**PostgreSQL**

**Database:**

A database is collection of information that is set up for easy access, management and updating.

**DBMS**:

A database typically requires a comprehensive database software program known as database management system. A DBMS serves as an interface between the database and its end users or programs, allowing users to retrieve, update and manage how the information is organized and optimized.

**Types of Database management systems**:

* **Hierarchical database systems** - it resembles a tree like structure and it tells about the relationship between parent and child nodes.
* **Network database systems** - this also has a hierarchical structure. But instead of using using single-parent tree hierarchy it supports many-to-many relationships, as child tables can have more than one parent.
* **Relational database systems** - the data are represented in the table form i.e., rows and columns.
* **Object-oriented database systems -** the information is represented as objects, with different types of relationships possible between two or more objects.

**SQL- Structured Query Language:**

A stsndardized programming language which is used for managing relational databases. With this, we can modify databases, add, update or delete rows of data, retrieve subsets of information from a database and many more.

**PostgreSQL:**

PostgreSQL is an open-source object relational database system with a 30+ years of active development in the industry.

It is the most advanced open source relational databases.

SQL command categories:

* DDL -Data Definition Language- CREATE, DROP, ALTER
* DML - Data Manipulation Language - SELECT, INSERT, UPDATE, DELETE.
* DCL - Data Control Language - GRANT, INVOKE
* TCL - Transaction Control Language - COMMIT, ROLLBACK, SAVEPOINT.

**Data Types:**

The data types provide several benefits like:

* Consistency
* Validation
* Compactness
* Performance

1. **Numeric types:**

Numeric types consists of 2-bytes, 4-bytes and 8-byte integers, 4-byte and 8-byte floating-point numbers and selectable-precision decimals.

1. **Monetary types:**

The money type stores a currency amount with a fixed fractional precision. Values of the numeric, int, and bigint data types can be ast to money. Floating point numbers is not rexommended due to rounding off errors.

1. **Character types:**

* varchar(n)- varying length with limit
* char(n) - fixed-length, blank padded
* Text - variable unlimited length

1. **Binary data types:**

Bytea-1 or 4 bytes - Store binary strings (varible length)

5.**Date/time types**:

* timestamp[(p)][without time zone]- 8 bytes - both date and time without zone
* TIMESTAMPTZ- 8 bytes - both date and time with time zone
* Date-date -4 bytes (only date no time will be displayed)
* time[(p)][without time zone] - 8 bytes - time but no date
* Time[ (p)]with time zone- 12 bytes - times of date only with time zone
* interval[fields][(p)]- 12 bytes - time interval

6. **Boolean type**

Boolean-1 byte-true or false

7. **Enumerated type:**

Static, ordered set of values.

CREATE TYPE week AS ENUM (‘Mon’,’Tue’,’Wed’,’Thu’,’Fri’,’Sat’,’sun’);

8. **Array type:**

To define a column of a table as a variable length multidimensional array.

* **Delcalaration of Arrays:**

CREATE TABLE monthly\_savings(

Name text,

Saving\_per\_quarter integer ARRAY[4],

Schema text[][]

);

* **Inserting values:**

INSERT INTO monthly\_savings

VALUES (‘Manisha’,’{20000,14600,23500,13250}’’{{“FD”,”MF”},{“FD”,”Property”}}’);

* **Accessing Arrays:**

SELECT name FROM monthly\_savings WHERE saving\_per\_quarter[2]>saving\_per\_quarter[4];

* **Modifying Arrays:**

UPDATE monthly\_savings SET saving\_per\_quarter=’{25000,25000,27000,27000}’ WHERE name=’Manisha’;

9. **Pseudo types:**

A pseudo-type cannot be used as a column data type, but it can be used to declare a function’s argument or result type.

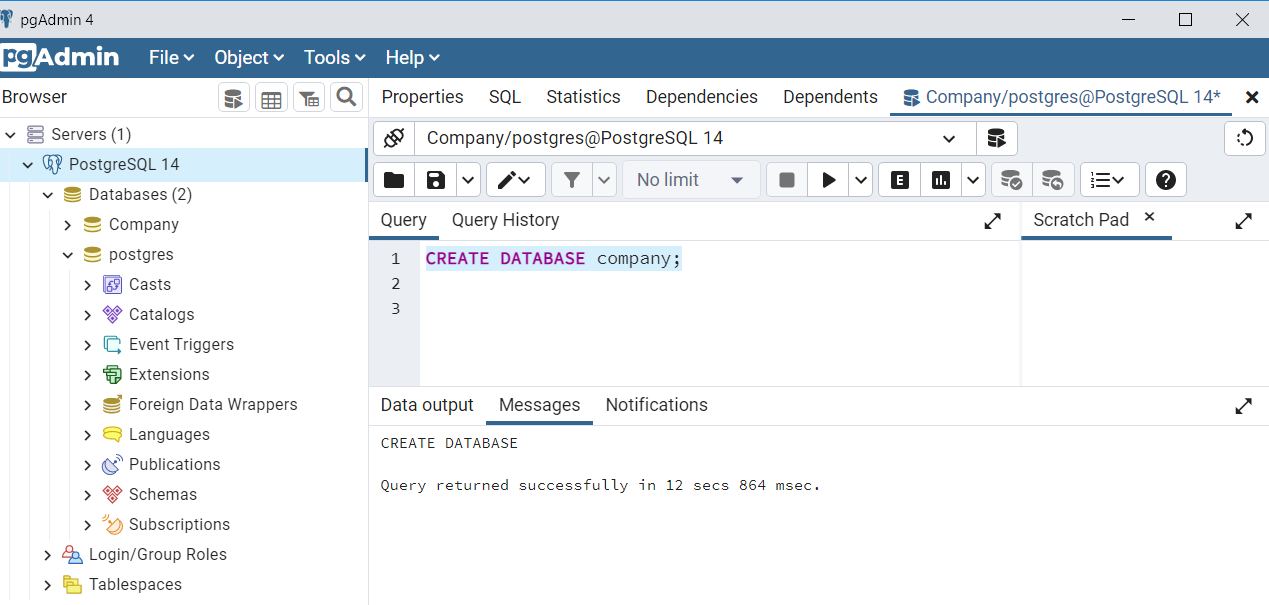
PostgreSQL-CREATE Database:

Allows to create database in two ways:

* Using CREATE DATABASE-an SQL command
* Using createdb a command-line executable.

**Using CREATE DATABASE:**

CREATE DATABASE dbname;



**Using createdb command:**

Createdb [option…] [dbname[description]]

* Dbname- name of a database to create
* Description- comment to be associate with the newly created database
* Options- command-line arguments, which createdb accepts
  + -D tablespace-default tablespace for the database
  + -e - Echo the commands that createdb generates and sends to the server.
  + -E encoding - specifies the character encoding scheme to be used in this database
  + -l locale
  + -T template
  + –help
  + -h host - host name of the machine on which the server is running
  + -p port
  + -U username
  + -w - never issue a password prompt
  + -W - force createdb to prompt for a password before connecting to a database.

**PostgreSQL- SELECT Database:**

**2 ways:**

* Database SQL Prompt -

\l - this will display the available databases

\c - this is used to connect/select a desired database.

* OS Command Prompt-

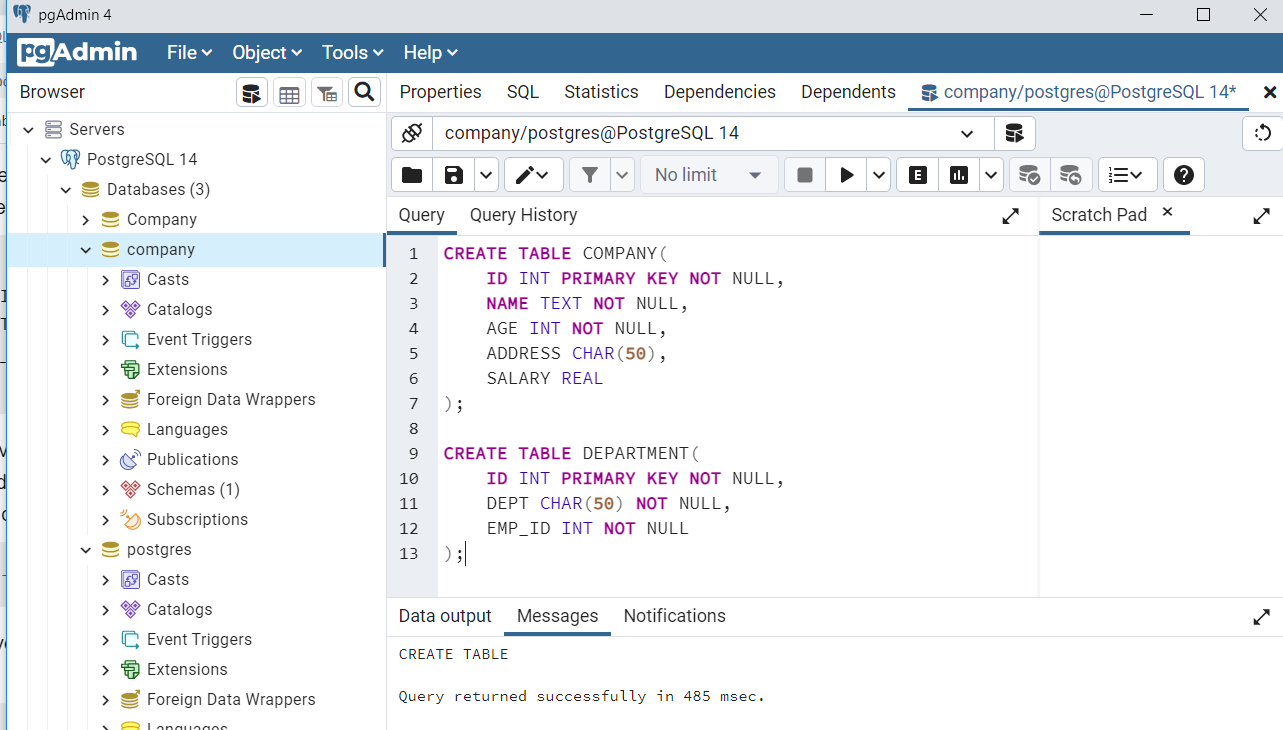
Psql -h loalhost -p 5432 -U postgress testdb. This will let you to connect to the database testdb.

