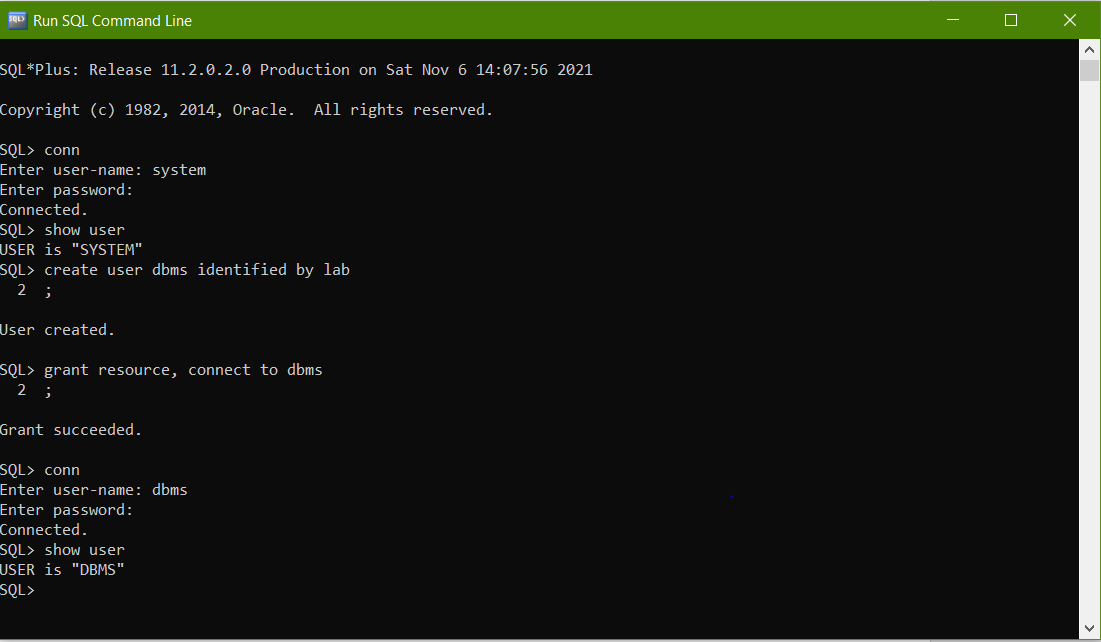
**Software Used :** SQL\*Plus: Release 11.2.0.2.0 Production

Before running any query on sql command line, we need to locally connect sql/oracle to the same computer which we are using.

Connecting locally means running SQL Command Line (SQL\*Plus) and Oracle Database XE on the same computer. There are two ways to start a local connection with SQL Command Line:

* From the desktop
* From a terminal session (Linux) or command window (Windows)

Note : We also created new user, the database user is the identity of the login when it is connected to a database.



**PROGRAMS**

1. **Creating Database Tables**

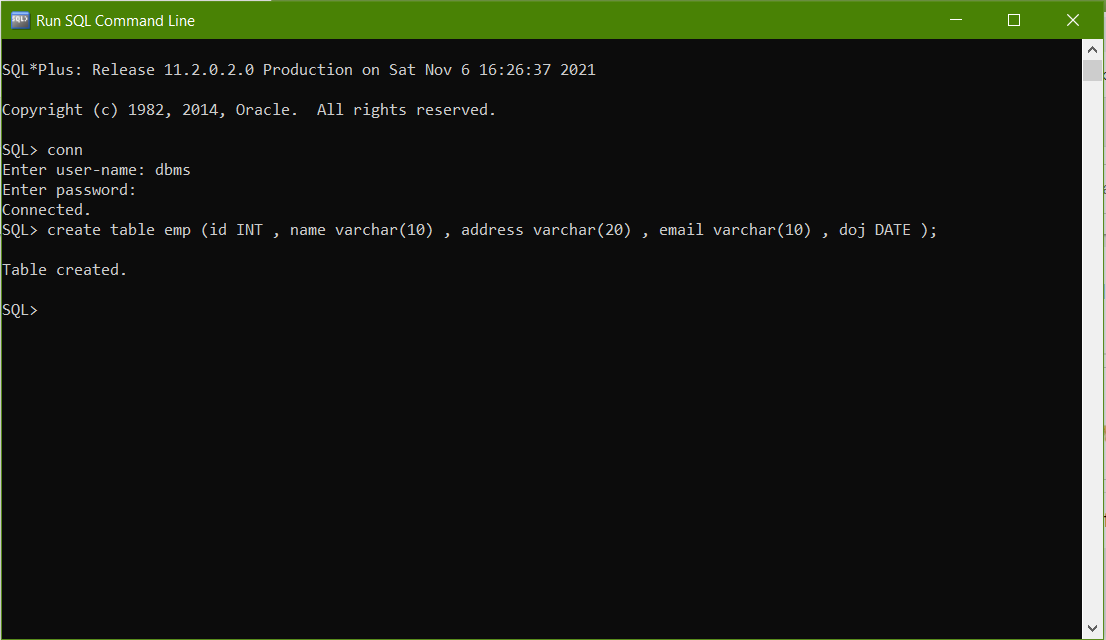
**1.1 Create table :**

**create** is a DDL SQL command. Create table is used to create a new table in the database.

**Syntax :**

CREATE TABLE table\_name (  
    column1 datatype,  
    column2 datatype,  
    …  
);

**Example :**



* 1. **Modify Table :**

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

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

**Syntax :**

ALTER TABLE table\_name  
ADD column\_name datatype;



