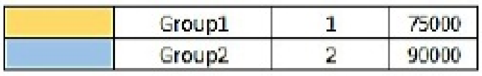
Faculty_ID	Department_ID	F_Name	L_Name	Joining_Date	Salary
11	1	Dr. Amit	Misra	9-Sep-14	70000
12	2	Dr. Kanak	Bisht	6-Aug-90	90000
13	2	Dr. Rishi	Gupta	23-Oct-99	80000
14	1	Mr. Manish	Negi	29-Jul-11	75000

If you want to retrieve maximum salary paid in each department, then **GROUP BY** clause will first do department wise grouping of records (as mentioned in a similar color in the following figure).



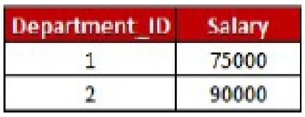
Faculty_ID	Department_ID	F_Name	L_Name	Joining_Date	Salary
11	1	Dr. Amit	Misra	9-Sep-14	70000
14	1	Mr. Manish	Negi	29-Jul-11	75000
13	2	Dr. Rishi	Gupta	23-Oct-99	80000
12	2	Dr. Kanak	Bisht	6-Aug-90	90000

Then it will identify the maximum salary from each group to show results.



🟨	Group1	1	75000
🟦	Group2	2	90000

Then you will get the output of query like the following table:



Department_ID	Salary
1	75000
2	90000

Following is the query for same (Oracle):

```
SELECT DEPARTMENT_ID, MAX(salary)
FROM
```