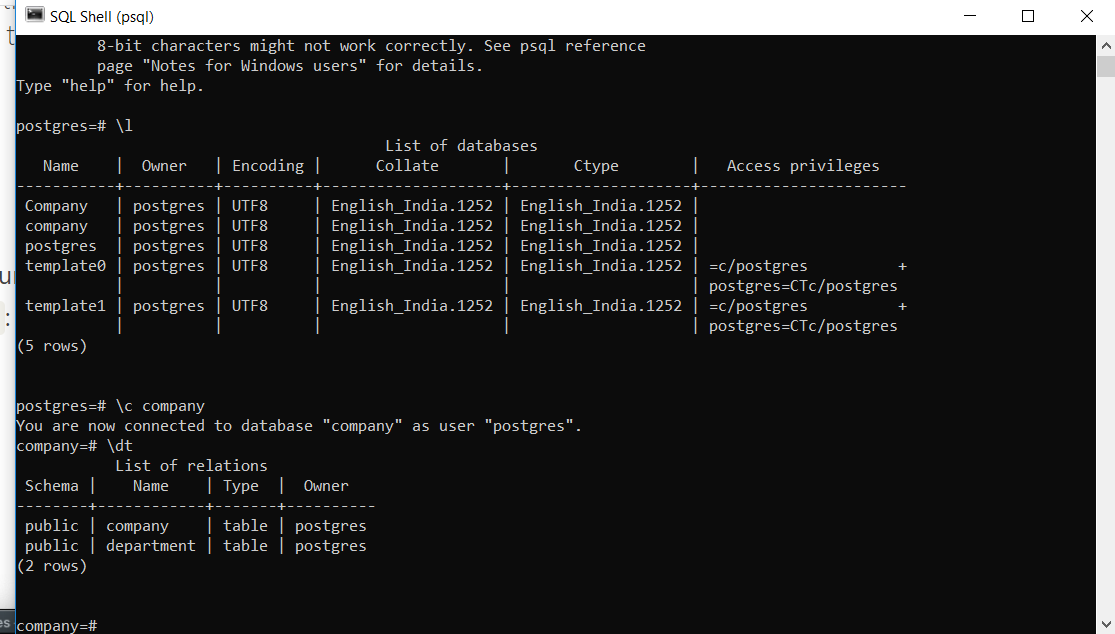
**PostgresSQL - DROP Database:**

**Syntax:**

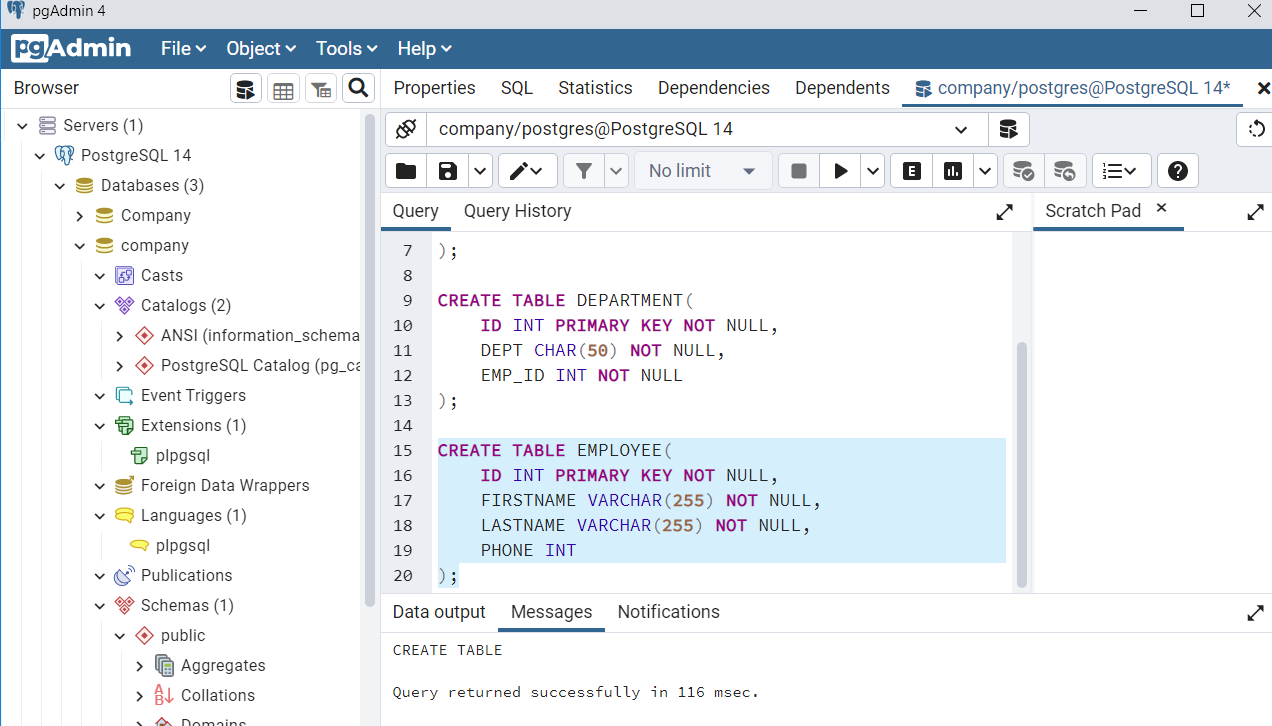
DROP DATABASE [IF EXISTS] dbname;

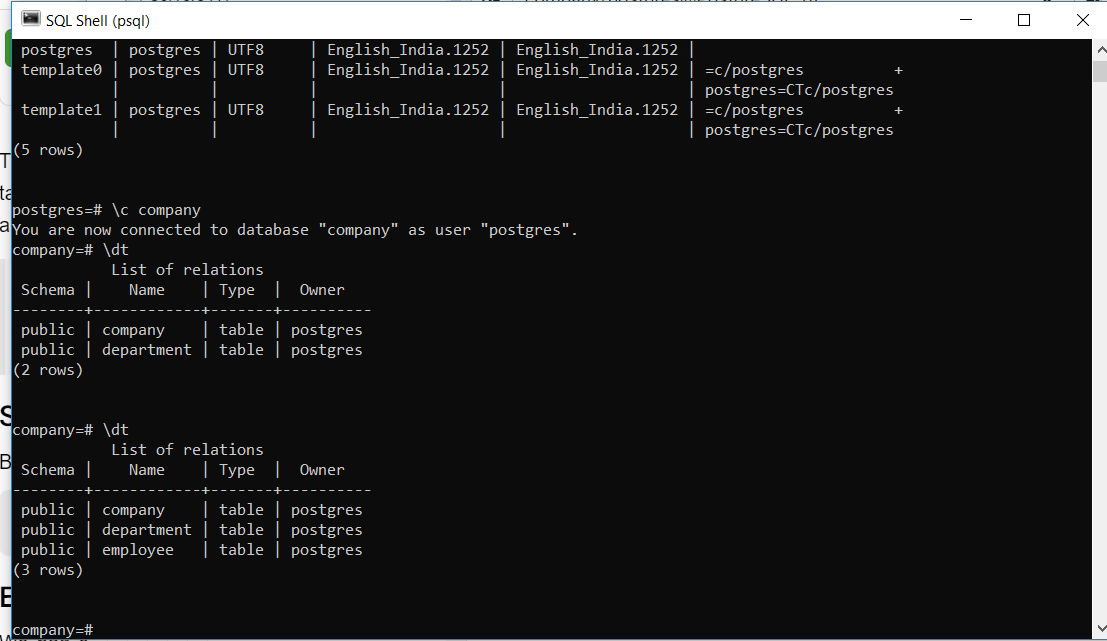
**PostgresSQL - CREATE table**

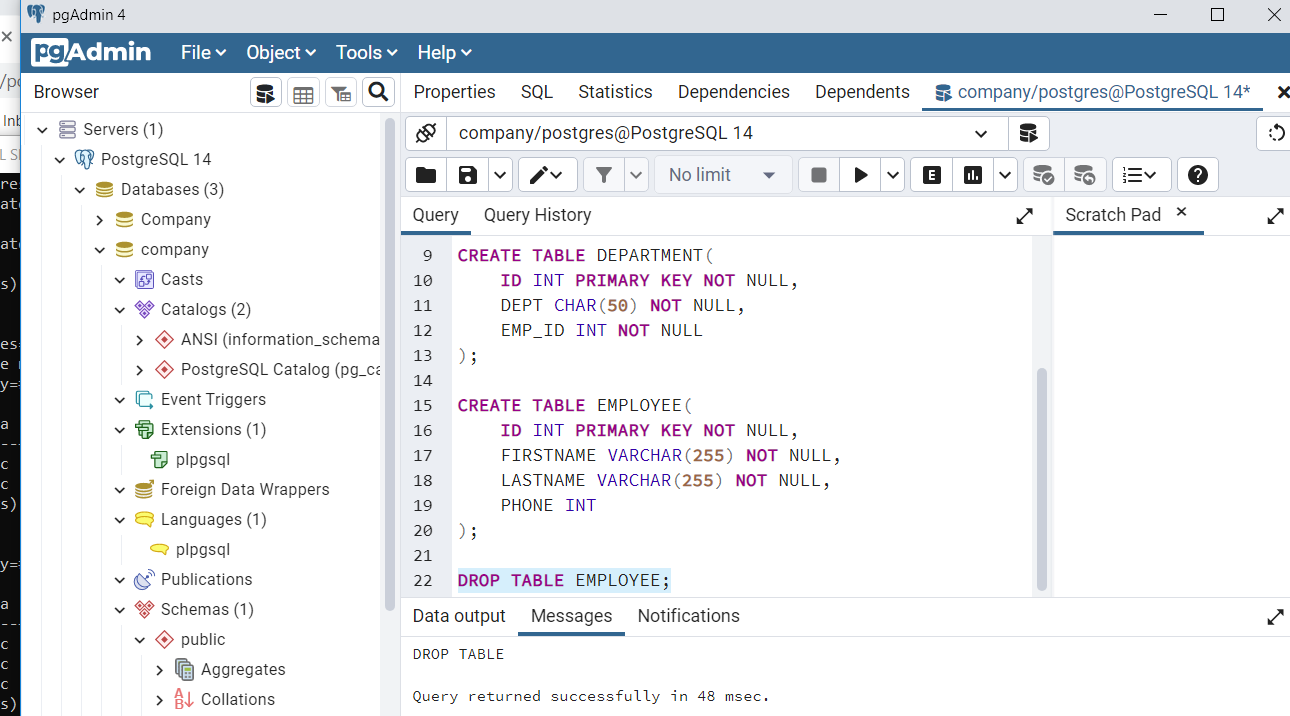
****

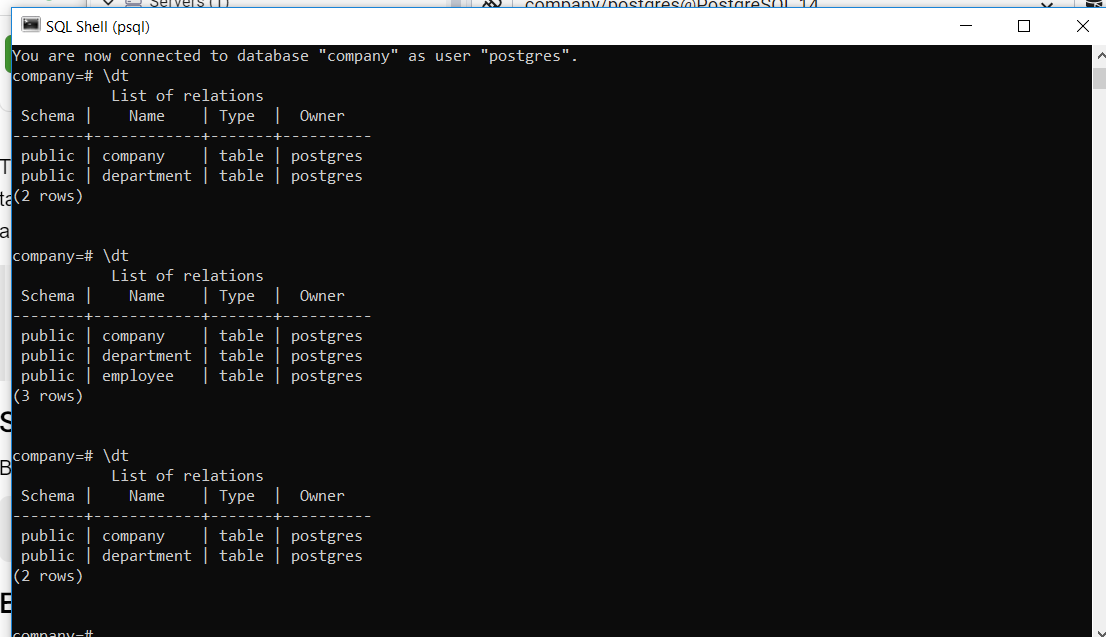


**DROP TABLE:**

****

****

****

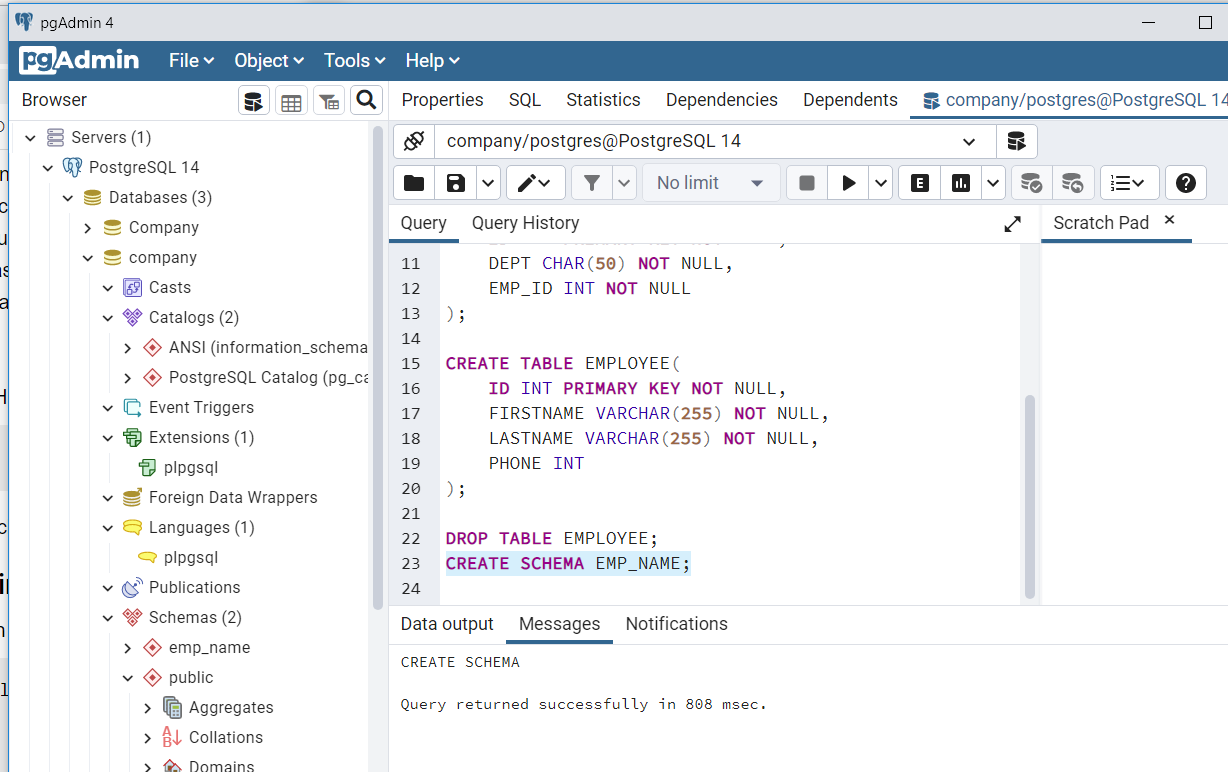
****

**PostgreSQL-Schema:**

A schema is a named collection of tables. A schema can aslo contain views, indexes, sequences, data types, operatore and functions.