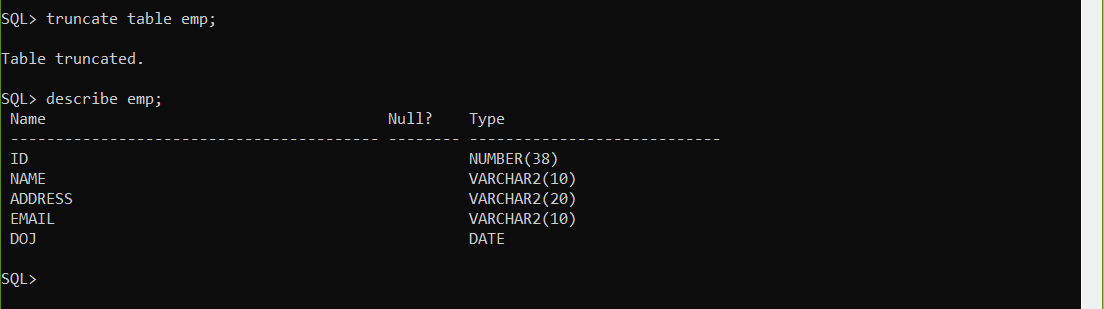
* 1. **Drop table :** The DROP table is used to drop an existing table in the database.

**Syntax :** DROP TABLE table\_name;



* 1. **Truncate :** The SQL **TRUNCATE TABLE** command is used to delete complete data from an existing table.

**Syntax :** TRUNCATE TABLEemp;



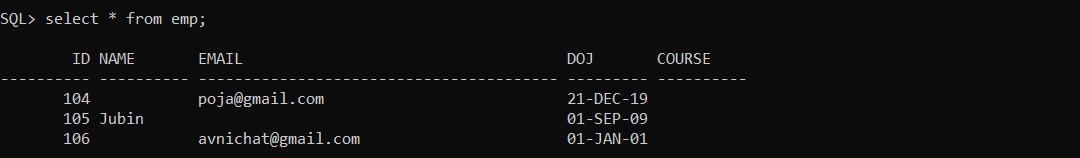
1. **Practical’s for retrieving Data Using following clauses**
   1. Simple SELECT clause

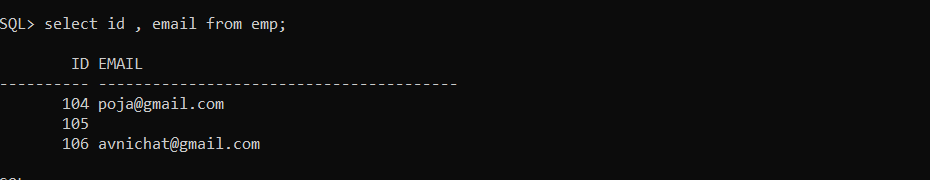
The SELECT statement is used to select data from a database.

**Syntax :**

* SELECT \* FROM table\_name; (To select all fields)
* SELECT column1, column2, ...  
  FROM table\_name;

**Example :**





2.2Accessing specific data with WHERE , Exist, Not Exist

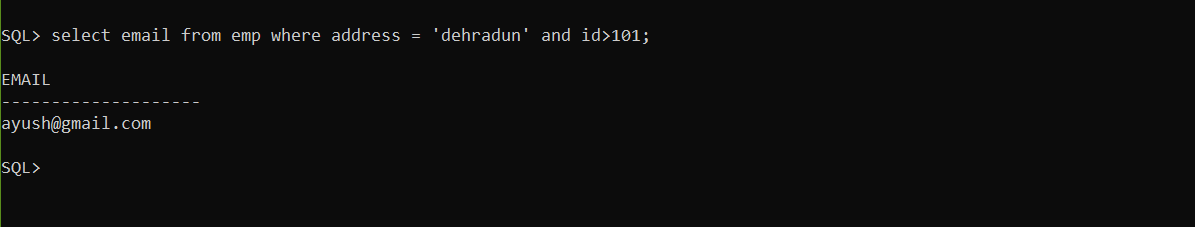
* The WHERE clause is used to filter records.

**Syntax :**

SELECT column1, column2, ...  
FROM table\_name  
WHERE condition;

*(to apply conditions based on records we need to add records to the table first)*

Example : select email from emp where address = 'dehradun' and id>101;



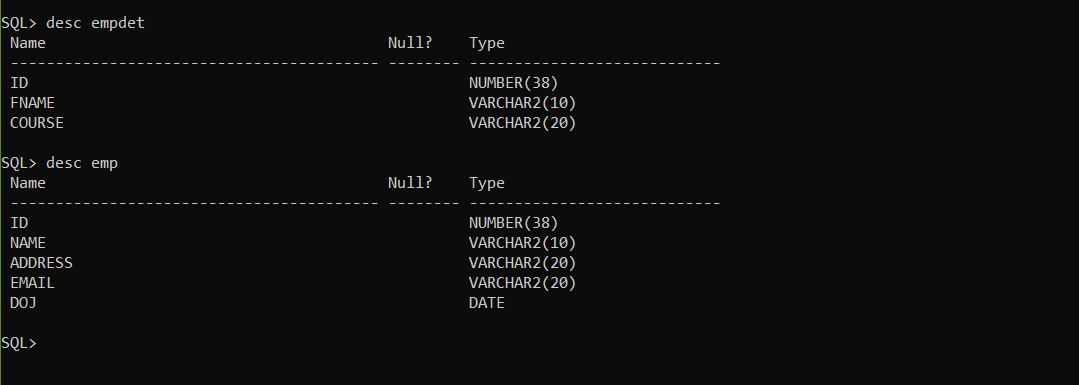
* The EXISTS operator is used to test for the existence of any record in a subquery.

