# SQL

## What is SQL?

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

## What Can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

## SQL Commands

The standard SQL commands to interact with relational databases are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP. These commands can be classified into the following groups based on their nature –

### DDL - Data Definition Language

| Sr.No. | Command & Description |
|---|---|
| 1 | **CREATE** |

| | |
|---|---|
| | Creates a new table, a view of a table, or other object in the database. |
| 2 | **ALTER**<br>Modifies an existing database object, such as a table. |
| 3 | **DROP**<br>Deletes an entire table, a view of a table or other objects in the database. |

## DML - Data Manipulation Language

| Sr.No. | Command & Description |
|---|---|
| 1 | **SELECT**<br>Retrieves certain records from one or more tables. |
| 2 | **INSERT**<br>Creates a record. |
| 3 | **UPDATE**<br>Modifies records. |
| 4 | **DELETE**<br>Deletes records. |

## DCL - Data Control Language

| Sr.No. | Command & Description |
|---|---|
| 1 | **GRANT**<br>Gives a privilege to user. |
| 2 | **REVOKE**<br>Takes back privileges granted from user. |

**What are the different subsets of SQL?**

DDL (Data Definition Language) – It allows you to perform various operations on the database such as CREATE, ALTER and DELETE objects.

DML ( Data Manipulation Language) – It allows you to access and manipulate data. It helps you to insert, update, delete and retrieve data from the database.

DCL ( Data Control Language) – It allows you to control access to the database. Example – Grant, Revoke access permissions.

**What do you mean by DBMS? What are its different types?**

database is a structured collection of data.

A Database Management System (DBMS) is a software application that interacts with the user, applications and the database itself to capture and analyze data.

A DBMS allows a user to interact with the database. The data stored in the database can be modified, retrieved and deleted and can be of any type like strings, numbers, images etc.

**There are two types of DBMS:**

Relational Database Management System: The data is stored in relations (tables). Example – MySQL, OracleSQL.

Non-Relational Database Management System: There is no concept of relations, tuples and attributes. Example – Mongo

What do you mean by table and field in SQL?

A table refers to a collection of data in an organised manner in form of rows and columns. A field refers to the number of columns in a table. For example:

Table: StudentInformation

Field: Stu Id, Stu Name, Stu Marks

**What are joins in SQL?**

A JOIN clause is used to combine rows from two or more tables, based on a related column between them. It is used to merge two tables or retrieve data from there. There are 4 joins in SQL namely:

1. Inner Join
2. Right Join
3. Left Join
4. Full Join

**What is the difference between CHAR and VARCHAR2 datatype in SQL?**

Both Char and Varchar2 are used for characters datatype but varchar2 is used for character strings of variable length whereas Char is used for strings of fixed length. For example, char(10) can only store 10 characters and will not be able to store a string of any other length whereas varchar2(10) can store any length i.e 6,8,2 in this variable.

**What is a Primary key?**

Primary key is a column (or collection of columns) or a set of columns that uniquely identifies each row in the table.

Uniquely identifies a single row in the table

Null values not allowed

Example- In the Student table, Stu_ID is the primary key.

**What are Constraints?**

Constraints are used to specify the limit on the data type of the table. It can be specified while creating or altering the table statement. The sample of constraints are:

1. NOT NULL
2. CHECK
3. DEFAULT
4. UNIQUE
5. PRIMARY KEY
6. FOREIGN KEY

## What is the difference between DELETE and TRUNCATE statements?

DELETE vs TRUNCATE

| DELETE | TRUNCATE |
|---|---|
| Delete command is used to delete a row in a table. | Truncate is used to delete all the rows from a table. |
| You can rollback data after using delete statement. | You cannot rollback data. |
| It is a DML command. | It is a DDL command. |
| It is slower than truncate statement. | It is faster. |

**What is a Unique key?**

Uniquely identifies a single row in the table.

Multiple values allowed per table.

Null values allowed.

Apart from this SQL Interview Questions blog, if you want to get trained from professionals on this technology, you can opt for a structured training from edureka! Click below to know more.

**What is a Foreign key?**

Foreign key maintains referential integrity by enforcing a link between the data in two tables.

The foreign key in the child table references the primary key in the parent table.

The foreign key constraint prevents actions that would destroy links between the child and parent tables.

**List the different type of joins?**

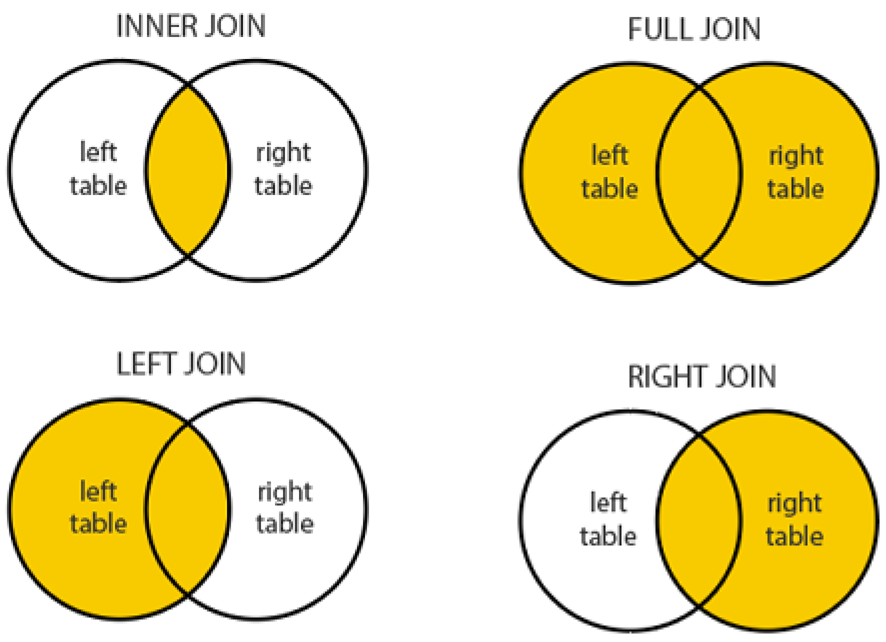
There are various types of joins which are used to retrieve data between the tables. There are four types of joins, namely:

**Inner join:** Inner Join in MySQL is the most common type of join. It is used to return all the rows from multiple tables where the join condition is satisfied.

**Left Join:** Left Join in MySQL is used to return all the rows from the left table but only the matching rows from the right table where the join condition is fulfilled.