Syntax:

CREATE SCHEMA name;

****

**PostgreSQL - Insert query:**

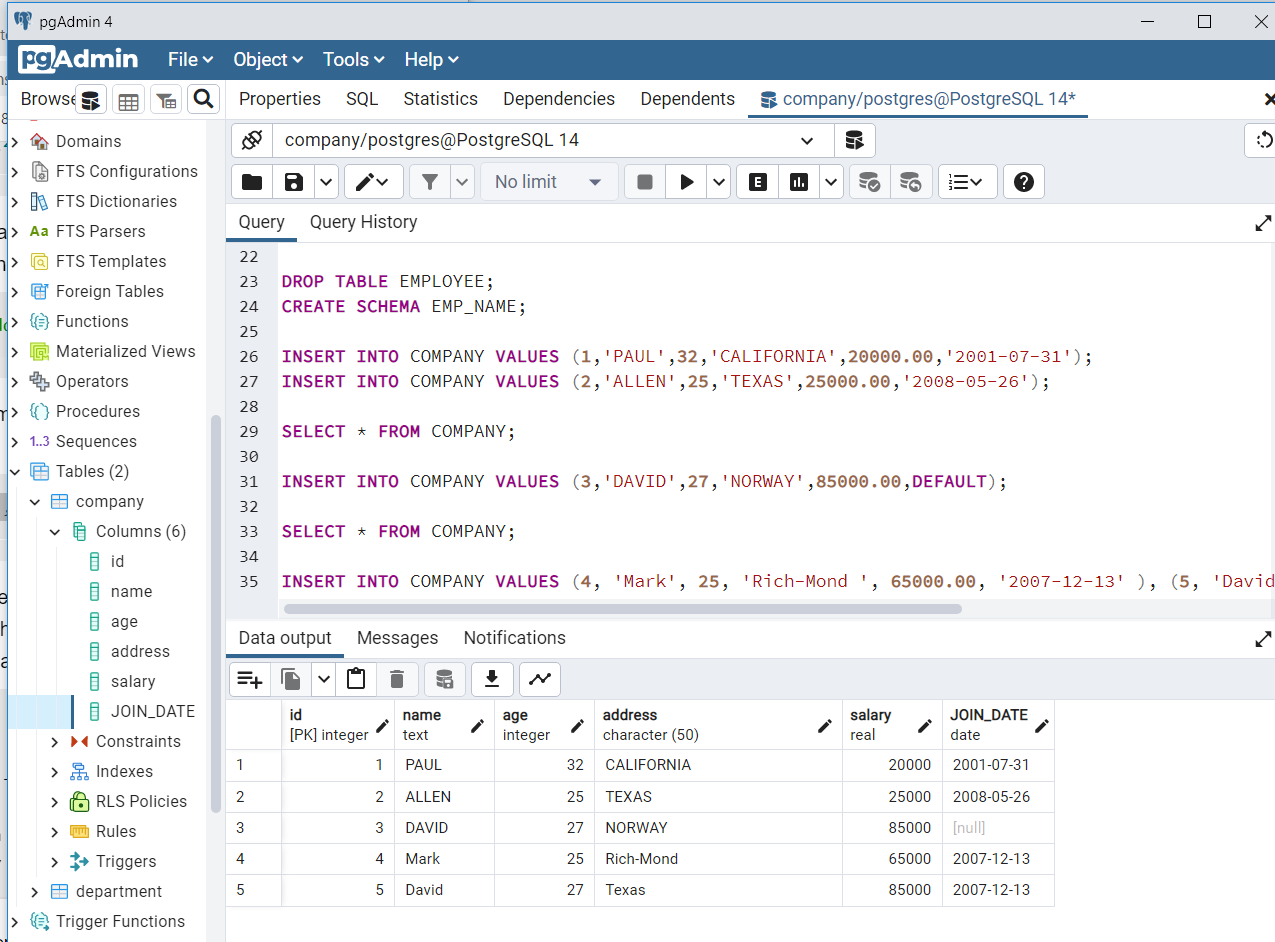
**Syntax:**

**INSERT INTO TABLE\_NAME (column1, column2, column3,...columnN)**

**VALUES (value1, value2, value3,...valueN);**

If no values is entered for a particular column then DEFAULT is used.

Multiple rows can be given in one INSERT query using a comma.

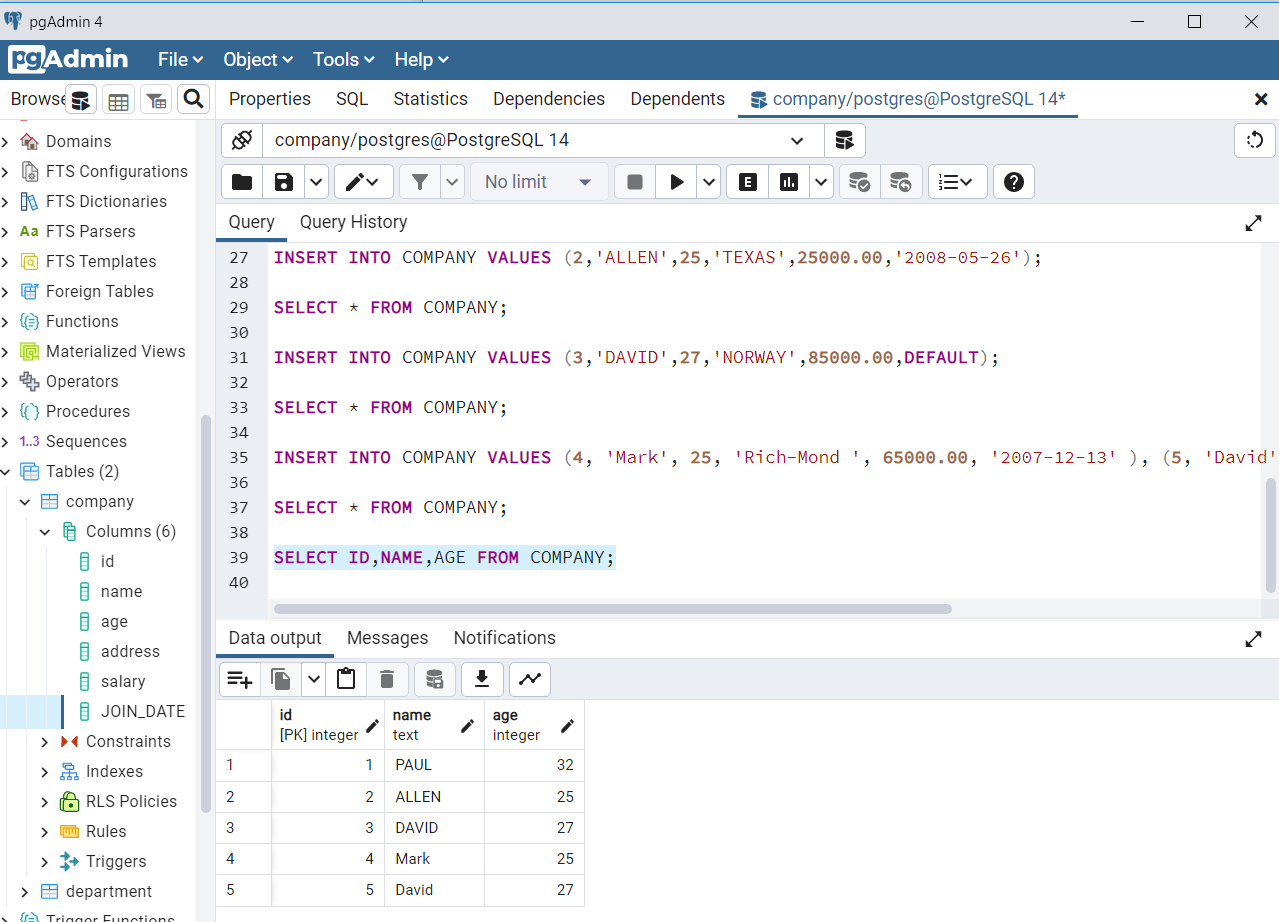
****

**postgreSQL - SELECT query**

**Syntax:**

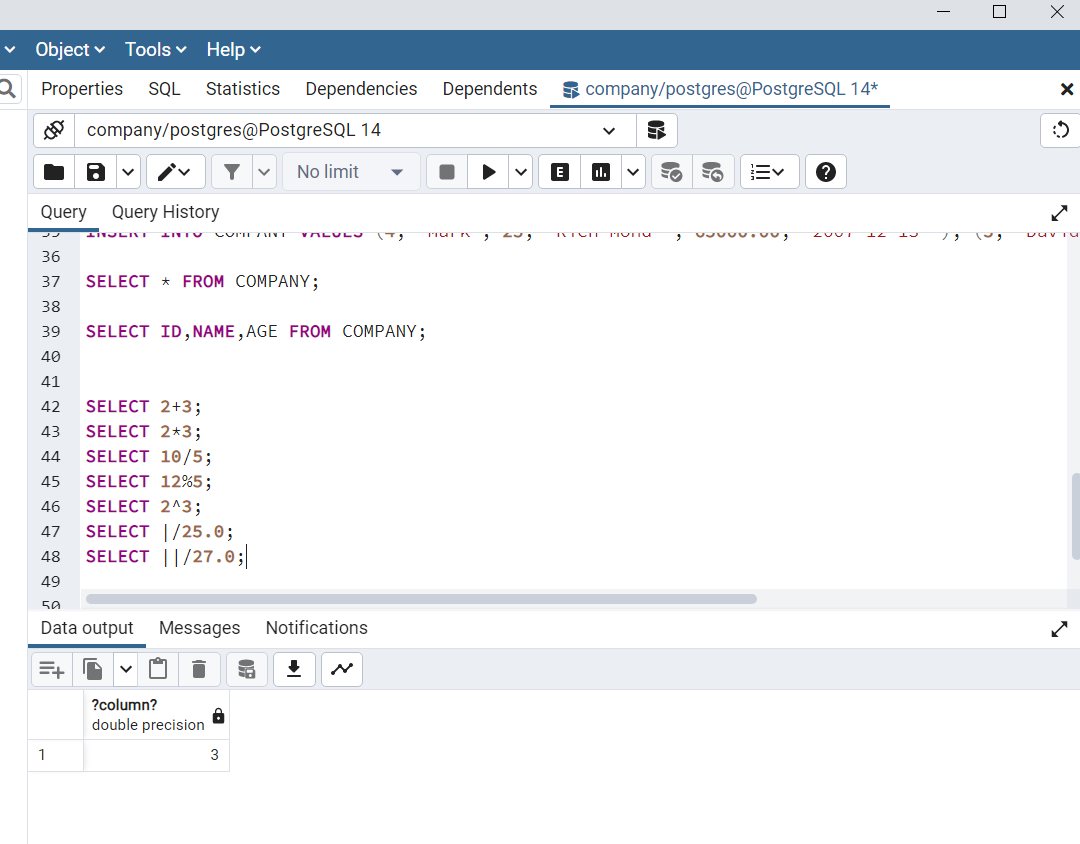
SELECT \* FROM TABLE\_NAME; - This will display the entire content of the table.

SELECT COLUMN\_NAME1, COLUMN\_NAME2 FROM TABLE\_NAME; - This will display the contents of only those particular columns of the table.

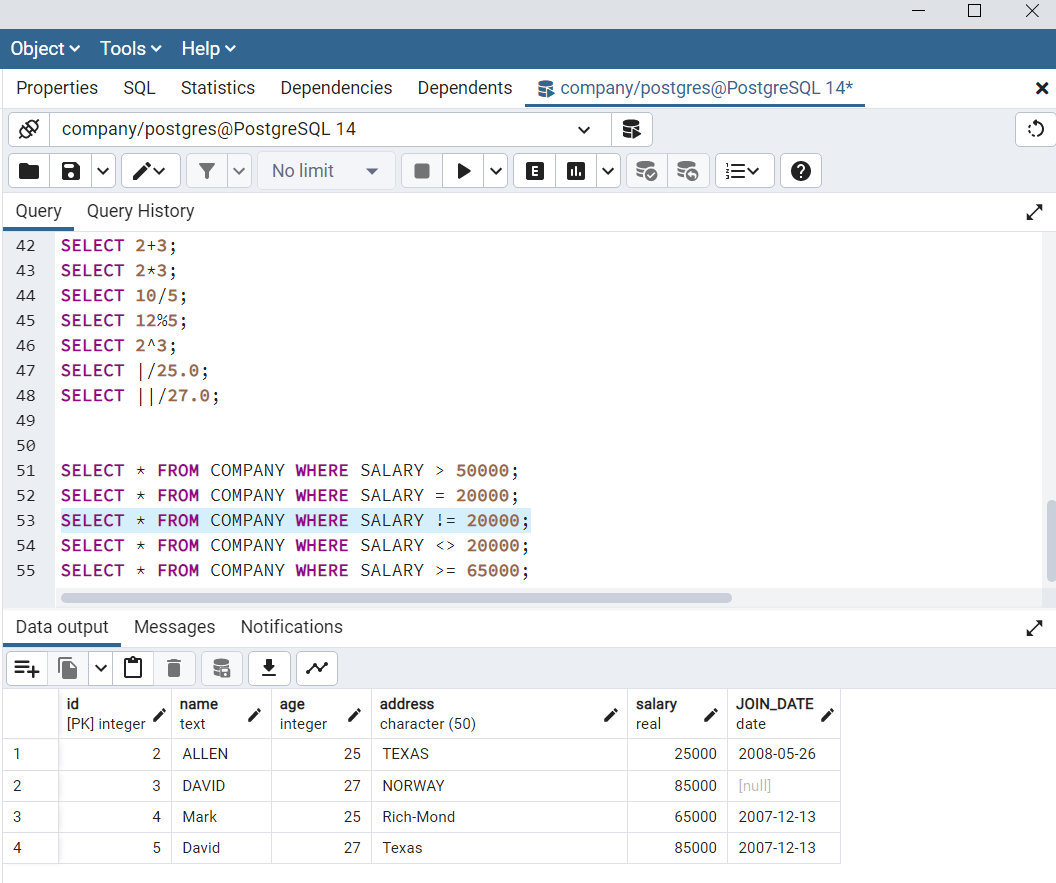


postgreSQL - operators:

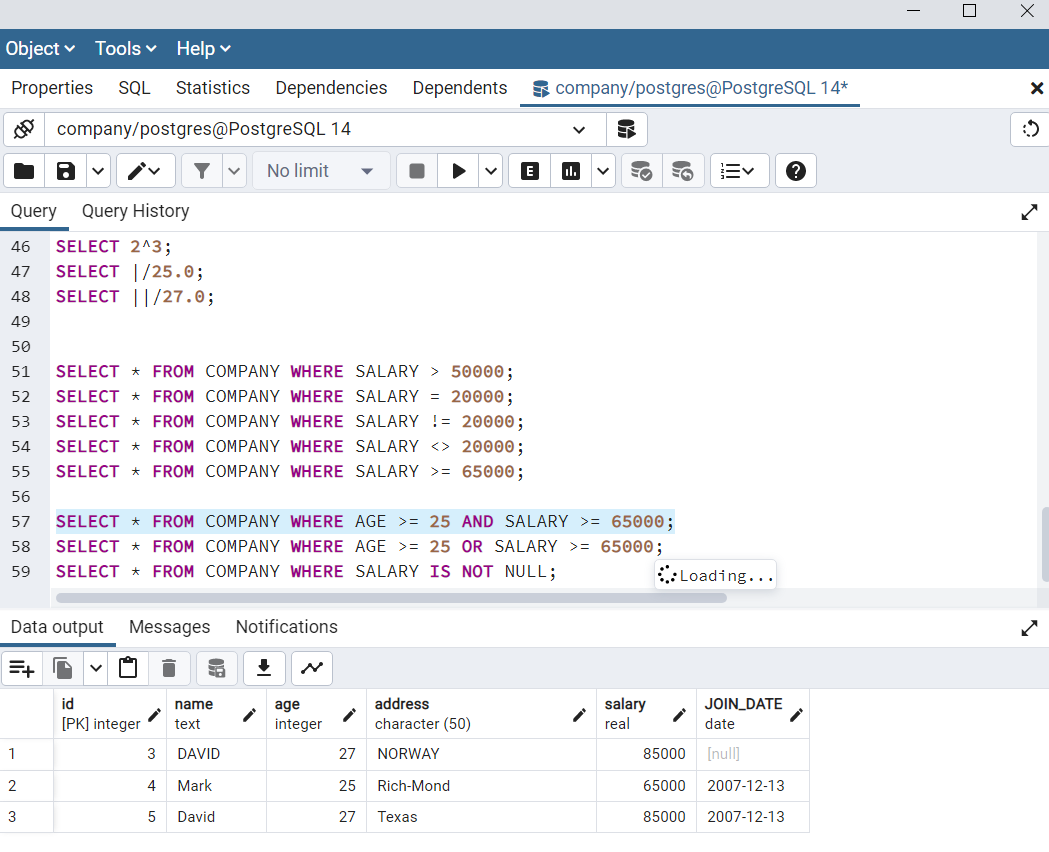
1. Arithmetic operators:

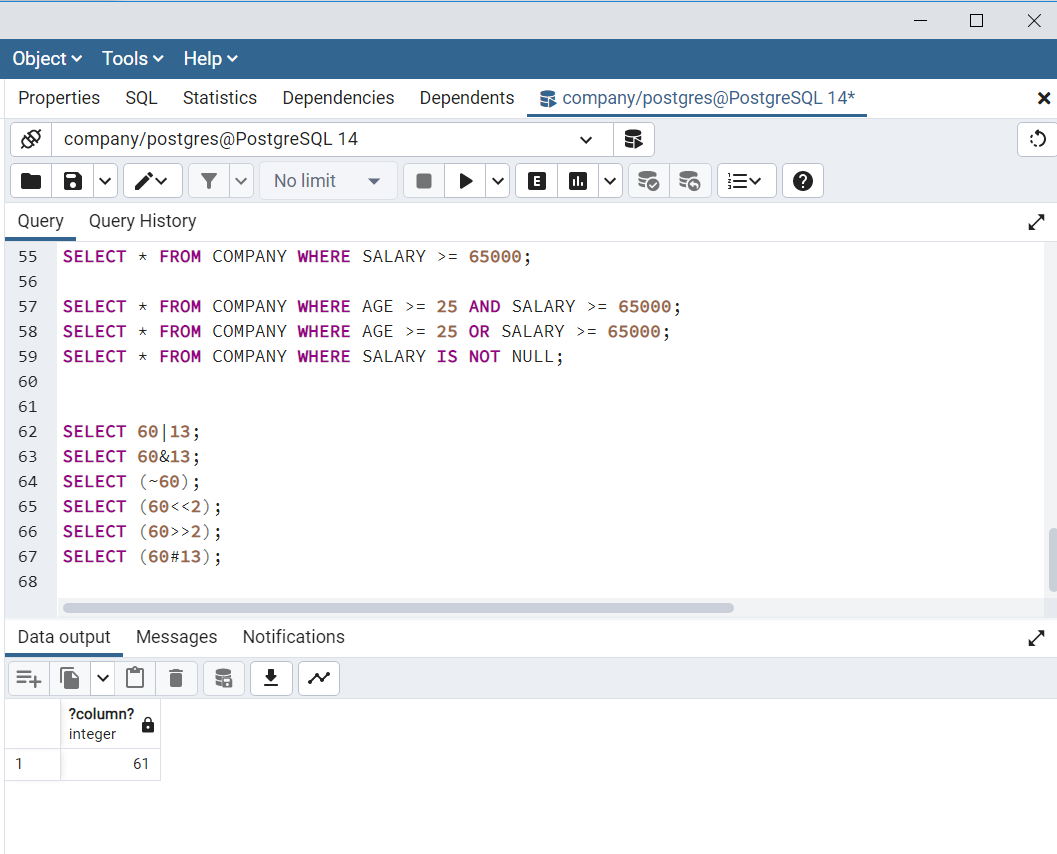


1. Comparison operators



1. Logical operators



1. Bitwise operators

postgreSQL - Expressions:

Syntax:

SELECT column1, column2, columnN

FROM table\_name

WHERE [CONDITION | EXPRESSION];

Example:

SELECT \* FROM COMPANY WHERE SALARY = 10000;

postgreSQL - WHERE:

The WHERE clause is used to specify a condition while fetching the data from single table or joining with multiple tables.

Syntax:

SELECT column1, column2, columnN

FROM table\_name

WHERE [search\_condition]

Conditions like:

**AND** - both the conditions should satisfy

**OR** - any one of the condition should satisfy

**IS NOT NULL** - displays all the rows which has values

**LIKE** - ‘Pa%’ - displays the name starting with Pa and doesnot matters about what comes after Pa.

**IN** - displays all the rows with the specified values

**NOT IN** - displays all rows other than the ones having the specified values.

**BETWEEN** - displays all the values between the range

**EXISTS** -

SELECT AGE FROM COMPANY WHERE EXISTS (SELECT AGE FROM COMPANY WHERE SALARY > 65000);

Here the inner query is executed first and all the values of age exists in the inner query will be displayed.

**postgreSQL - AND & OR Clauses**

**Syntax:**

**SELECT column1, column2, columnN**

**FROM table\_name**

**WHERE [condition1] AND [condition2]...AND [conditionN];**

**Example:**

SELECT \* FROM COMPANY WHERE AGE >= 25 AND SALARY >= 65000;

**SELECT column1, column2, columnN**

**FROM table\_name**

**WHERE [condition1] OR [condition2]...OR [conditionN];**

Example:

SELECT \* FROM COMPANY WHERE AGE >= 25 OR SALARY >= 65000;

postgreSQL- UPDATE

Syntax:

UPDATE table\_name

SET column1 = value1, column2 = value2...., columnN = valueN

WHERE [condition];

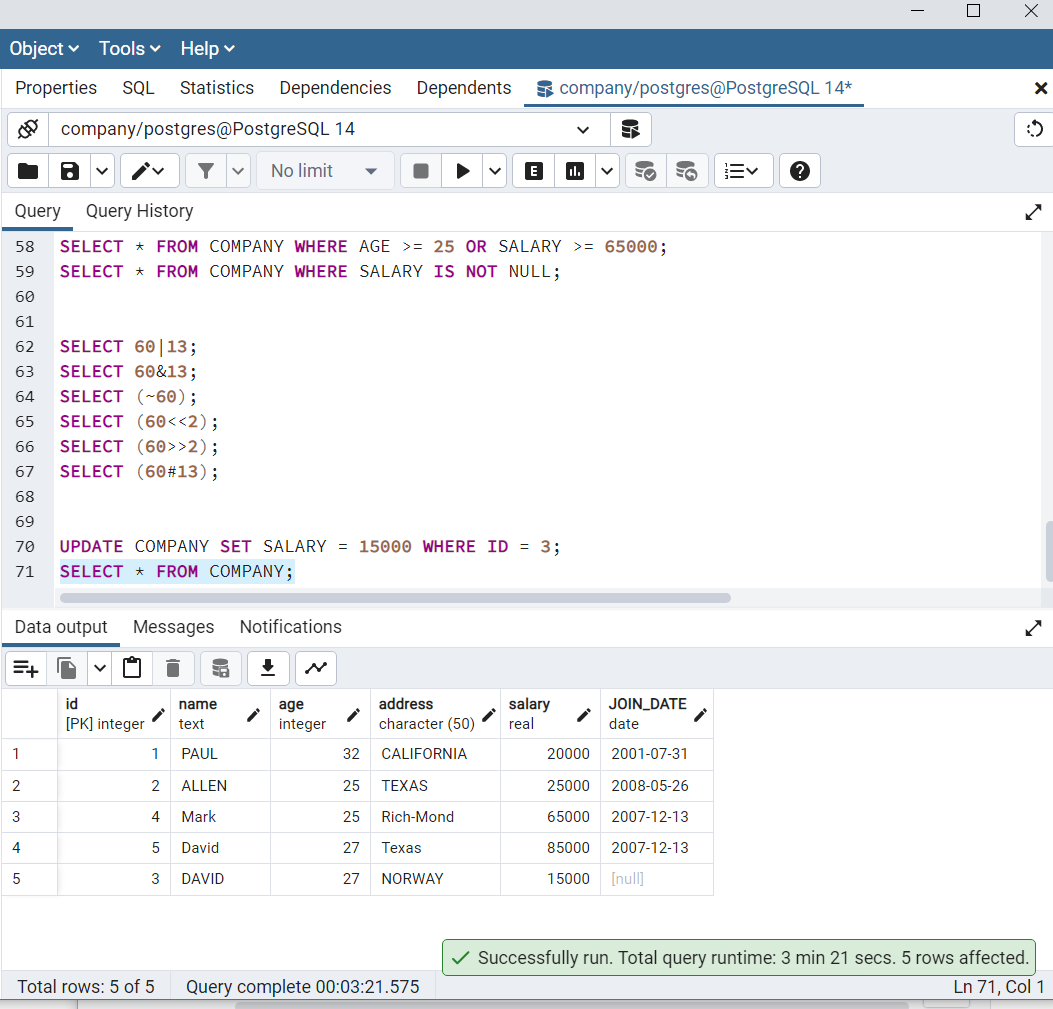
postgreSQL - Update query

Syntax:

UPDATE table\_name

SET column1 = value1, column2 = value2...., columnN = valueN

WHERE [condition];

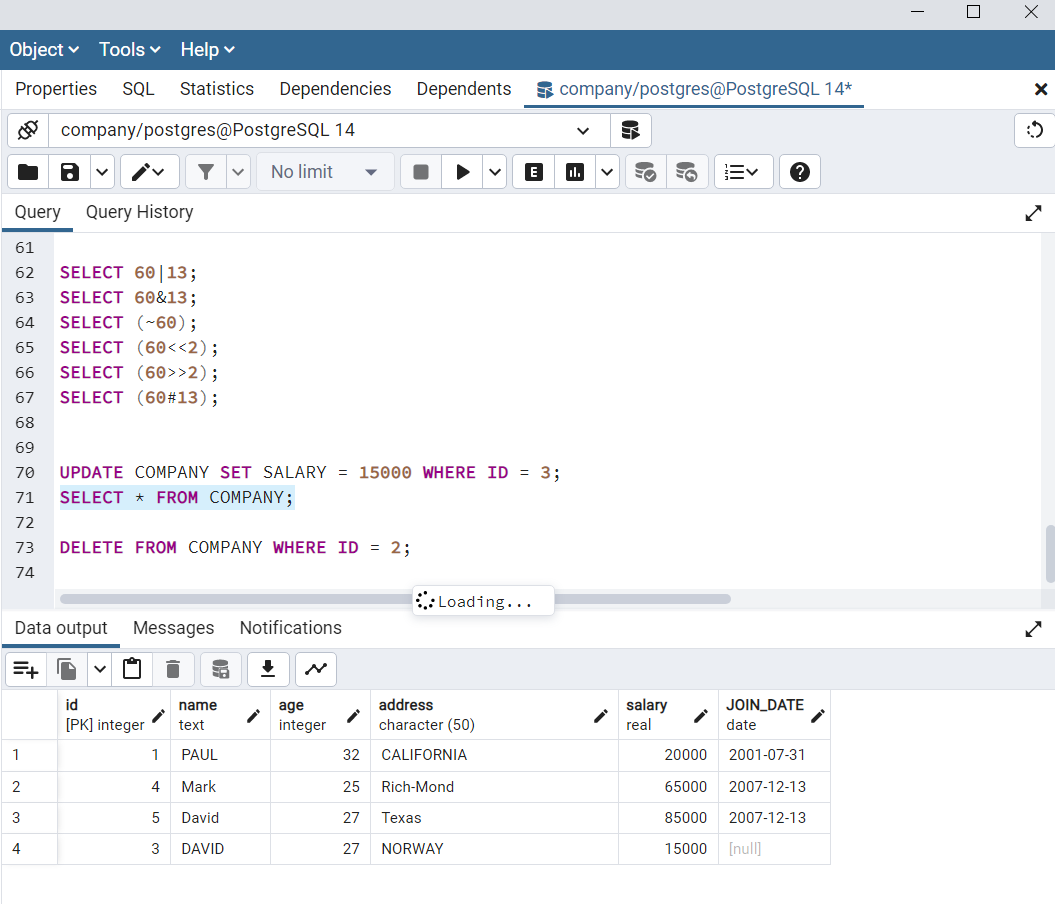


**postgreSQL - Delete query**

Syntax:

DELETE FROM table\_name

WHERE [condition];



**postgresSQL - LIKE query:**

The like operator will return true, which is 1.