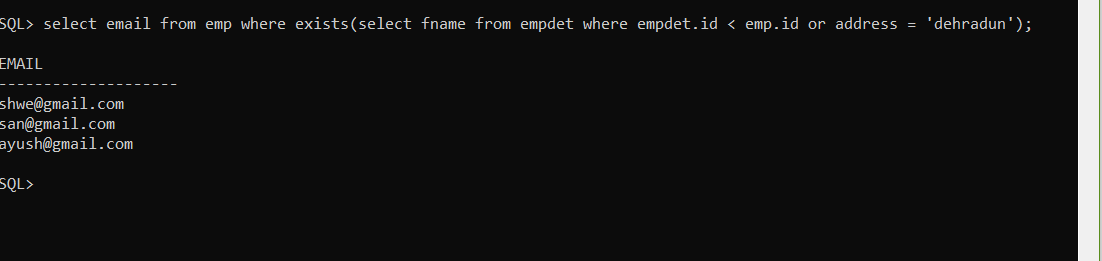
The EXISTS operator returns TRUE if the subquery returns one or more records.

**Syntax :**

SELECT column\_name(s)  
FROM table\_name  
WHERE EXISTS  
(SELECT column\_name FROM table\_name WHERE condition);

*Note : I created another new table named ‘empdet’ and used that table in the EXISTS command query.*





* **NOT EXISTS**

The NOT EXISTS in SQL Server will check the Subquery for rows existence, and if there are no rows then it will return TRUE, otherwise FALSE.

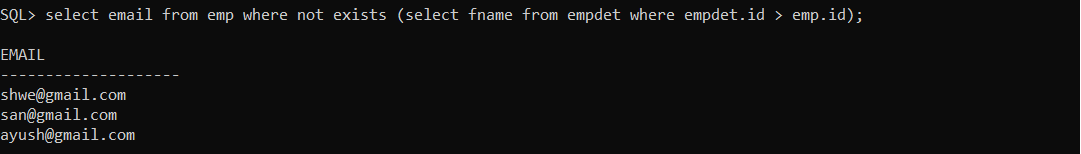
**Syntax :**

SELECT [Column Names]

FROM [Source]

WHERE NOT EXISTS (Write Subquery to Check)

Example : here condition applied is false but NOT EXISTS return true if there is no rows selected. (therefore showed rows instead no rows selected)



• **Ordered By, Distinct clause**

The ORDER BY keyword is used to sort the result-set in ascending or descending order. (default asc)

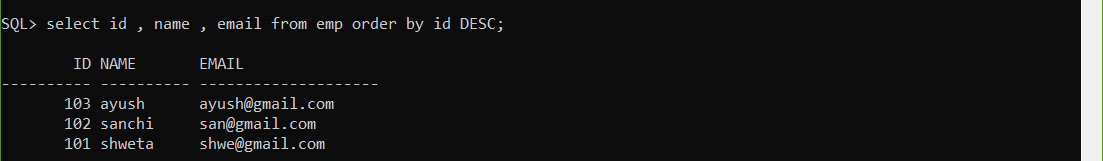
**Syntax :**

SELECT column1, column2, ...  
FROM table\_name  
ORDER BY column1, column2, ... ASC|DESC;

In Ascending order (by default)



In Descending order



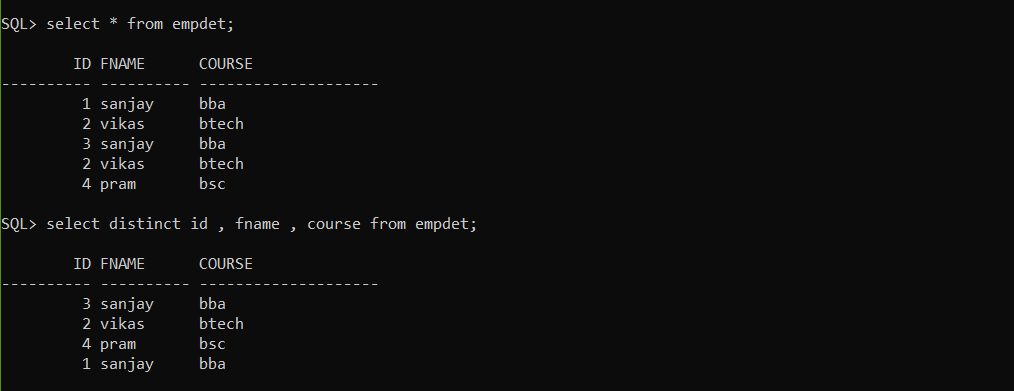
* Distinct Clause

The SELECT DISTINCT statement is used to return only distinct (different) values.

**Syntax :**

SELECT DISTINCT column1, column2, ...  
FROM table\_name;

Example :



• **Operators like IN, BETWEEN, LIKE, IS NULL, NOT NULL etc**

**IN Operator :**

The IN operator allows you to specify multiple values in a WHERE clause.