**Right Join:** Right Join in MySQL is used to return all the rows from the right table but only the matching rows from the left table where the join condition is fulfilled.

**Full Join:** Full join returns all the records when there is a match in any of the tables. Therefore, it returns all the rows from the left-hand side table and all the rows from the right-hand side table.



**Please follow the link to Learn how download and install Oracle Database and SQL Developer**

[https://www.youtube.com/watch?v=UnvZvaYEMak&list=PLcrUOdq4zdt0pr4tC84uGWXogNxjnzFPo](https://www.youtube.com/watch?v=UnvZvaYEMak&list=PLcrUOdq4zdt0pr4tC84uGWXogNxjnzFPo)

1. DDL (CREATE, ALTER, DROP, TRUNCATE)
2. DML (INSERT, UPDATE, DELETE)
3. DQL (SELECT)
4. DCL (GRANT, REVOKE)

KEYWORDS AND IDENTIERS ARE CASE INSENSITIVE

LITERALS ARE CASE SENSITIVE

```
CREATE TABLE PERSON(ID NUMBER, FIRST_NAME VARCHAR2(90), AGE
NUMBER);
```

```
Table PERSON created.



INSERT INTO PERSON VALUES(1, 'VIJAY', 22);
1 row inserted.

INSERT INTO PERSON VALUES(2, 'MANU', 25);
INSERT INTO PERSON VALUES(3, 'ARUN', 26);
INSERT INTO PERSON VALUES(4, 'KUMAR', 28);
INSERT INTO PERSON VALUES(5, 'KIRAN', 25);
INSERT INTO PERSON VALUES(6, 'JYOTHI', 20);
INSERT INTO PERSON VALUES(7, 'RAMU', 35);

1 row inserted.



1 row inserted.



1 row inserted.



1 row inserted.



1 row inserted.
```

```
1 row inserted.

INSERT INTO PERSON VALUES(7, 'RAMU', 35);

1 row inserted.



INSERT INTO PERSON(ID, FIRST_NAME) VALUES(8, 'MURALI');

1 row inserted.



INSERT INTO PERSON(ID, AGE) VALUES(9, 45);

1 row inserted.



INSERT INTO PERSON(FIRST_NAME, AGE) VALUES('MOHAN', 45);

1 row inserted.

INSERT INTO PERSON(AGE) VALUES(45);

1 row inserted.

INSERT INTO PERSON(AGE, ID) VALUES(45, 12);

1 row inserted.

INSERT INTO PERSON(AGE, ID, FIRST_NAME) VALUES(48, 13,
'RAGHU');

1 row inserted.
```

```
INSERT INTO PERSON(AGE, FIRST_NAME, ID) VALUES(48, 'RUPA',
14);

1 row inserted.


UPDATE PERSON SET FIRST_NAME='RAVI' WHERE ID = 5;

1 row updated.


UPDATE PERSON SET FIRST_NAME='RAMANA', AGE=33 WHERE ID = 6;

1 row updated.


UPDATE PERSON SET ID=25, AGE=38 WHERE ID = 7;

2 rows updated.



UPDATE PERSON SET AGE=28 WHERE AGE IS NULL;

1 row updated.


UPDATE PERSON SET AGE=38 WHERE FIRST_NAME IS NOT NULL;

12 rows updated.


UPDATE PERSON SET FIRST_NAME='ANU' WHERE ID > 4;

9 rows updated.


UPDATE PERSON SET AGE=22, ID = 10 WHERE ID <= 7;

6 rows updated.
```

```
UPDATE PERSON SET AGE=42, ID = 15;

15 rows updated.


DELETE FROM PERSON WHERE ID = 6;

0 rows deleted.


DELETE FROM PERSON WHERE FIRST_NAME = 'KUMAR';

1 row deleted.


DELETE FROM PERSON WHERE FIRST_NAME = 'ANU';

9 rows deleted.


DELETE FROM PERSON;

5 rows deleted.


DROP TABLE PERSON;



CREATE TABLE EMPLOYEE (ID NUMBER,
                     FIRST_NAME VARCHAR2(90),
                     LAST_NAME VARCHAR2(90),
                     AGE NUMBER,
             SALARY NUMBER,
                     EMAIL VARCHAR2(90));


INSERT INTO EMPLOYEE VALUES(1, 'RAMU', 'BTM', 22, 4000,
'R@G.IN');
```

```
INSERT INTO EMPLOYEE VALUES(2, 'RAVI', 'RAO', 24, 3000,
'R@G.IN');
INSERT INTO EMPLOYEE VALUES(3, 'MANU', 'BTM', 23, 5000,
'M@G.IN');
INSERT INTO EMPLOYEE VALUES(4, 'MURALI', 'RAO', 26, 5500,
'M@G.IN');
INSERT INTO EMPLOYEE VALUES(5, 'RAGHU', 'JD', 25, 2000,
'R@G.IN');
INSERT INTO EMPLOYEE VALUES(6, 'PAVAN', 'NAO', 28, 3500,
'P@G.IN');
INSERT INTO EMPLOYEE VALUES(7, 'ANU', 'JP', 21, 6000,
'A@G.IN');
INSERT INTO EMPLOYEE VALUES(8, 'SWETHA', 'NAO', 29, 5600,
'W@G.IN');
INSERT INTO EMPLOYEE VALUES(9, 'SUNITHA', 'JD', 32, 7000,
'SU@G.IN');
INSERT INTO EMPLOYEE VALUES(10, 'KIRAN', 'JP', 30, 5300,
'K@G.IN');


SELECT * FROM EMPLOYEE;
SELECT * FROM EMPLOYEE WHERE ID = 5;
SELECT * FROM EMPLOYEE WHERE ID > 5;
SELECT * FROM EMPLOYEE WHERE AGE BETWEEN 22 AND 28;
SELECT * FROM EMPLOYEE WHERE AGE NOT BETWEEN 22 AND 28;
SELECT * FROM EMPLOYEE WHERE SALARY IN (4000, 5500, 7000);
SELECT * FROM EMPLOYEE WHERE SALARY NOT IN (4000, 5500,
7000);
SELECT * FROM EMPLOYEE WHERE FIRST_NAME LIKE '%R%';