It uses 2 signs:

* The percent sign (%)- this represents zero, one, or multiple numbers or characters
* The underscore(\_) - this represents a single number or character.

Syntax:

SELECT FROM table\_name

WHERE column LIKE 'XXXX%'

or

SELECT FROM table\_name

WHERE column LIKE '%XXXX%'

or

SELECT FROM table\_name

WHERE column LIKE 'XXXX\_'

or

SELECT FROM table\_name

WHERE column LIKE '\_XXXX'

or

SELECT FROM table\_name

WHERE column LIKE '\_XXXX\_'

* '200%' - Finds any values that start with 200
* %200% - Finds any values that have 200 in any position
* \_00% - Finds any values that have 00 in the second and third positions
* 2\_%\_% - Finds any values that start with 2 and are at least 3 characters in length
* %2 - Finds any values that end with 2
* \_2%3 - Finds any values that have 2 in the second position and end with a 3
* 2\_\_\_3 - Finds any values in a five-digit number that start with 2 and end with 3
* %-% - displays all the records in which it has a hyphen (-)

**PostgreSQL - LIMIT clause:**

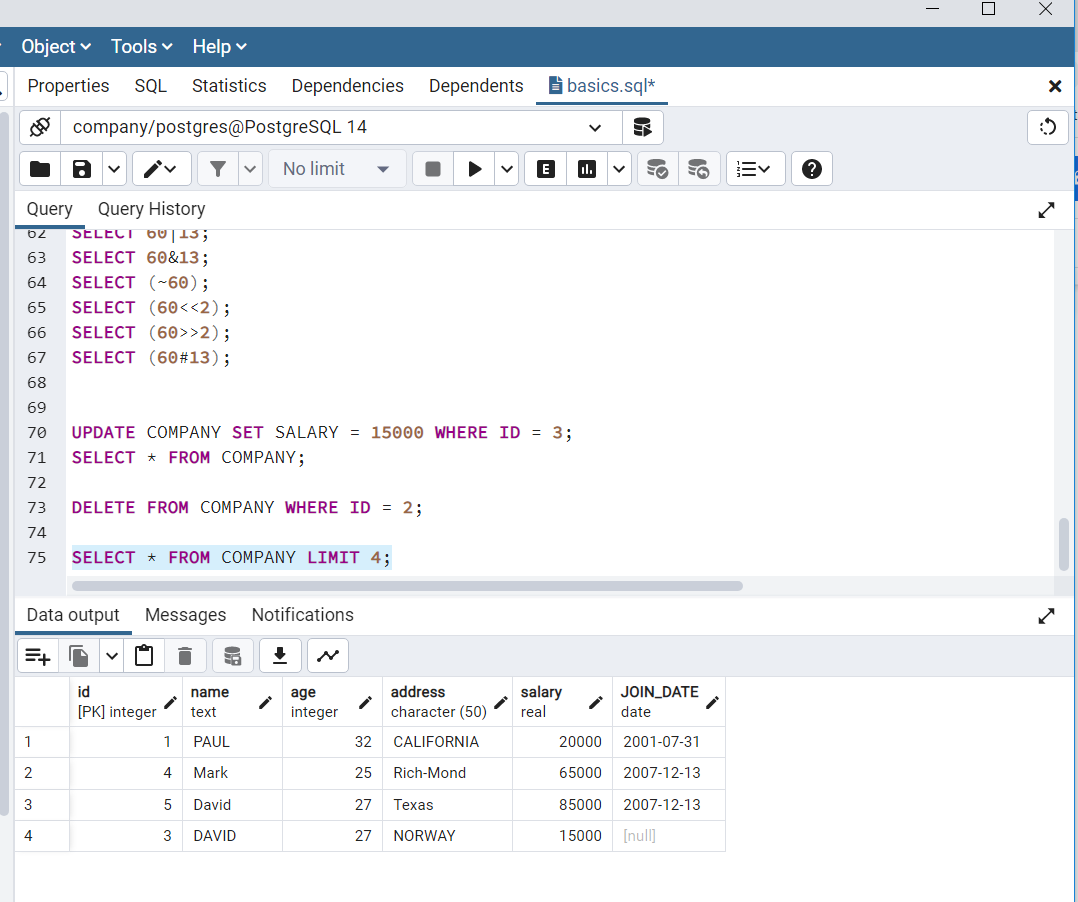
It is used to limit the data amount returned by the SELECT statement.

**Syntax:**

SELECT column1, column2, columnN

FROM table\_name

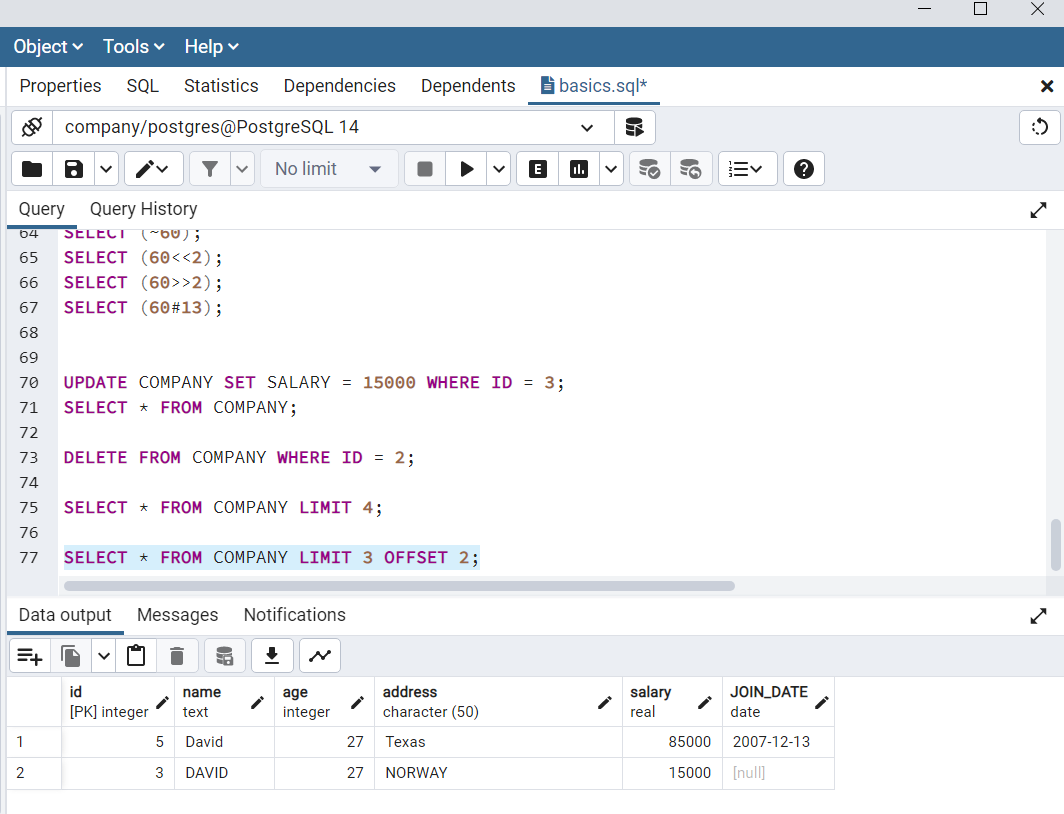
LIMIT [no of rows]



SELECT column1, column2, columnN

FROM table\_name

LIMIT [no of rows] OFFSET [row num]



# **PostgreSQL - ORDER BY Clause**

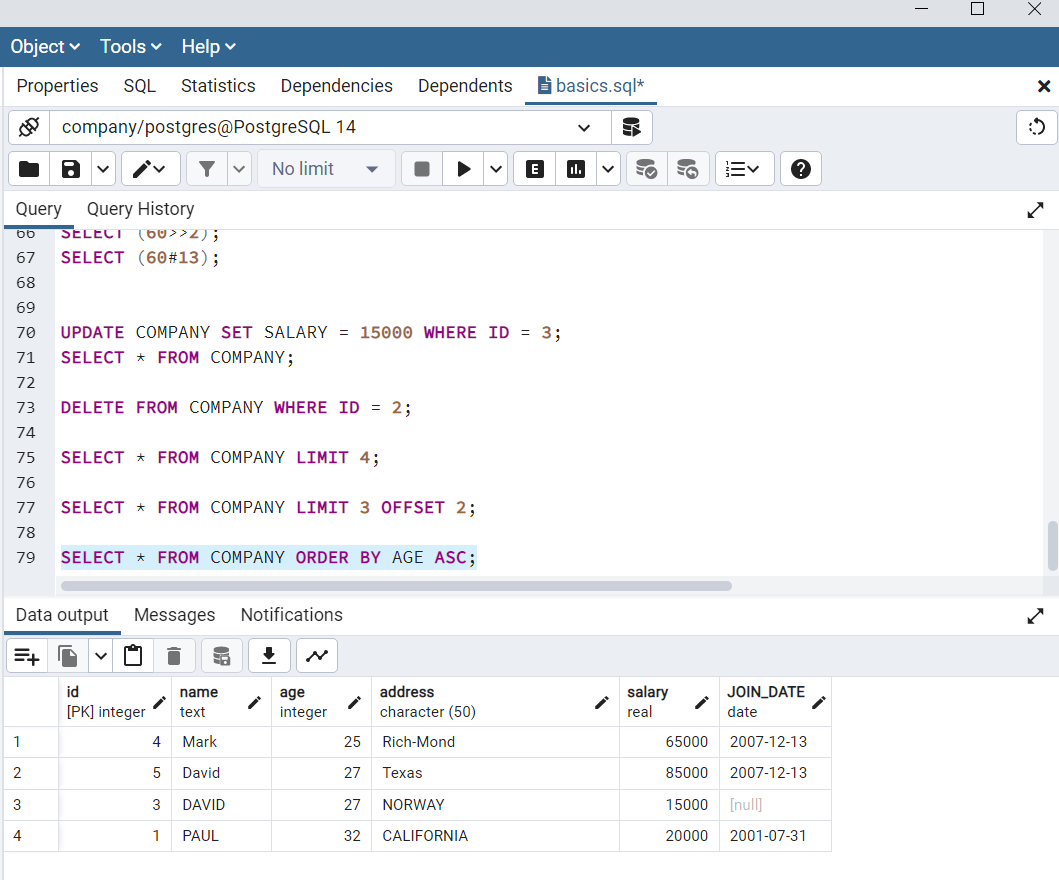
**Syntax:**

SELECT column-list

FROM table\_name

[WHERE condition]

[ORDER BY column1, column2, .. columnN] [ASC | DESC];



Example:

SELECT \* FROM COMPANY ORDER BY NAME, SALARY ASC;

Here it would sort the result in ascending order by NAME and SALARY

**postgreSQL - GROUPBY**

The PostgreSQL GROUP BY clause is used in collaboration with the SELECT statement to group together those rows in a table that have identical data.