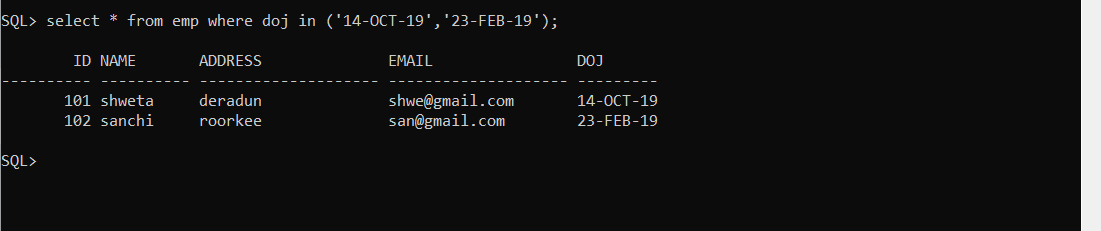
The IN operator is a shorthand for multiple OR conditions.

**Syntax :**

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_name IN (value1, value2, ...);

Note : You can also replace values by select statement

Example : SQL> select \* from emp where doj in ('14-OCT-19','23-FEB-19');



**BETWEEN Operator:**

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates. (Inclusive)

**Syntax :**

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_name BETWEEN value1 AND value2;

****

**Like Operator :**

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

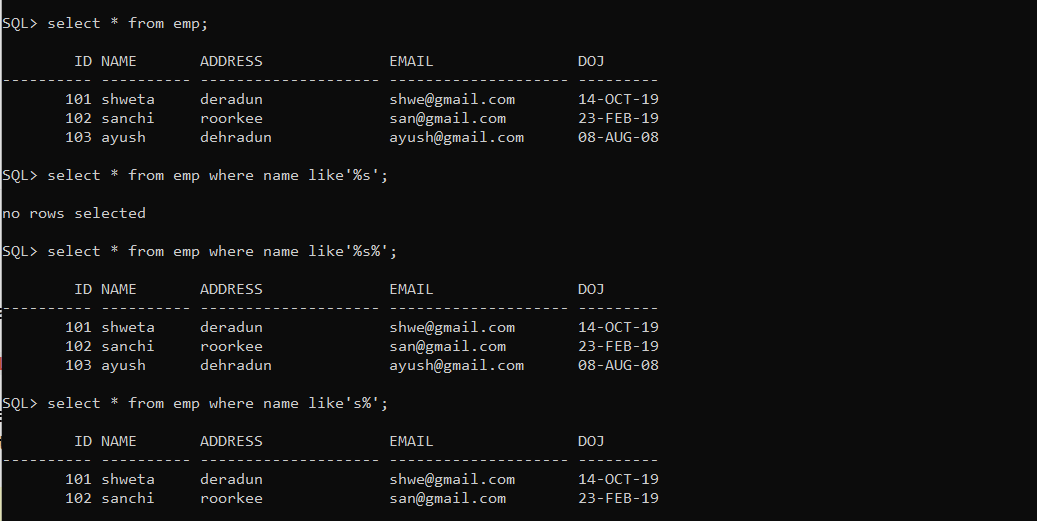
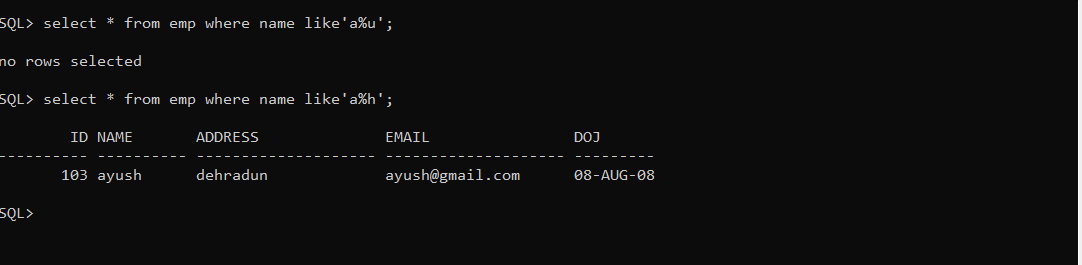
* The percent sign (%) represents zero, one, or multiple characters
* The underscore sign (\_) represents one, single character

**Syntax :**

SELECT column1, column2, ...  
FROM table\_name  
WHERE column LIKE pattern;

Example :

Demonstrated all different examples using ‘%’ and ‘\_’ whilecards :

********

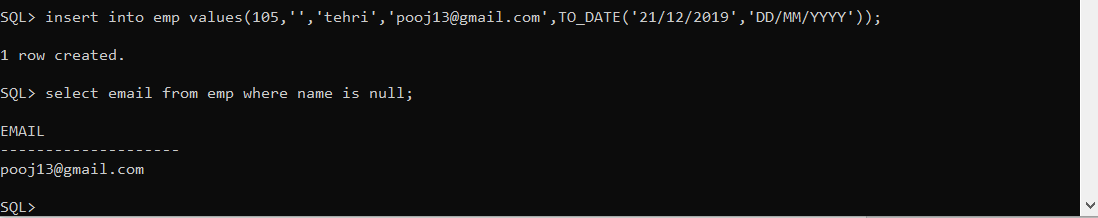
**Is Null Operator :**

The IS NULL operator is used to test for empty values (NULL values).