SELECT FIRST_NAME, AGE FROM EMPLOYEE;
SELECT FIRST_NAME, AGE, EMAIL FROM EMPLOYEE;
SELECT FIRST_NAME, ID, AGE FROM EMPLOYEE;

SELECT FIRST_NAME, AGE, LAST_NAME FROM EMPLOYEE;

SELECT FIRST_NAME AS MYNAME, AGE, LAST_NAME FROM EMPLOYEE;

SELECT FIRST_NAME AS MYNAME, AGE AS MYAGE, LAST_NAME FROM
EMPLOYEE;

SELECT FIRST_NAME MYNAME, AGE, LAST_NAME FROM EMPLOYEE;

SELECT FIRST_NAME MYNAME, AGE MYAGE, LAST_NAME FROM
EMPLOYEE;



SELECT COUNT(*) FROM EMPLOYEE

SELECT COUNT(*) AS "RECORDS COUNT" FROM EMPLOYEE

SELECT COUNT(*)  "RECORDS COUNT" FROM EMPLOYEE

SELECT COUNT(AGE)  "RECORDS COUNT" FROM EMPLOYEE

SELECT COUNT(LAST_NAME)  "RECORDS COUNT" FROM EMPLOYEE


SELECT MAX(AGE) FROM EMPLOYEE;

SELECT MAX(AGE) AS "MAX AGE" FROM EMPLOYEE;

SELECT MAX(AGE) "MAX AGE" FROM EMPLOYEE;

SELECT MAX(AGE) "MAX AGE" FROM EMPLOYEE;

SELECT MIN(SALARY) FROM EMPLOYEE;

SELECT MIN(SALARY) MIN_SAL FROM EMPLOYEE;

SELECT MIN(SALARY) "MIN SAL" FROM EMPLOYEE;

SELECT AVG(SALARY) FROM EMPLOYEE;

SELECT AVG(SALARY) "AVG SALARY" FROM EMPLOYEE;

SELECT AVG(AGE) "AVG AGE" FROM EMPLOYEE;

SELECT MIN(FIRST_NAME) FROM EMPLOYEE;

SELECT MAX(FIRST_NAME) FROM EMPLOYEE;
SELECT * FROM EMPLOYEE ORDER BY FIRST_NAME;
SELECT * FROM EMPLOYEE ORDER BY AGE;
SELECT * FROM EMPLOYEE ORDER BY SALARY;
SELECT * FROM EMPLOYEE ORDER BY AGE, SALARY;

//SELECT FIRST_NAME, MAX(SALAY) FROM EMPLOYEE;

SELECT FIRST_NAME FROM EMPLOYEE WHERE SALARY =
(SELECT MAX(SALARY) FROM EMPLOYEE);

SELECT FIRST_NAME FROM EMPLOYEE WHERE AGE =
(SELECT MAX(AGE) FROM EMPLOYEE);

SELECT FIRST_NAME FROM EMPLOYEE WHERE AGE =
(SELECT MIN(AGE) FROM EMPLOYEE);

SELECT FIRST_NAME FROM EMPLOYEE WHERE SALARY <
(SELECT AVG(SALARY) FROM EMPLOYEE);



SELECT MAX(SALARY) FROM EMPLOYEE
WHERE SALARY < (SELECT MAX(SALARY) FROM EMPLOYEE);

SELECT MIN(SALARY) FROM EMPLOYEE
```

```
WHERE SALARY > (SELECT MIN(SALARY) FROM EMPLOYEE);

SELECT FIRST_NAME FROM EMPLOYEE WHERE SALARY =
(SELECT MAX(SALARY) FROM EMPLOYEE
WHERE SALARY < (SELECT MAX(SALARY) FROM EMPLOYEE));

SELECT * FROM EMPLOYEE WHERE SALARY =
(SELECT MIN(SALARY) FROM EMPLOYEE
WHERE SALARY > (SELECT MIN(SALARY) FROM EMPLOYEE));



SELECT ID, FIRST_NAME, LAST_NAME, AGE, SALARY, EMAIL,
RANK() OVER (ORDER BY SALARY DESC) FROM EMPLOYEE;



SELECT ID, FIRST_NAME, LAST_NAME, AGE, SALARY, EMAIL,
RANK() OVER (ORDER BY SALARY DESC) RANK FROM EMPLOYEE;

SELECT * FROM (SELECT ID, FIRST_NAME, LAST_NAME, AGE,
SALARY,
EMAIL, RANK() OVER(ORDER BY SALARY DESC) RANK FROM
EMPLOYEE)
WHERE RANK = 3;



SELECT * FROM (SELECT ID, FIRST_NAME, LAST_NAME, AGE,
SALARY,
EMAIL, RANK() OVER(ORDER BY SALARY DESC) RANK FROM
EMPLOYEE)

WHERE RANK = 5;


INSERT INTO EMPLOYEE VALUES(1, 'RAMU', 'BTM', 22, 4000,
'R@G.IN');

INSERT INTO EMPLOYEE VALUES(2, 'RAVI', 'RAO', 24, 3000,
'R@G.IN');

INSERT INTO EMPLOYEE VALUES(3, 'MANU', 'BTM', 23, 5000,
'M@G.IN');

INSERT INTO EMPLOYEE VALUES(4, 'MURALI', 'RAO', 26, 5500,
'M@G.IN');

INSERT INTO EMPLOYEE VALUES(5, 'RAGHU', 'JD', 25, 2000,
'R@G.IN');

INSERT INTO EMPLOYEE VALUES(6, 'PAVAN', 'NAO', 28, 3500,
'P@G.IN');

INSERT INTO EMPLOYEE VALUES(7, 'ANU', 'JP', 21, 6000,
'A@G.IN');

INSERT INTO EMPLOYEE VALUES(8, 'SWETHA', 'NAO', 29, 5600,
'W@G.IN');

INSERT INTO EMPLOYEE VALUES(9, 'SUNITHA', 'JD', 32, 7000,
'SU@G.IN');

INSERT INTO EMPLOYEE VALUES(10, 'KIRAN', 'JP', 30, 5300,
'K@G.IN');


INSERT INTO EMPLOYEE VALUES(1, 'RAMU', 'BTM', 22, 4000,
'R@G.IN');

INSERT INTO EMPLOYEE VALUES(2, 'RAVI', 'RAO', 24, 3000,
'R@G.IN');

INSERT INTO EMPLOYEE VALUES(3, 'MANU', 'BTM', 23, 5000,
'M@G.IN');
```

```
INSERT INTO EMPLOYEE VALUES(4, 'MURALI', 'RAO', 26, 5500,
'M@G.IN');
INSERT INTO EMPLOYEE VALUES(5, 'RAGHU', 'JD', 25, 2000,
'R@G.IN');
INSERT INTO EMPLOYEE VALUES(6, 'PAVAN', 'NAO', 28, 3500,
'P@G.IN');
INSERT INTO EMPLOYEE VALUES(7, 'ANU', 'JP', 21, 6000,
'A@G.IN');
INSERT INTO EMPLOYEE VALUES(8, 'SWETHA', 'NAO', 29, 5600,
'W@G.IN');
INSERT INTO EMPLOYEE VALUES(9, 'SUNITHA', 'JD', 32, 7000,
'SU@G.IN');
INSERT INTO EMPLOYEE VALUES(10, 'KIRAN', 'JP', 30, 5300,
'K@G.IN');

SELECT * FROM EMPLOYEE;
```