**Syntax:**

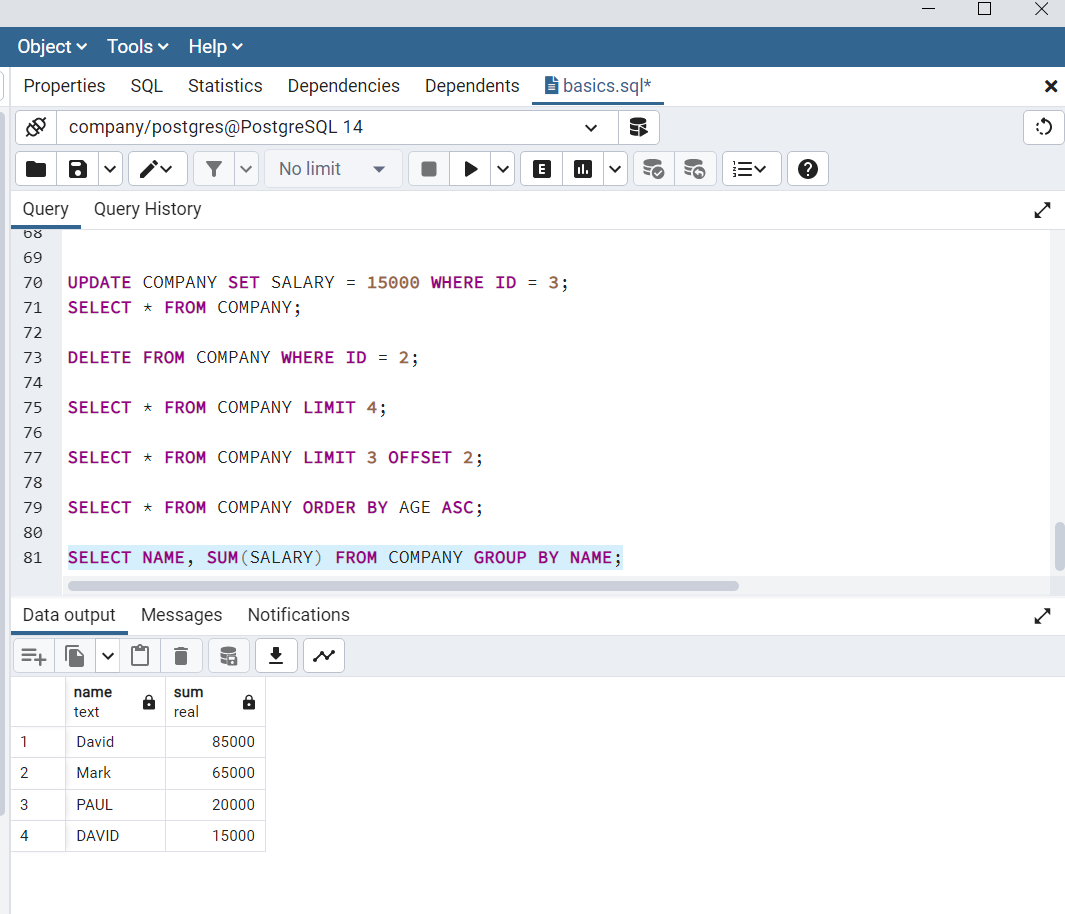
SELECT column-list

FROM table\_name

WHERE [ conditions ]

GROUP BY column1, column2....columnN

ORDER BY column1, column2....columnN



**postgreSQL - Having clause**

**Syntax:**

SELECT column1, column2

FROM table1, table2

WHERE [ conditions ]

GROUP BY column1, column2

HAVING [ conditions ]

ORDER BY column1, column2

Example:

SELECT NAME FROM COMPANY GROUP BY name HAVING count(name) < 2;

**postgreSQL - DISTINCT**

The PostgreSQL DISTINCT keyword is used in conjunction with SELECT statement to eliminate all the duplicate records and fetching only unique records.

Syntax:

SELECT DISTINCT column1, column2,.....columnN

FROM table\_name

WHERE [condition]

Example:

SELECT DISTINCT name FROM COMPANY;

**PostgreSQL - Constraints**

Constraints are the rules enforced on data columns on table. These are used to prevent invalid data from being entered into the database. This ensures the accuracy and reliability of the data in the database. These constraints can be column level or table level.

* NOT NULL constraint - ensures the column does not have NULL values.

Example:

CREATE TABLE COMPANY1(

ID INT PRIMARY KEY NOT NULL,

NAME TEXT NOT NULL,

AGE INT NOT NULL,

ADDRESS CHAR(50),

SALARY REAL

);

* UNIQUES constraint - ensures that all values in a column are different

Example:

CREATE TABLE COMPANY3(

ID INT PRIMARY KEY NOT NULL,

NAME TEXT NOT NULL,

AGE INT NOT NULL UNIQUE,

ADDRESS CHAR(50),

SALARY REAL DEFAULT 50000.00

);

* PRMARY key - uniquely identifies each row/record in a database table. The primary key cannot have NULL values. A table can have only one primary key, which may consists of single or multiple fields. When fields are used as a primary key, they are called as composite key.

Example:

CREATE TABLE COMPANY4(

ID INT PRIMARY KEY NOT NULL,

NAME TEXT NOT NULL,

AGE INT NOT NULL,

ADDRESS CHAR(50),

SALARY REAL

);

* FOREIGN key - constrains data based on columns in other tables

Example:

CREATE TABLE COMPANY6(

ID INT PRIMARY KEY NOT NULL,

NAME TEXT NOT NULL,

AGE INT NOT NULL,

ADDRESS CHAR(50),

SALARY REAL

);

CREATE TABLE DEPARTMENT1(

ID INT PRIMARY KEY NOT NULL,

DEPT CHAR(50) NOT NULL,

EMP\_ID INT references COMPANY6(ID)

);

* CHECK constraint - THE CHECK constraint ensures that all values in a column satisfy certain conditions

Example:

CREATE TABLE COMPANY5(

ID INT PRIMARY KEY NOT NULL,

NAME TEXT NOT NULL,

AGE INT NOT NULL,

ADDRESS CHAR(50),

SALARY REAL CHECK(SALARY > 0)

);

* EXCLUSION constraint - The EXCLUDE constraint ensures that if any two rows are compared on the specified columns or expressions using the specified operators, not all of these comparisons will return true.

Example:

CREATE TABLE COMPANY7(

ID INT PRIMARY KEY NOT NULL,

NAME TEXT,

AGE INT ,

ADDRESS CHAR(50),

SALARY REAL,

EXCLUDE USING gist

(NAME WITH =,

AGE WITH <>)

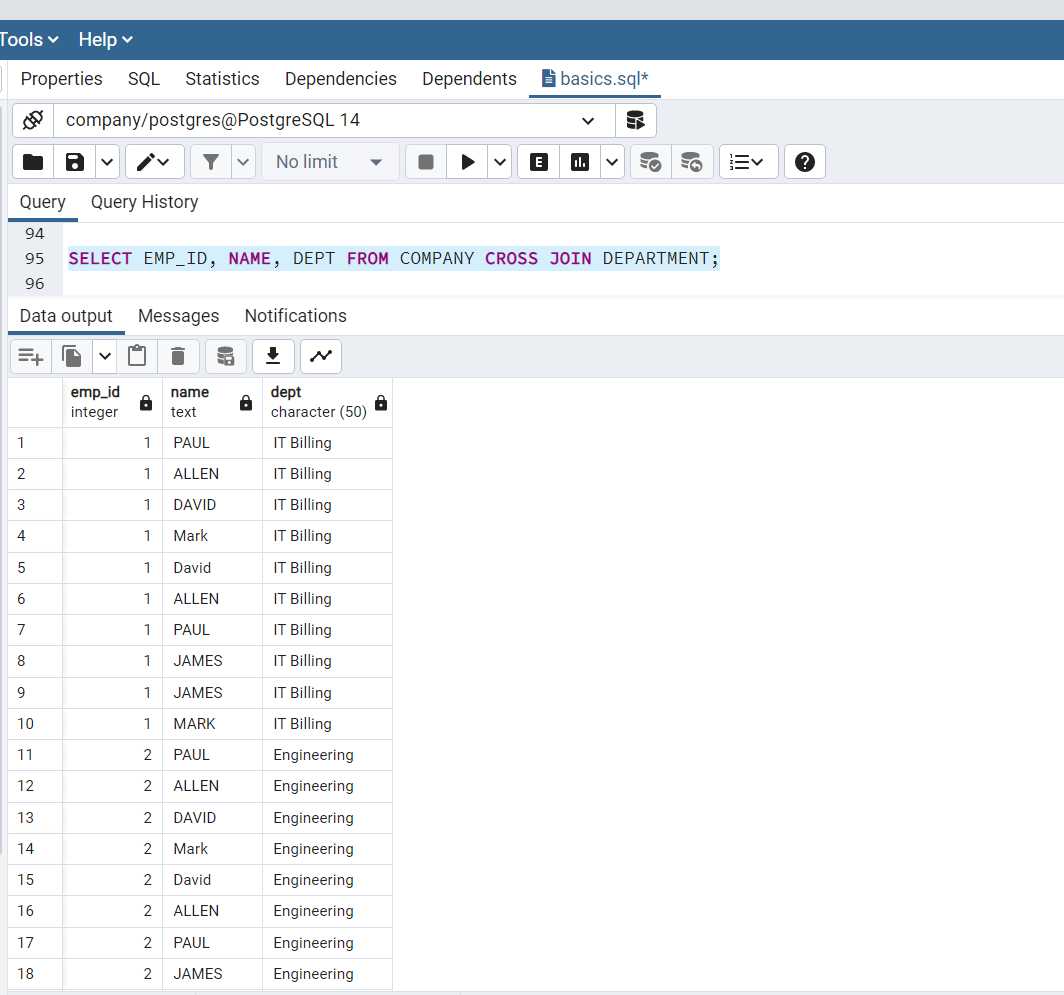
);

PostgreSQL - JOINS:

The postgreSQL joins clause is used to combine records from two or more tables in a database. The types are:

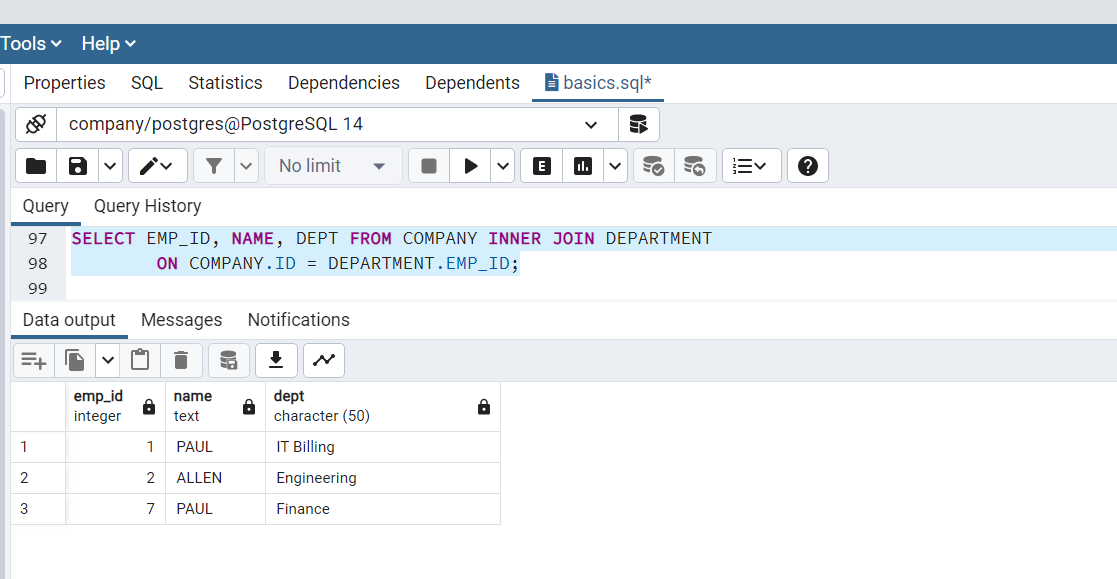
* CROSS JOIN

A cross join matches every row of the first table with every row of the second table.



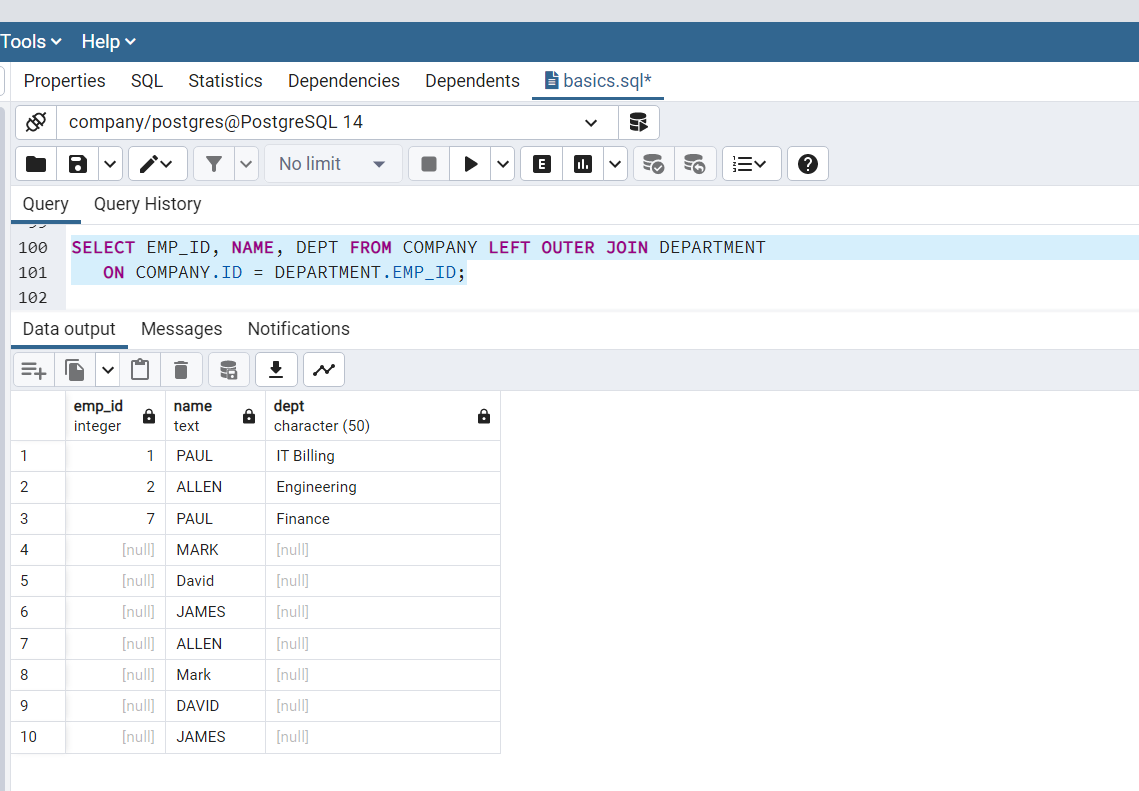
* INNER JOIN

Inner join creates a new result table by combining column values of two tables (table 1 and table 2) based upon the join-predicate.