**Syntax :**

SELECT CustomerName, ContactName, Address  
FROM Customers  
WHERE Address IS NULL;

Example :

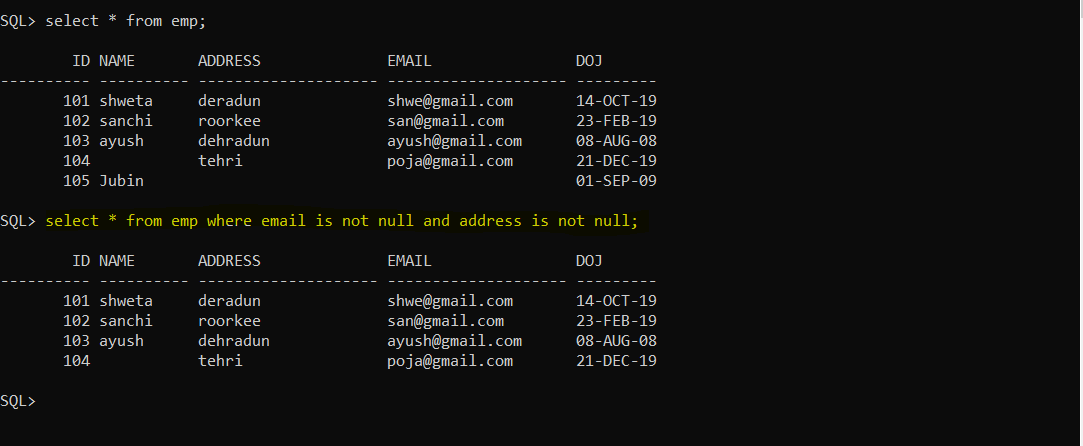
****

**Is Not Null Operator :**

Contradicts Is Null Operator, the IS NULL operator is used to test for empty values (NULL values).

SELECT column\_namesFROM table\_name  
WHERE column\_name IS NOT NULL;

**Example :**



1. **Practical’s based on Data manipulation**

**• Adding data with Insert**

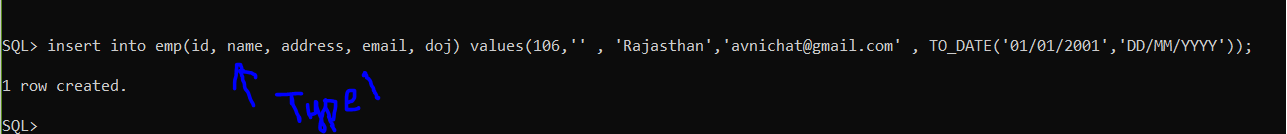
The SQL **INSERT INTO** Statement is used to add new rows of data to a table in the database.

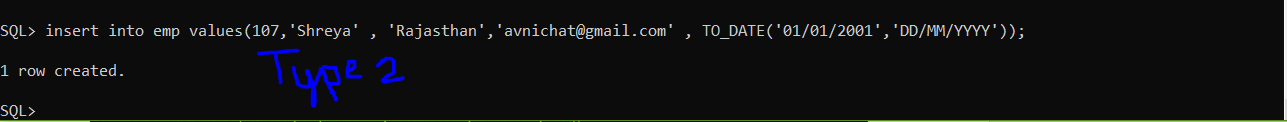
**Syntax : (3 ways)**

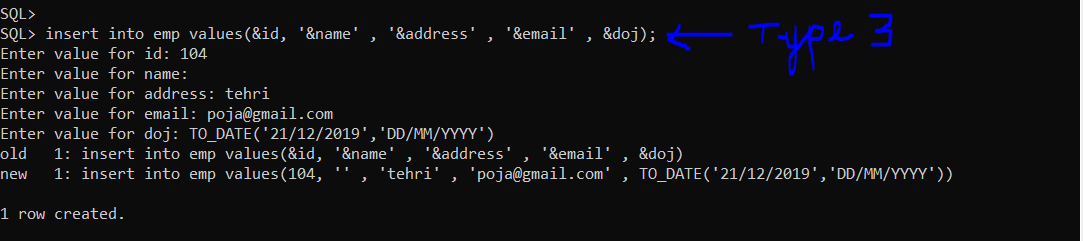
* Insert into table\_name (column1 , column2 , …) values (value1 , value2, value3 , …);
* Insert into table\_name values (value1 , value2, value3 , …);
* Insert into table\_name (&column1 , &column2 , …);

*(write ‘&column’ within quotes if of varchar type)*

**Example :**







**• Modify Data with Update**

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

**Syntax :**

UPDATE table\_name  
SET column1 = value1, column2 = value2, ...  
WHERE condition;

**Example :**



**• Deleting records with Delete**

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

**Syntax :**

DELETE FROM table\_name WHERE condition;

**Example :**



**• Alter Table Command**

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

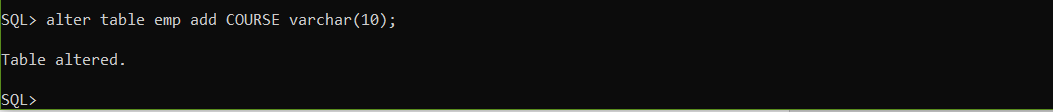
1. **ALTER TABLE ADD COLUMN**

To add a column in a table

**Syntax :**

ALTER TABLE table\_name  
ADD column\_name datatype;

**Example :**



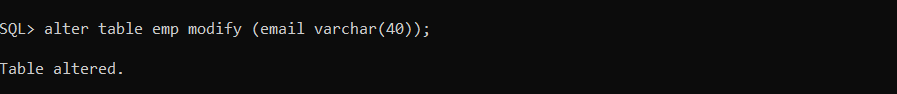
1. **ALTER TABLE TO MODIFY(ALTER) COLUMN**

To change the data type of a column in a table.