- In a page if I want to display only particular number of records even we have so many records in the table. We use pagination for that purpose.
- ROWNUM is the built-in Oracle, and this is not the RANK. Simply all the records are getting a number starting from digit 1.
- Through this ROWNUM we can achieve pagination.
- ROWNUM keyword we use for numbering rows.

```
SELECT ID, FIRST_NAME, LAST_NAME, AGE, EMAIL, SALARY,
ROWNUM FROM EMPLOYEE;

SELECT * FROM EMPLOYEE WHERE ROWNUM BETWEEN 10 AND 20;
```

- For the above query we don’t get the rows between 10 and 20 because we are using where clause but ROWNUM is a keyword not a column in the employee table.

- We don’t use ROWNUM for ordering based on particular column.

```
SELECT * FROM (SELECT ID, FIRST_NAME, LAST_NAME, AGE,
           EMAIL, SALARY, ROWNUM AS RN FROM EMPLOYEE)
           WHERE RN <= 10;
```

- In the above query we are using a temporary table. In this case we can get the ROWNUMS in between.

```
SELECT * FROM (SELECT ID, FIRST_NAME, LAST_NAME, AGE,
           EMAIL, SALARY, ROWNUM AS RN FROM EMPLOYEE)
           WHERE RN BETWEEN 11 AND 20;
```

- In the above query we are using a temporary table. In this case we can get the ROWNUMS in between.

```
SELECT * FROM (SELECT ID, FIRST_NAME, LAST_NAME, AGE,
           EMAIL, SALARY, ROWNUM AS RN FROM EMPLOYEE)
           WHERE RN BETWEEN 21 AND 30;
```

- In the above query we are using a temporary table. In this case we can get the ROWNUMS in between.

```
SELECT ID, FIRST_NAME, LAST_NAME, AGE, EMAIL, SALARY FROM
EMPLOYEE;
```

```
SELECT DISTINCT ID, FIRST_NAME, LAST_NAME, AGE, EMAIL,
SALARY FROM EMPLOYEE;
```

- If we want to get only unique records then we should use DISTINCT.
- If 2 records has same ID, FIRST_NAME, LAST_NAME, AGE, EMAIL, SALARY in the EMPLOYEE table then only one record will be selected.

```
SELECT * FROM EMPLOYEE GROUP BY
        ID, FIRST_NAME, LAST_NAME, AGE, EMAIL, SALARY;
```

- Whichever the records has same ID, FIRST_NAME, LAST_NAME, AGE, EMAIL, SALARY in the EMPLOYEE table are grouping into one.
- Totally 10 groups are creating, each group contains 3 records of same data.
- From every group only one record is displayed.
- DISTINCT and GROUP BY are similar.

```
SELECT ID, FIRST_NAME, LAST_NAME, AGE,
    EMAIL, SALARY, ROWID FROM EMPLOYEE;
```

- While inserting records in the Oracle database every record will be getting a unique ROWID

```
SELECT MIN(ROWID) FROM EMPLOYEE;
SELECT MAX(ROWID) FROM EMPLOYEE;
```

```
DELETE FROM EMPLOYEE WHERE ROWID NOT IN
(SELECT MIN(ROWID) FROM EMPLOYEE GROUP BY
        ID, FIRST_NAME, LAST_NAME, AGE, EMAIL, SALARY);
```

- Using GROUP BY we make the groups are similar records and display only one record from that group.
- In the above query we select a row from each group which has minimum ROWID and delete other two records of the group.
- Inner query selects the one row which has the minimum ROWID in the group and the outer query deletes other two records.

## Constraints

```
DROP TABLE TAB1;
CREATE TABLE TAB1(ID NUMBER, NAME VARCHAR2(90), AGE NUMBER);
INSERT INTO TAB1(ID) VALUES(1);
```

- By default column allows NULL values.

```
INSERT INTO TAB1(ID, NAME) VALUES(2, 'ABC');
INSERT INTO TAB1(ID, AGE) VALUES(3, 33);
INSERT INTO TAB1(NAME, AGE) VALUES('RAMU', 44);
INSERT INTO TAB1(NAME) VALUES('MANU');
INSERT INTO TAB1(AGE) VALUES(25);
SELECT * FROM TAB1;


DROP TABLE TAB2;
CREATE TABLE TAB2(ID NUMBER,
          NAME VARCHAR2(90) NOT NULL,
          AGE NUMBER);
```

- By using NOT NULL we can make sure that column are not having null values.
- In one table any number of columns can be NOT NULL.

```
INSERT INTO TAB2(ID) VALUES(1); //ERROR
INSERT INTO TAB2(ID, NAME) VALUES(2, 'ABC');
INSERT INTO TAB2(ID, AGE) VALUES(3, 33); //ERROR
INSERT INTO TAB2(NAME, AGE) VALUES('RAMU', 44);
INSERT INTO TAB2(NAME) VALUES('MANU');
INSERT INTO TAB2(AGE) VALUES(25);//ERROR
SELECT * FROM TAB2;

DROP TABLE TAB3;
CREATE TABLE TAB3(ID NUMBER NOT NULL,
          NAME VARCHAR2(90) NOT NULL,
          AGE NUMBER);
INSERT INTO TAB3(ID) VALUES(1); //ERROR
INSERT INTO TAB3(ID, NAME) VALUES(2, 'ABC');
INSERT INTO TAB3(ID, AGE) VALUES(3, 33); //ERROR
INSERT INTO TAB3(NAME, AGE) VALUES('RAMU', 44); //ERROR
INSERT INTO TAB3(NAME) VALUES('MANU'); //ERROR
INSERT INTO TAB3(AGE) VALUES(25);//ERROR
SELECT * FROM TAB3;

DROP TABLE TAB4;
CREATE TABLE TAB4(ID NUMBER,
          NAME VARCHAR2(90),
          AGE NUMBER);
INSERT INTO TAB4(ID, NAME) VALUES(1, 'MANU');
INSERT INTO TAB4(ID, NAME) VALUES(1, 'MANU');
INSERT INTO TAB4(ID, NAME, AGE) VALUES(2, 'RAMU', 22);
INSERT INTO TAB4(ID, NAME, AGE) VALUES(2, 'RAMU', 22);

INSERT INTO TAB4(NAME, AGE) VALUES('RAMU', 22);

INSERT INTO TAB4(NAME, AGE) VALUES('RAMU', 22);

DROP TABLE TAB5;

CREATE TABLE TAB5(ID NUMBER,

          NAME VARCHAR2(90) UNIQUE,

          AGE NUMBER);
```