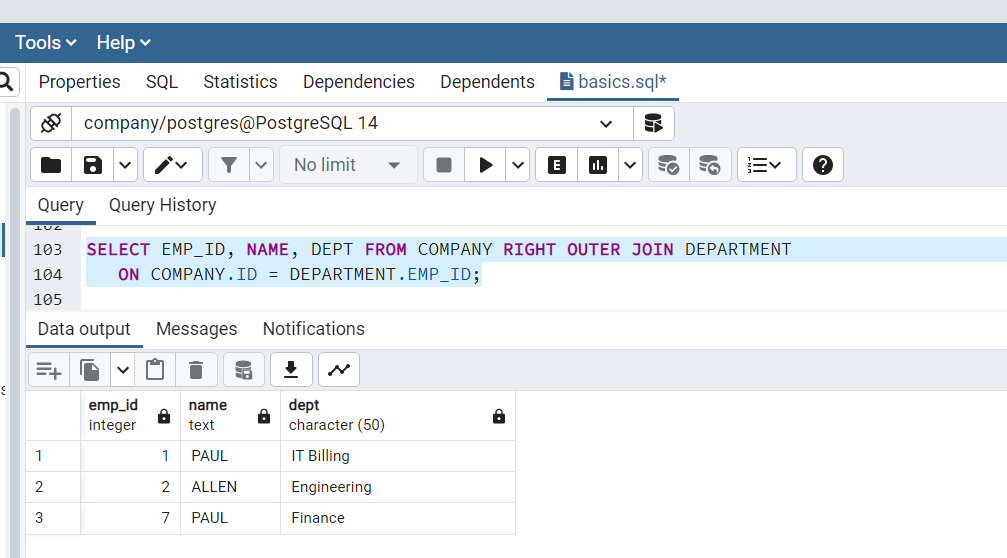
The outer join is an extension of the inner join. SQL standard defines three types of OUTER JOINS : LEFT, RIGHT, FULL

* LEFT OUTER JOIN: In case of left-outer join, an inner join is performed first. Then, for each row in table1 that does not satisfy the join condition with any row in table 2, joined row is added with null values in columns of table2 .



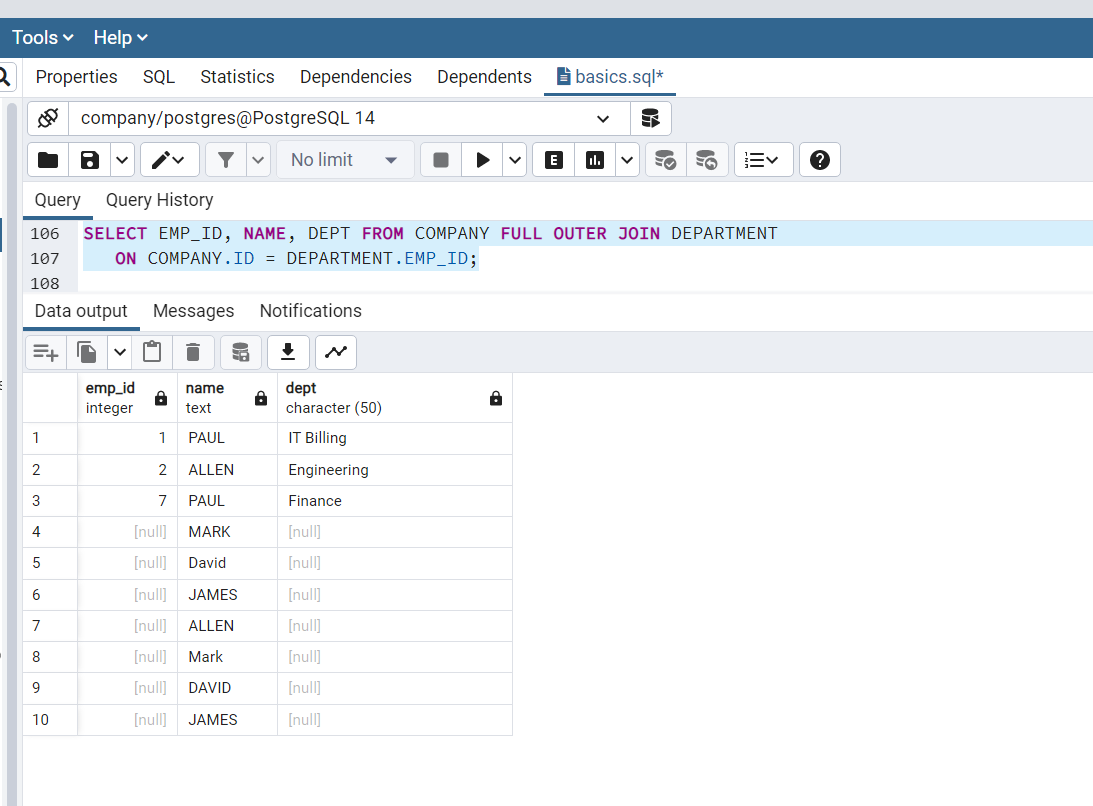
* RIGHT OUTER JOIN:

First, an inner join is performed. Then, for each row in table2 that does not satisfy the join condition with any row in table 1, joined row is added with null values in columns of table1.



* FULL OUTER JOIN:

First, an inner join is performed. Then, for each row in table T1 that does not satisfy the join condition with any row in table T2, a joined row is added with null values in columns of T2. In addition, for each row of T2 that does not satisfy the join condition with any row in T1, a joined row with null values in the columns of T1 is added.

****

postgreSQL - Unions clause:

The postgreSQL UNION clause/operator is used to combine the results of two or more SELECT statements without returning any duplicate rows.

Syntax:

SELECT column1 [, column2 ]

FROM table1 [, table2 ]

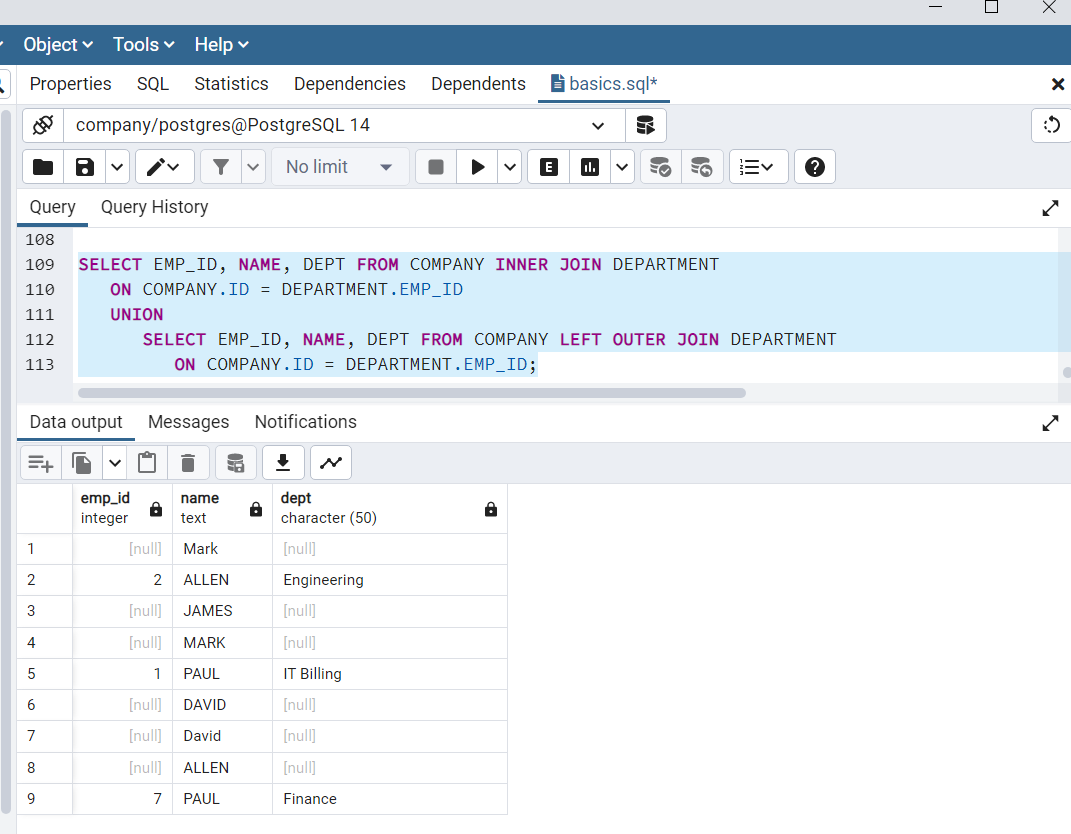
[WHERE condition]

UNION

SELECT column1 [, column2 ]

FROM table1 [, table2 ]

[WHERE condition]



UNION ALL: it will display all the duplicate rows.

Syntax:

SELECT column1 [, column2 ]

FROM table1 [, table2 ]

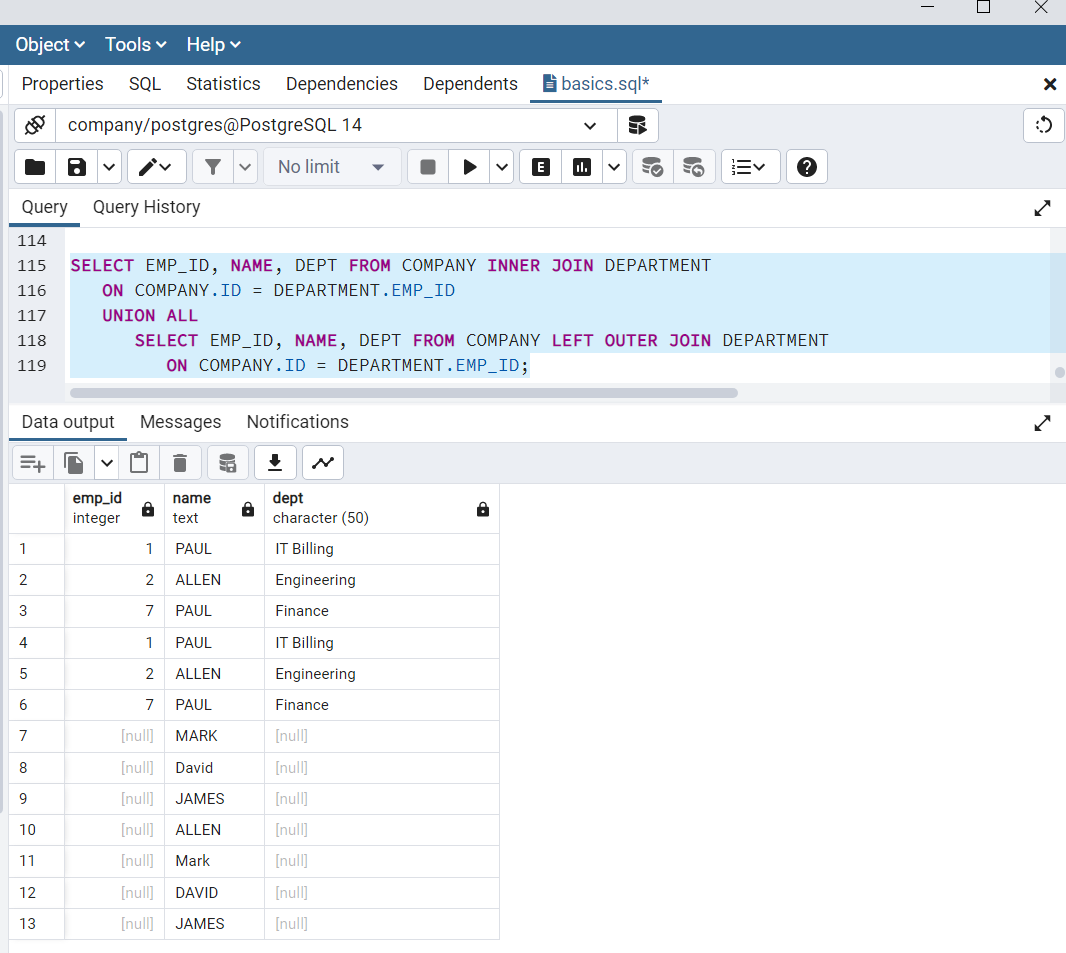
[WHERE condition]

UNION ALL

SELECT column1 [, column2 ]

FROM table1 [, table2 ]

[WHERE condition]



postgreSQL - NULL values:

This is used to represent a missing values.