**Syntax :**

ALTER TABLE table\_name  
MODIFY ( column\_name datatype(size(optional));

**Example :**



1. **ALTER TABLE DROP COLUMN**

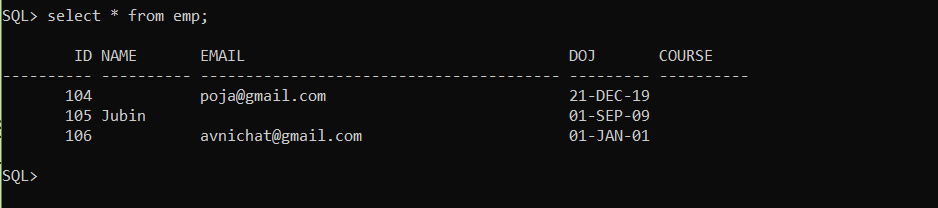
To drop the table permanently from the database (with records and structure both).

**Syntax :**

ALTER TABLE table\_name  
DROP COLUMN column\_name;

**Example :**





1. **Practical’s based on Implementing the Constraints**

*(ADD CONSTRAINT / CREATE NEW TABLE AND ADD CONSTRAINT)*

**• NULL & NOT NULL**

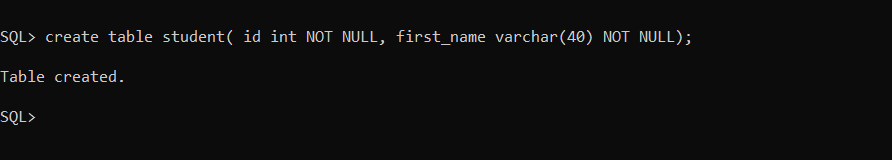
By default, a column can hold NULL values.

The NOT NULL constraint enforces a column to NOT accept NULL values.

**Syntax :**

CREATE TABLE Table\_name (  
    column1 datatype(size) NOT NULL,  
    column2 datatype(size) NOT NULL,  
    column3 datatype(size) NOT NULL,  
    …  
);

**Example :**

****

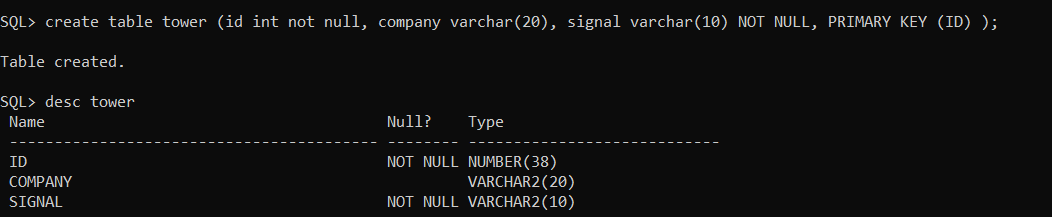
**• PRIMARY KEY Constraint**

A  PRIMARY KEY constraint declares a column or a combination of columns whose values uniquely identify each row in a table.

**Syntax :**

CREATE TABLE Table\_name (  
    column1 int NOT NULL,  
    column2 varchar(255) NOT NULL,  
    column3 varchar(255),  
    column4 int,  
    PRIMARY KEY (ID)  
);

**Example:**

****

**• FOREIGN KEY Constraint**

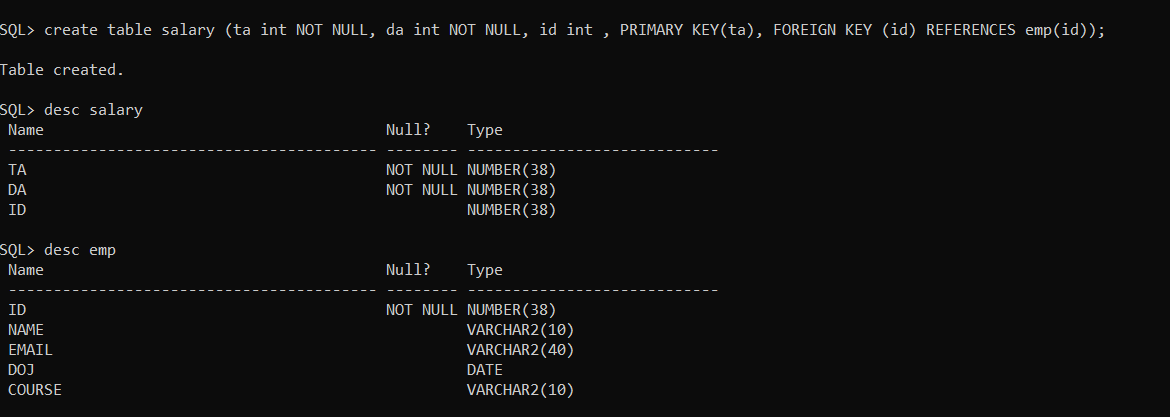
A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the [PRIMARY KEY](https://www.w3schools.com/sql/sql_primarykey.asp) in another table. The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

**Syntax :**

CREATE TABLE table1\_name (  
    column1 int NOT NULL,  
    column2 int NOT NULL,  
    column3 int,  
    PRIMARY KEY (column1),  
    );

CREATE TABLE table2\_name (  
    column1 int NOT NULL,  
    column2 int NOT NULL,  
    column3 int,  
    PRIMARY KEY (column1),  
    FOREIGN KEY (column3)  
    REFERENCES table1\_name(column3)  
);

Example :



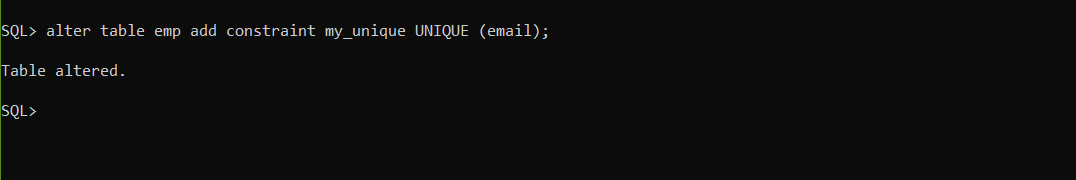
**• UNIQUE KEY Constraint**

The UNIQUE constraint ensures that all values in a column are different.