- By default columns allow duplicate values.
- In one table any number of columns can be UNIQUE
- By using UNIQUE we can avoid duplicate values in the same column in the table.
- UNIQUE column allows any number of NULL values in a column but not duplicate values.

```
INSERT INTO TAB5(ID, NAME) VALUES(1, 'MANU');

INSERT INTO TAB5(ID, NAME) VALUES(1, 'MANU'); //ERROR

INSERT INTO TAB5(ID, NAME, AGE) VALUES(2, 'RAMU', 22);

INSERT INTO TAB5(ID, NAME, AGE) VALUES(2, 'RAMU',
22);//ERROR

INSERT INTO TAB5(NAME, AGE) VALUES('RAMU', 22);//ERROR

INSERT INTO TAB5(NAME, AGE) VALUES('RAMU', 22);//ERROR

INSERT INTO TAB5(ID, AGE) VALUES(5, 22);

INSERT INTO TAB5(ID, AGE) VALUES(5, 22);

INSERT INTO TAB5(ID) VALUES(5);

INSERT INTO TAB5(AGE) VALUES(25);

SELECT * FROM TAB5;



DROP TABLE TAB6;

CREATE TABLE TAB6(ID NUMBER,
          NAME VARCHAR2(90) UNIQUE,
          AGE NUMBER UNIQUE);
INSERT INTO TAB6(ID, NAME) VALUES(1, 'MANU');
INSERT INTO TAB6(ID, NAME) VALUES(1, 'MANU'); //ERROR
INSERT INTO TAB6(ID, NAME, AGE) VALUES(2, 'RAMU', 22);
INSERT INTO TAB6(ID, NAME, AGE) VALUES(2, 'RAMU',
22);//ERROR
INSERT INTO TAB6(NAME, AGE) VALUES('RAMU', 22);//ERROR
INSERT INTO TAB6(NAME, AGE) VALUES('RAMU', 22);//ERROR
INSERT INTO TAB6(ID, AGE) VALUES(5, 22); //ERROR
INSERT INTO TAB6(ID, AGE) VALUES(5, 22); //ERROR
INSERT INTO TAB6(ID) VALUES(5);
INSERT INTO TAB6(AGE) VALUES(25);
SELECT * FROM TAB6;


DROP TABLE TAB7;
CREATE TABLE TAB7(ID NUMBER,
          NAME VARCHAR2(90),
          AGE NUMBER,
CONSTRAINT TAB7_UK1 UNIQUE(NAME),
CONSTRAINT TAB7_UK2 UNIQUE(AGE))
```

- Syntax CONSTRAINT(declaration) TAB7_UK1(IDENTIFIER) UNIQUE(NAME)(type of the constraint and column name)
- Every constraints should be having unique identifier names in across the tables.
- We can disable or permanently drop the constraints. It is the better approach than previous.

```
INSERT INTO TAB7(ID, NAME) VALUES(1, 'MANU');
INSERT INTO TAB7(ID, NAME) VALUES(1, 'MANU'); //ERROR

INSERT INTO TAB7(ID, NAME, AGE) VALUES(2, 'RAMU', 22);

INSERT INTO TAB7(ID, NAME, AGE) VALUES(2, 'RAMU',
22);//ERROR

INSERT INTO TAB7(NAME, AGE) VALUES('RAMU', 22);//ERROR

INSERT INTO TAB7(NAME, AGE) VALUES('RAMU', 22);//ERROR

INSERT INTO TAB7(ID, AGE) VALUES(5, 22); //ERROR

INSERT INTO TAB7(ID, AGE) VALUES(5, 22); //ERROR

INSERT INTO TAB7(ID) VALUES(5);

INSERT INTO TAB7(AGE) VALUES(25);

SELECT * FROM TAB7;
```

```
DROP TABLE TAB8;

CREATE TABLE TAB8(ID NUMBER,
          NAME VARCHAR2(90),
          AGE NUMBER,
CONSTRAINT TAB8_UK1 UNIQUE(NAME, AGE));
```

- In the above constraint two records cant be having same values for the NAME and AGE columns.
- We can refer to it as a composite unique key.

```
INSERT INTO TAB8(ID, NAME) VALUES(1, 'MANU');

INSERT INTO TAB8(ID, NAME) VALUES(1, 'MANU'); //ERROR
```

- We get error because MANU, NULL and again MANU, NULL

```
INSERT INTO TAB8(ID, NAME, AGE) VALUES(2, 'RAMU', 22);

INSERT INTO TAB8(ID, NAME, AGE) VALUES(2, 'RAMU', 23);

INSERT INTO TAB8(ID, NAME, AGE) VALUES(2, 'AMU', 23);
INSERT INTO TAB8(ID, NAME, AGE) VALUES(2, 'RAMU', 
22);//ERROR

INSERT INTO TAB8(NAME, AGE) VALUES('RAMU', 22);//ERROR

INSERT INTO TAB8(NAME, AGE) VALUES('RAMU', 22);//ERROR

INSERT INTO TAB8(ID, AGE) VALUES(5, 22); //ERROR

INSERT INTO TAB8(ID, AGE) VALUES(5, 22); //ERROR

INSERT INTO TAB8(ID) VALUES(5);

INSERT INTO TAB8(ID) VALUES(5);

INSERT INTO TAB8(ID) VALUES(5);

INSERT INTO TAB8(ID) VALUES(5);
```

- The above queries possible because under UNIQUE constraint any number of NULL values and that wont be considered as a combination.

```
INSERT INTO TAB8(AGE) VALUES(25);

SELECT * FROM TAB8;
```

```
DROP TABLE TAB9;

CREATE TABLE TAB9(ID NUMBER,
          NAME VARCHAR2(90) PRIMARY KEY,
          AGE NUMBER);
```

- PRIMARY KEY is a combination of NOT NULL and UNIQUE.
- If any column is declared as PRIMARY KEY then that column value should not be NULL value and should not contain duplicate value.

```
INSERT INTO TAB9(ID, NAME) VALUES(1, 'MANU');

INSERT INTO TAB9(ID, NAME) VALUES(1, 'MANU'); //ERROR

INSERT INTO TAB9(ID, NAME, AGE) VALUES(2, 'RAMU', 22);
INSERT INTO TAB9(ID, NAME, AGE) VALUES(2, 'RAMU',
22);//ERROR

INSERT INTO TAB9(NAME, AGE) VALUES('RAMU', 22);//ERROR

INSERT INTO TAB9(NAME, AGE) VALUES('RAMU', 22);//ERROR

INSERT INTO TAB9(ID, AGE) VALUES(5, 22); //ERROR

INSERT INTO TAB9(ID, AGE) VALUES(5, 22); //ERROR

INSERT INTO TAB9(ID) VALUES(5); //ERROR

INSERT INTO TAB9(AGE) VALUES(25); //ERROR

INSERT INTO TAB9(NAME) VALUES('VIJAY');

SELECT * FROM TAB9;
```

```
CREATE TABLE TAB10(ID NUMBER,

          NAME VARCHAR2(90) PRIMARY KEY,

          AGE NUMBER PRIMARY KEY);
```

- In a table there should be only one column declared as PRIMARY KEY not more than one column.

```
CREATE TABLE TAB11(ID NUMBER,

          NAME VARCHAR2(90),

          AGE NUMBER,

CONSTRAINT TAB11_PK1 PRIMARY KEY(AGE));

INSERT INTO TAB11(ID, NAME) VALUES(1, 'MANU'); //ERROR

INSERT INTO TAB11(ID, NAME) VALUES(1, 'MANU'); //ERROR
INSERT INTO TAB11(ID, NAME, AGE) VALUES(2, 'RAMU', 22);

INSERT INTO TAB11(ID, NAME, AGE) VALUES(2, 'RAMU',
22);//ERROR

INSERT INTO TAB11(NAME, AGE) VALUES('RAMU', 22);//ERROR

INSERT INTO TAB11(NAME, AGE) VALUES('RAMU', 28);

INSERT INTO TAB11(ID, AGE) VALUES(5, 22); //ERROR

INSERT INTO TAB11(ID, AGE) VALUES(5, 28); //ERROR

INSERT INTO TAB11(ID) VALUES(5); //ERROR

INSERT INTO TAB11(AGE) VALUES(25);

INSERT INTO TAB11(NAME) VALUES('VIJAY'); //ERROR

SELECT * FROM TAB11;


CREATE TABLE TAB12(ID NUMBER,
          NAME VARCHAR2(90),
          AGE NUMBER,
CONSTRAINT TAB12_PK1 PRIMARY KEY(AGE, NAME));
```

- Composite PRIMARY key is possible.

```
INSERT INTO TAB12 VALUES(1, 'ABC', 22);

INSERT INTO TAB12 VALUES(2, 'ABC', 22);//ERROR

INSERT INTO TAB12 VALUES(3, 'ABC', 23);

INSERT INTO TAB12 VALUES(4, 'ABC1', 23);

INSERT INTO TAB12(ID, NAME) VALUES(5, 'XYZ'); //ERROR

INSERT INTO TAB12(ID, AGE) VALUES(6, 33); //ERROR
```

## FOREIGN KEY

```
CREATE TABLE STUDENT(ID NUMBER UNIQUE,
             FIRST_NAME VARCHAR2(90),
             LAST_NAME VARCHAR2(90),
             AGE NUMBER,
             EMAIL VARCHAR2(90));

CREATE TABLE ADDRESS(HOUSE_NO VARCHAR2(90),
             STREET_NAME VARCHAR2(90),
             CITY VARCHAR2(90),
             STATE VARCHAR2(90),
             STUDENT_ID NUMBER,
CONSTRAINT ADDRESS_FK1 FOREIGN KEY(STUDENT_ID)
                REFERENCES STUDENT(ID));
```

- For FOREIGN KEY purpose REFERENCES table column either UNIQUE or PRIMARY

```
INSERT INTO STUDENT VALUES(1, 'RAMU', 'B', 22, 'R@R.COM');
INSERT INTO ADDRESS VALUES('123/B', 'BTM', 'BLR', 'KAR',
1);

INSERT INTO STUDENT VALUES(2, 'MANU', 'B', 24, 'M@R.COM');
INSERT INTO ADDRESS VALUES('123/C', 'JPN', 'BLR', 'KAR',
2);

INSERT INTO STUDENT VALUES(3, 'VIJAY', 'B', 26, 'V@R.COM');
INSERT INTO ADDRESS VALUES('123/V', 'JN', 'BLR', 'KAR', 3);
INSERT INTO ADDRESS VALUES('123/R', 'JN', 'BLR', 'KAR',
4);//ERROR
```

- Trying to insert child record straight away without parent.
- FOREIGN KEY should have reference value of the column from the parent.

```
INSERT INTO STUDENT VALUES(4, 'MURALI', 'B', 26,
'M@R.COM');

INSERT INTO ADDRESS VALUES('123/V', 'JN', 'BLR', 'KAR', 4);
```

```
DELETE FROM STUDENT WHERE ID = 4; //ERROR
```

- STUDENT ID = 4 having child in the ADDRESS.
- You cant delete parent record without deleting a child record.

```
DELETE FROM ADDRESS WHERE STUDENT_ID = 4;

DELETE FROM STUDENT WHERE ID = 4;
```

```
DROP TABLE STUDENT;  //ERROR
```

- Straight away we cant delete STUDENT table.
- STUDENT table is a parent to ADDRESS table
- Without deleting the child we cant able to delete PARENT

```
DELETE FROM ADDRESS;

DROP TABLE STUDENT;  //ERROR
```

- Even though ADDRESS table is empty we cant able to drop STUDENT table.

```
DROP TABLE ADDRESS;

DROP TABLE STUDENT;
```