**Syntax:**

ALTER TABLE table\_name ADD CONSTRAINT my\_unique UNIQUE(col1,col2…);

**Example :**



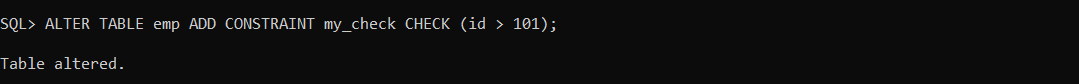
**• CHECK Constraint**

The CHECK constraint is used to limit the value range that can be placed in a column.

**Syntax :**

ALTER TABLE table\_name ADD CONSTRAINT my\_check CHECK (condition);

**Example :**



**• DEFAULT Constraint**

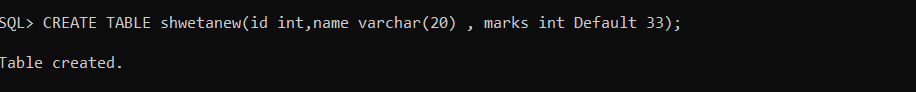
The DEFAULT constraint is used to set a default value for a column.

**Syntax :**

*(if creating new table, you can also add constraint using alter add command in the existing table)*

CREATE TABLE table\_name (col1 datatype(size) , col2 datatype(size) Default default\_value,…);

**Example :**



1. **Practical’s based on Aggregate & Mathematical Functions**

(Do Using Group By, Having Clause Also)

Note : you may or may not add WHERE Clause.

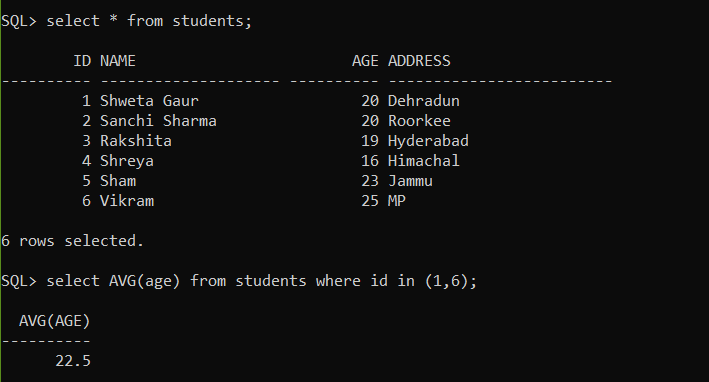
• AVG

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

**Syntax :**

SELECT AVG(column\_name)  
FROM table\_name  
WHERE condition;

**Example :**



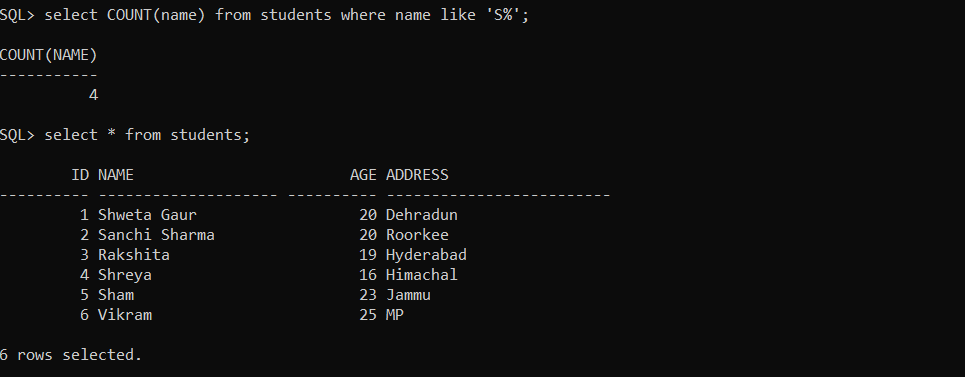
• COUNT

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

**Syntax :**

SELECT COUNT(column\_name)  
FROM table\_name  
WHERE condition;

**Example :**



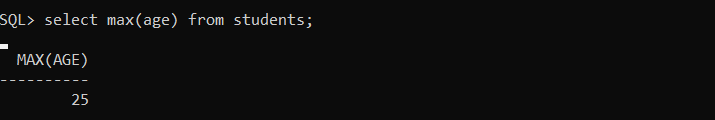
• MAX

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

**Syntax :**

SELECT MAX(column\_name)  
FROM table\_name  
WHERE condition;

**Example :**



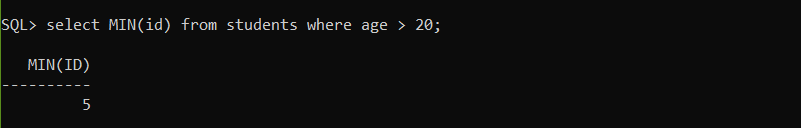
• MIN

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

**Syntax :**

SELECT MIN(column\_name)  
FROM table\_name  
WHERE condition;

**Example :**



**USING GROUP BY FOR AGGREGATE FUNCTIONS**

* groups rows that have the same values into summary rows

The GROUP BY statement is (often) used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