- First we need to drop ADDRESS table then only we can able to drop the STUDENT table.

```
CREATE TABLE STUDENT(ID NUMBER,
             FIRST_NAME VARCHAR2(90),
             LAST_NAME VARCHAR2(90),
             AGE NUMBER,
             EMAIL VARCHAR2(90));

CREATE TABLE ADDRESS(HOUSE_NO VARCHAR2(90),
             STREET_NAME VARCHAR2(90),
             CITY VARCHAR2(90),
             STATE VARCHAR2(90),
             STUDENT_ID NUMBER,
CONSTRAINT ADDRESS_FK1 FOREIGN KEY(STUDENT_ID)
                REFERENCES STUDENT(ID)); //ERROR

DROP TABLE STUDENT;
```

```
CREATE TABLE STUDENT(ID NUMBER UNIQUE,
             FIRST_NAME VARCHAR2(90),
             LAST_NAME VARCHAR2(90),
             AGE NUMBER,
             EMAIL VARCHAR2(90));

CREATE TABLE ADDRESS(HOUSE_NO VARCHAR2(90),
             STREET_NAME VARCHAR2(90),
             CITY VARCHAR2(90),
             STATE VARCHAR2(90),
             STUDENT_ID NUMBER,
CONSTRAINT ADDRESS_FK1 FOREIGN KEY(STUDENT_ID)
                REFERENCES STUDENT(ID));
```

```
INSERT INTO STUDENT VALUES(1, 'RAMU', 'B', 22,
'R@GMAIL.COM');
INSERT INTO ADDRESS(HOUSE_NO, STREET_NAME, CITY, STATE)
        VALUES('123/T', 'BTM', 'BLR', 'KAR');
```

- we can have NULL values for FOREIGN KEY REFERENCE.
- If STUDENT_ID is not PRIMARY KEY in the ADDRESS we can have NULL values.
- We are inserting an ADDRESS which doesn’t belong to any STUDENT.
- By default FOREIGN KEY allows NULL values.

-

```
UPDATE ADDRESS SET STUDENT_ID = 3 WHERE HOUSE_NO = '123/T';
//ERROR
```

- There is no corresponding record.

```
UPDATE ADDRESS SET STUDENT_ID = 1 WHERE HOUSE_NO = '123/T';


INSERT INTO STUDENT(FIRST_NAME, LAST_NAME, AGE, EMAIL)
        VALUES('MANU', 'B', 22, 'R@GMAIL.COM');
INSERT INTO ADDRESS(HOUSE_NO, STREET_NAME, CITY, STATE)
        VALUES('123/Y', 'BTM', 'BLR', 'KAR');
```

- In the base table ID column is NULL and in the child table STUDENT_ID is also NULL.
- Here the created ADDRESS record is not belong to created STUDENT record.
- NULL cant be assigned to another NULL.
- NULL cant be mapped to another NULL.

```
UPDATE STUDENT SET STUDENT_ID = 2 WHERE FIRST_NAME =
'MANU';
UPDATE ADDRESS SET STUDENT_ID = 2 WHERE HOUSE_NO = '123/Y';
```

**ONE-TO-ONE**

```
DROP TABLE PERSON;
CREATE TABLE PERSON (ID NUMBER PRIMARY KEY,
             FIRST_NAME VARCHAR2(90),
             LAST_NAME VARCHAR2(90),
             AGE NUMBER);
```

- Because of ID column is PRIMARY KEY PERSON table can become a parent to child table.

```
DROP TABLE ADDRESS;
CREATE TABLE ADDRESS(HOUSE_NO VARCHAR2(90),
                     STREET_NAME VARCHAR2(90),
             CITY VARCHAR2(90),
             STATE VARCHAR2(90),
             PERSON_ID NUMBER UNIQUE,
CONSTRAINT ADDRESS_FK1 FOREIGN KEY(PERSON_ID)
            REFERENCES PERSON(ID));



INSERT INTO PERSON VALUES(1, 'RAMU', 'ABC', 22);
INSERT INTO PERSON VALUES(2, 'MANU', 'XYZ', 24);
INSERT INTO PERSON VALUES(3, 'SWETHA', 'TEST', 21);
INSERT INTO PERSON VALUES(4, 'KUMAR', 'BLR', 23);
INSERT INTO PERSON VALUES(5, 'MURALI', 'BLR', 23);

INSERT INTO ADDRESS VALUES('123/B', 'BTM', 'BLR', 'KAR',
1);
```

```
INSERT INTO ADDRESS VALUES('123/C', 'BTM', 'BLR', 'KAR', 1);//ERROR
```

- FOREIGN KEY is a UNIQUE so we cant insert duplicates.
- One record of PERSON mapping to only one record of ADDRESS. So we call it as one to one mapping.

```
INSERT INTO ADDRESS VALUES('123/D', 'BTM', 'BLR', 'KAR', 2);

INSERT INTO ADDRESS VALUES('123/E', 'BTM', 'BLR', 'KAR', 3);

INSERT INTO ADDRESS VALUES('123/F', 'BTM', 'BLR', 'KAR', 4);

INSERT INTO ADDRESS VALUES('123/G', 'BTM', 'BLR', 'KAR', 6); //ERROR
```

- No PERSON with ID as 6

```
INSERT INTO ADDRESS(HOUSE_NO, STREET_NAME, CITY, STATE)
        VALUES('123/G', 'BTM', 'BLR', 'KAR');
```

- We can insert ADDRESS without choosing PERSON_ID because this column is UNIQUE and allows NULL values.

```
SELECT * FROM PERSON;

SELECT * FROM ADDRESS;



SELECT * FROM PERSON WHERE FIRST_NAME = 'RAMU';

SELECT * FROM ADDRESS WHERE HOUSE_NO = '123/D';


SELECT * FROM ADDRESS WHERE PERSON_ID = 3;
SELECT * FROM ADDRESS WHERE PERSON_ID =
(SELECT ID FROM PERSON WHERE FIRST_NAME = 'RAMU');

SELECT * FROM PERSON WHERE ID =
(SELECT PERSON_ID FROM ADDRESS WHERE HOUSE_NO = '123/E');



SELECT * FROM PERSON, ADDRESS WHERE PERSON.ID =
ADDRESS.PERSON_ID;
SELECT * FROM PERSON P, ADDRESS A WHERE P.ID = A.PERSON_ID;


SELECT * FROM PERSON P INNER JOIN ADDRESS A ON P.ID =
A.PERSON_ID;
SELECT * FROM PERSON P LEFT OUTER JOIN ADDRESS A ON P.ID =
A.PERSON_ID;
SELECT * FROM PERSON P RIGHT OUTER JOIN ADDRESS A ON P.ID =
A.PERSON_ID;
SELECT * FROM PERSON P FULL OUTER JOIN ADDRESS A ON P.ID =
A.PERSON_ID;
```