**Syntax:**

SELECT column\_name(s)  
FROM table\_name  
WHERE condition  
GROUP BY column\_name(s)HAVING conditionORDER BY column\_name(s);

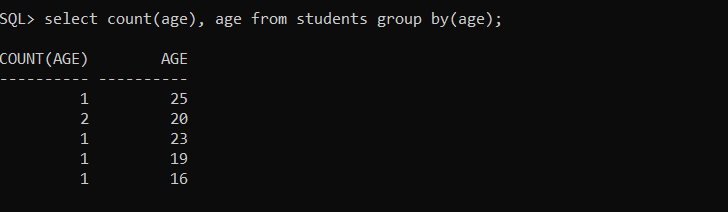
* **GROUP BY COUNT()**

It will return the count of the frequency of each value of the column passed

In group by () function.

Note :- When we do group by then either function or the column name on which the group has been made can be given.

**Example :**



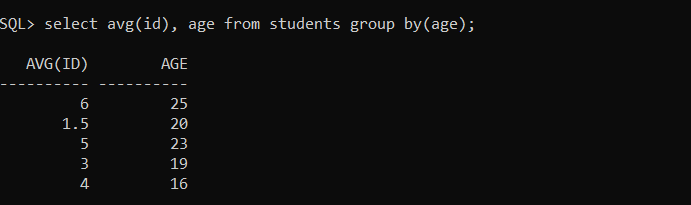
* **GROUP BY AVG()**

Here avg of id is grouped on the basis of age frequency.

Eg. Age = 20 occurs for id = 1,2

* 1+2 = 3
* 3/freq(age)
* 3/2 (20 occurred two times)
* 1.5

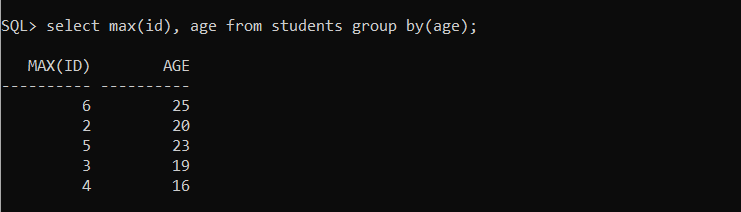
**Example :**

****

* **GROUP BY MAX()**

Here query returned max(ids with same age).

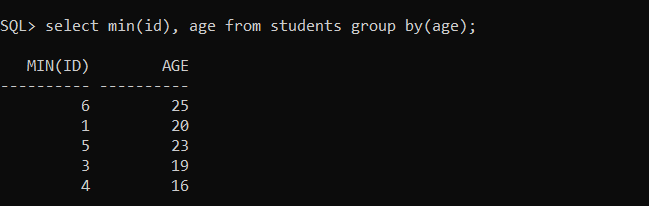
**Example :**



* **GROUP BY MIN()**

Here query returned min(ids with same age).

**Example :**



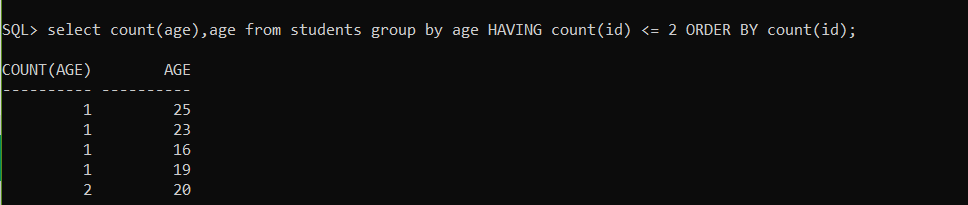
**USING HAVING FOR AGGREGATE FUNCTIONS**

The HAVING clause is used in place of WHERE keyword since WHERE Keyword cannot be used with aggregate functions.

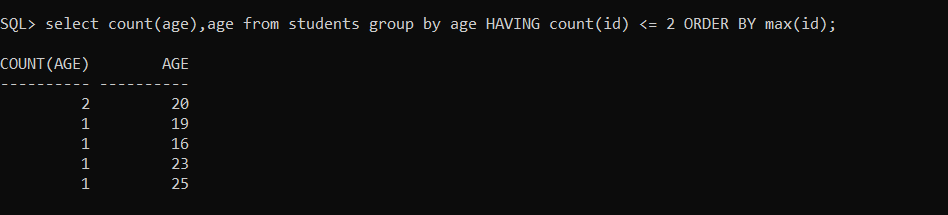
**Syntax:**

SELECT column\_name(s)  
FROM table\_name  
WHERE condition  
GROUP BY column\_name(s)HAVING conditionORDER BY column\_name(s);

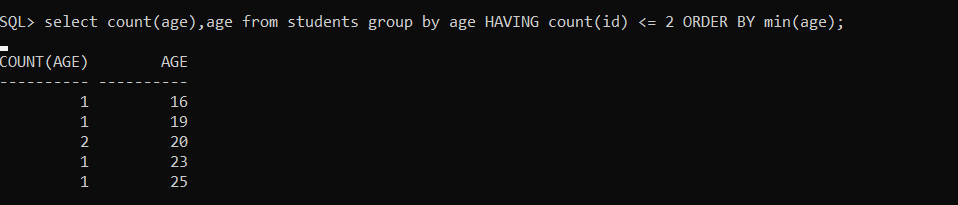
* **HAVING COUNT()**



* **HAVING MAX()**



* **HAVING MIN()**



* **HAVING AVG()**