**ONE-TO-MANY**

```
DROP TABLE PERSON CASCADE CONSTRAINTS;
```

- Even though child table is there we are trying to remove parent table forcefully.
- Only the foreign key constraint will be removed not the column itself.

```
CREATE TABLE PERSON (ID NUMBER PRIMARY KEY,
             FIRST_NAME VARCHAR2(90),
             LAST_NAME VARCHAR2(90),
             AGE NUMBER);


DROP TABLE MAIL_ACCOUNT;
CREATE TABLE MAIL_ACCOUNT(USERNAME VARCHAR2(90),
                     PASSWORD VARCHAR2(90),
             PROVIDER VARCHAR2(90),
             PERSON_ID NUMBER,
CONSTRAINT MK_FK1 FOREIGN KEY(PERSON_ID)
            REFERENCES PERSON(ID));


INSERT INTO PERSON VALUES(1, 'RAMU', 'ABC', 22);
INSERT INTO PERSON VALUES(2, 'MANU', 'XYZ', 24);
INSERT INTO PERSON VALUES(3, 'SWETHA', 'TEST', 21);
INSERT INTO PERSON VALUES(4, 'KUMAR', 'BLR', 23);
INSERT INTO PERSON VALUES(5, 'MURALI', 'BLR', 23);


INSERT INTO MAIL_ACCOUNT VALUES('RAMU', 'XYZ', 'GMAIL', 1);
INSERT INTO MAIL_ACCOUNT VALUES('RAMU', 'XYZ', 'GMAIL', 1);
INSERT INTO MAIL_ACCOUNT VALUES('RAMU', 'XYZ', 'HOTMAIL',
1);

INSERT INTO MAIL_ACCOUNT VALUES('MANU', 'XYZ', 'HOTMAIL',
2);

INSERT INTO MAIL_ACCOUNT VALUES('MANU', 'XYZ', 'GMAIL', 2);

INSERT INTO MAIL_ACCOUNT VALUES('SWETHA', 'XYZ', 'GMAIL',
3);

INSERT INTO MAIL_ACCOUNT VALUES('KUMAR', 'XYZ', 'GMAIL',
4);

INSERT INTO MAIL_ACCOUNT(USERNAME, PASSWORD, PROVIDER)
    VALUES('USER1', 'XYZ', 'GMAIL');


SELECT * FROM PERSON;

SELECT * FROM MAIL_ACCOUNT;

SELECT * FROM PERSON WHERE FIRST_NAME = 'RAMU';

SELECT * FROM MAIL_ACCOUNT WHERE USERNAME = 'RAMU1';


SELECT * FROM MAIL_ACCOUNT WHERE PERSON_ID = 3;


SELECT * FROM MAIL_ACCOUNT WHERE PERSON_ID =
(SELECT ID FROM PERSON WHERE FIRST_NAME = 'RAMU');


SELECT * FROM PERSON WHERE ID =
(SELECT PERSON_ID FROM MAIL_ACCOUNT WHERE USERNAME =
'RAMU');


SELECT * FROM PERSON, MAIL_ACCOUNT WHERE PERSON.ID =
MAIL_ACCOUNT.PERSON_ID;
```

```
SELECT * FROM PERSON P, MAIL_ACCOUNT M WHERE P.ID =
M.PERSON_ID;


SELECT * FROM PERSON P INNER JOIN MAIL_ACCOUNT M ON P.ID =
M.PERSON_ID;

SELECT * FROM PERSON P LEFT OUTER JOIN MAIL_ACCOUNT M ON
P.ID = M.PERSON_ID;

SELECT * FROM PERSON P RIGHT OUTER JOIN MAIL_ACCOUNT M ON
P.ID = M.PERSON_ID;

SELECT * FROM PERSON P FULL OUTER JOIN MAIL_ACCOUNT M ON
P.ID = M.PERSON_ID;
```

## MANY-TO-MANY

```
DROP TABLE STUDENT CASCADE CONSTRAINTS;

CREATE TABLE STUDENT(ID NUMBER UNIQUE,
             FIRST_NAME VARCHAR2(90),
             LAST_NAME VARCHAR2(90));

DROP TABLE SKILL CASCADE CONSTRAINT;

CREATE TABLE SKILL(ID NUMBER UNIQUE, NAME VARCHAR2(90));

INSERT INTO STUDENT VALUES(1, 'RAMU', 'B');

INSERT INTO STUDENT VALUES(2, 'MANU', 'C');

INSERT INTO STUDENT VALUES(3, 'MURALI', 'D');

INSERT INTO STUDENT VALUES(4, 'KUMAR', 'E');

INSERT INTO SKILL VALUES(1, 'C');

INSERT INTO SKILL VALUES(2, 'C++');

INSERT INTO SKILL VALUES(3, 'JAVA');

INSERT INTO SKILL VALUES(4, 'ORACLE');
```

```
DROP TABLE STUDENT_SKILL CASCADE CONSTRAINTS;

CREATE TABLE STUDENT_SKILL(STUDENT_ID NUMBER, SKILL_ID
NUMBER,

CONSTRAINT SS_FK1 FOREIGN KEY(STUDENT_ID) REFERENCES
STUDENT(ID),

CONSTRAINT SS_FK2 FOREIGN KEY(SKILL_ID) REFERENCES
SKILL(ID));

INSERT INTO STUDENT_SKILL VALUES(1, 1);

INSERT INTO STUDENT_SKILL VALUES(1, 2);

INSERT INTO STUDENT_SKILL VALUES(2, 2);

INSERT INTO STUDENT_SKILL VALUES(3, 1);

INSERT INTO STUDENT_SKILL VALUES(3, 4);

SELECT * FROM STUDENT;

SELECT * FROM SKILL;

SELECT * FROM STUDENT_SKILL;


SELECT NAME FROM SKILL WHERE ID IN

(SELECT SKILL_ID FROM STUDENT_SKILL WHERE STUDENT_ID =

(SELECT ID FROM STUDENT WHERE FIRST_NAME = 'VIJAY'));


SELECT FIRST_NAME FROM STUDENT WHERE ID IN

(SELECT STUDENT_ID FROM STUDENT_SKILL WHERE SKILL_ID =

(SELECT ID FROM SKILL WHERE NAME = 'C'));



SELECT * FROM STUDENT, STUDENT_SKILL, SKILL WHERE

    STUDENT.ID = STUDENT_SKILL.STUDENT_ID AND

    STUDENT_SKILL.SKILL_ID = SKILL.ID;
```

```
SELECT * FROM STUDENT S INNER JOIN STUDENT_SKILL SS
ON S.ID = SS.STUDENT_ID INNER JOIN SKILL SK ON SS.SKILL_ID
= SK.ID;
```