Example:

UPDATE COMPANY SET ADDRESS = NULL, SALARY = NULL where ID IN(6,7);

SELECT ID, NAME, AGE, ADDRESS, SALARY

FROM COMPANY

WHERE SALARY IS NOT NULL;

SELECT ID, NAME, AGE, ADDRESS, SALARY

FROM COMPANY

WHERE SALARY IS NULL;

**postgreSQL - ALIAS:**

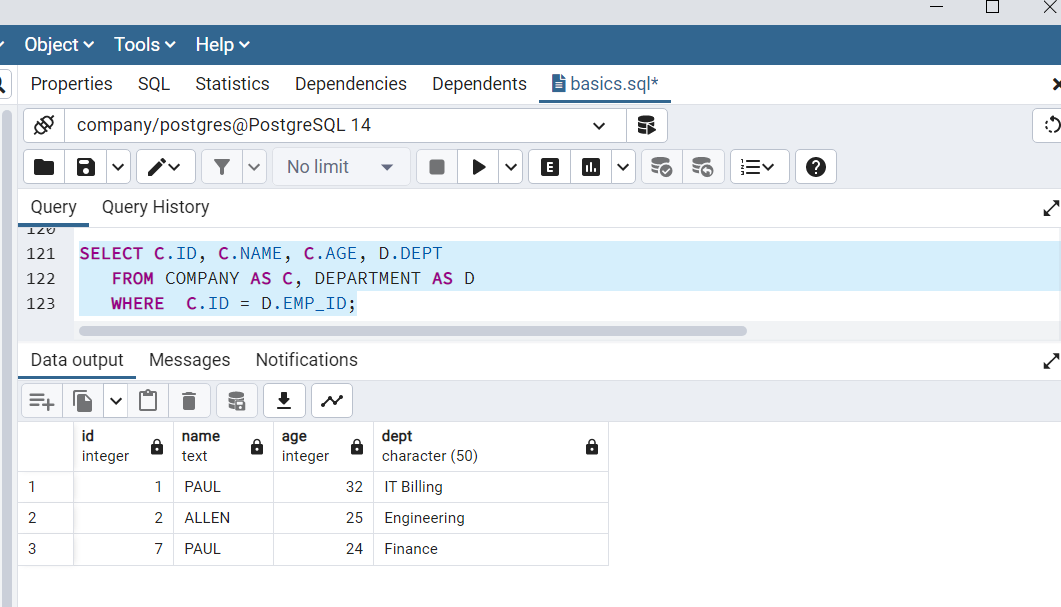
It is temporary name given to a table or column.

**Syntax:**

SELECT column1, column2....

FROM table\_name AS alias\_name

WHERE [condition];



**postgreSQL - Triggers:**

postgreSQl triggers are database callback functions, which are automatically performed/invoked when a specified database event occurs.

**postgreSQL- Indexes:**

Indexes are special lookup tables that the database search engine can be used to speed up dta retrieval.

Syntax:

CREATE INDEX index\_name ON table\_name;

Or

CREATE INDEX index\_name ON table\_name (column1\_name, column2\_name);

CREATE UNIQUE INDEX index\_name on table\_name (column\_name);

Or

CREATE INDEX index\_name on table\_name (conditional\_expression);

DROP INDEX index\_name;

**postgreSQL - ALTER TABLE command:**

* **Alter table**

ALTER TABLE table\_name ADD column\_name datatype;

* **Drop cloumn**

ALTER TABLE table\_name DROP COLUMN column\_name;

* **Data type**

ALTER TABLE table\_name ALTER COLUMN column\_name TYPE datatype;

* **Not Null:**

ALTER TABLE table\_name MODIFY column\_name datatype NOT NULL;

* **Add Check constraint:**

ALTER TABLE table\_name ADD CONSTRAINT MyUniqueConstraint UNIQUE(column1, column2...);

* **Add primary key:**

ALTER TABLE table\_name ADD CONSTRAINT MyUniqueConstraint CHECK (CONDITION);

* **Drop constraint:**

ALTER TABLE table\_name ADD CONSTRAINT MyPrimaryKey PRIMARY KEY (column1, column2...);

* **Drop primary key:**

ALTER TABLE table\_name DROP CONSTRAINT MyUniqueConstraint;

**postgreSQL - TRUNCATE:**

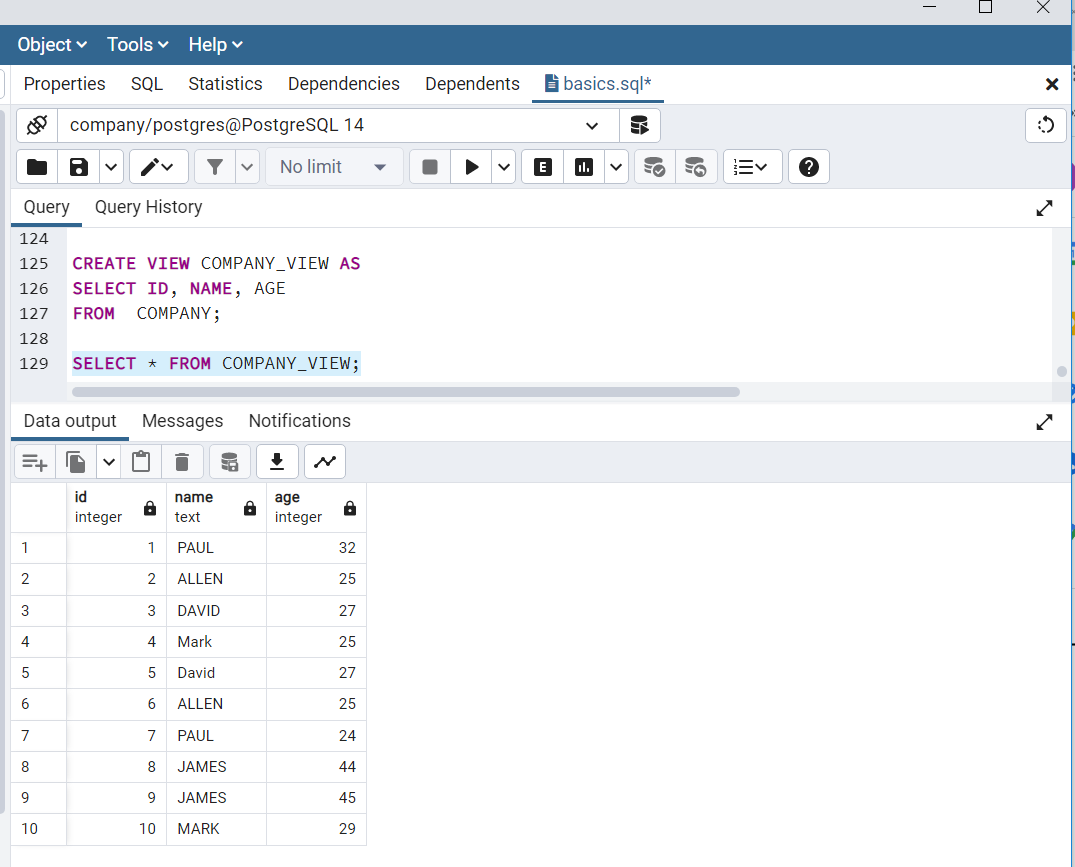
Truncate table command is used to delete complete data from an existing table.Drop table command is used to delete complete table but it would remove complete table structure from the database.

Syntax:

TRUNCATE TABLE table\_name;

**postgreSQL - views:**

Views are pseudo-tables. That is, they are not real tables; nevertheless appear as ordinary tables to SELECT. A view can represent a subset of a real table, selecting certain columns or certain rows from an ordinary table. A view can even represent joined tables.



postgreSQL-Transaction:

A transaction is a unit of work that is performed against a database. Transactions are units or sequences of work accomplished in a logical order, whether in a manual fashion by a user or automatically by some sort of a database program.

Properties of transactions:

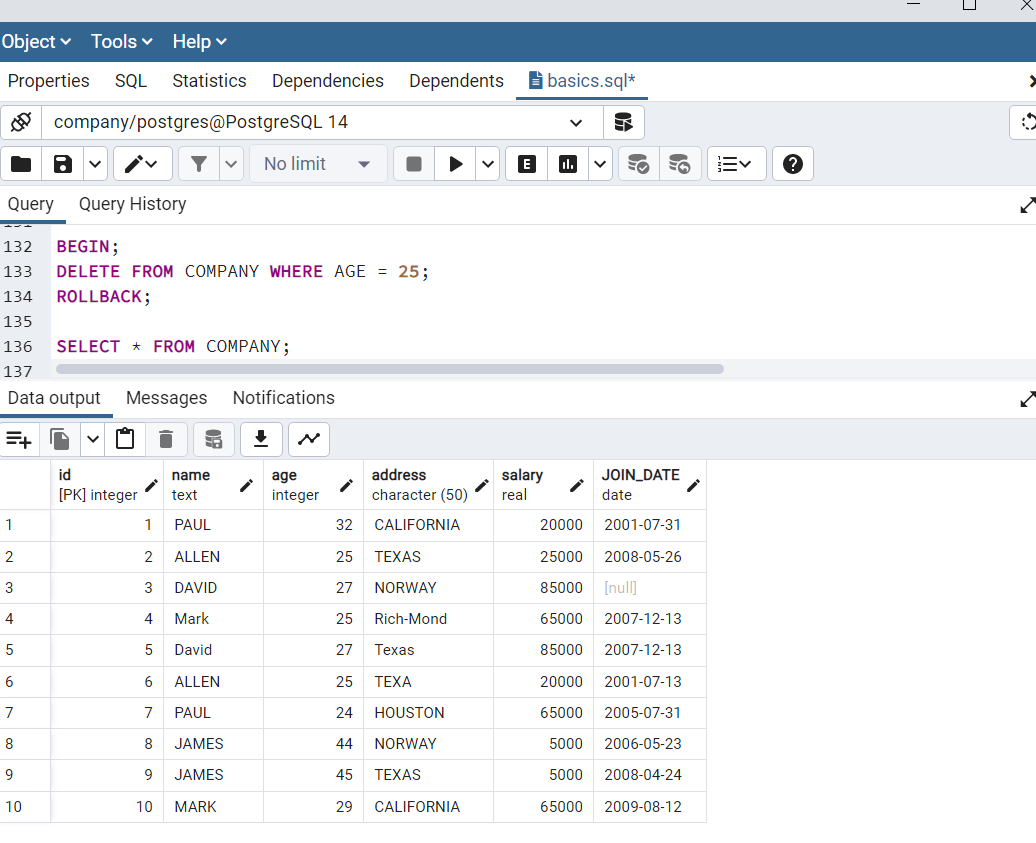
* Atomicity - Ensures that all operations within the work unit are completed successfully;
* Consistency - Ensures that the database properly changes states upon a successfully committed transaction.
* Isolation - Enables transactions to operate independently of and transparent to each other.
* Durability - Ensures that the result or effect of a committed transaction persists in case of a system failure.

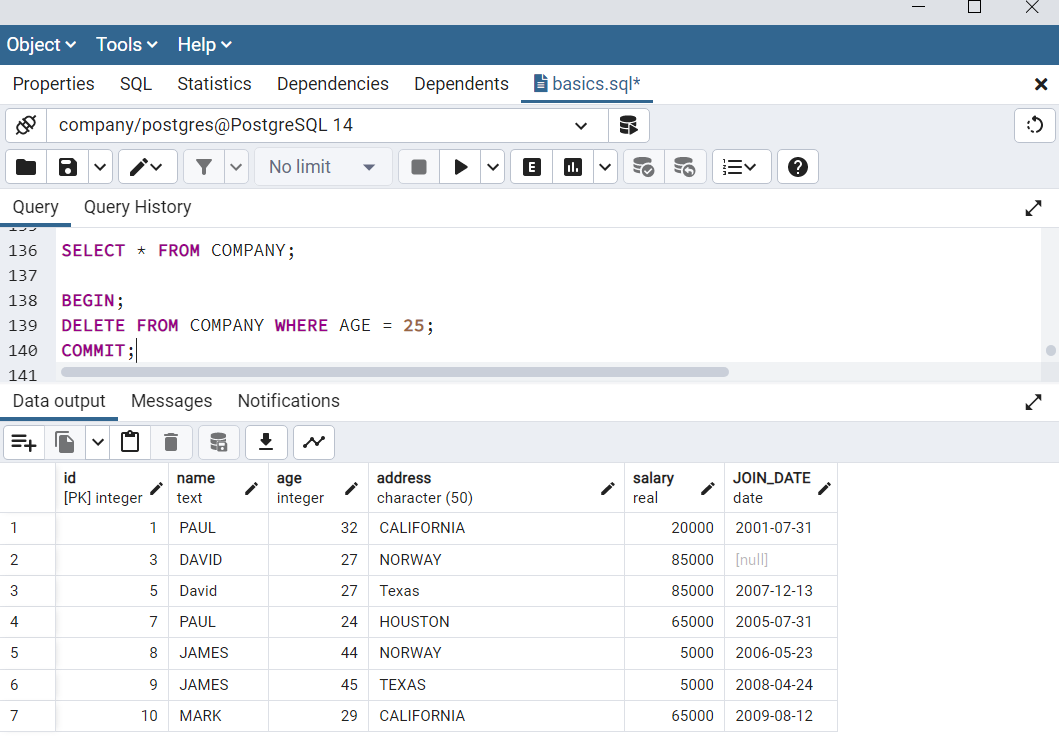
## Transaction Control

The following commands are used to control transactions −

* BEGIN TRANSACTION − To start a transaction.
* COMMIT − To save the changes, alternatively you can use END TRANSACTION command.
* ROLLBACK − To rollback the changes.

Transactional control commands are only used with the DML commands INSERT, UPDATE and DELETE only.





In case of rollback the transaction will not be saved so to save changes commit should be used.

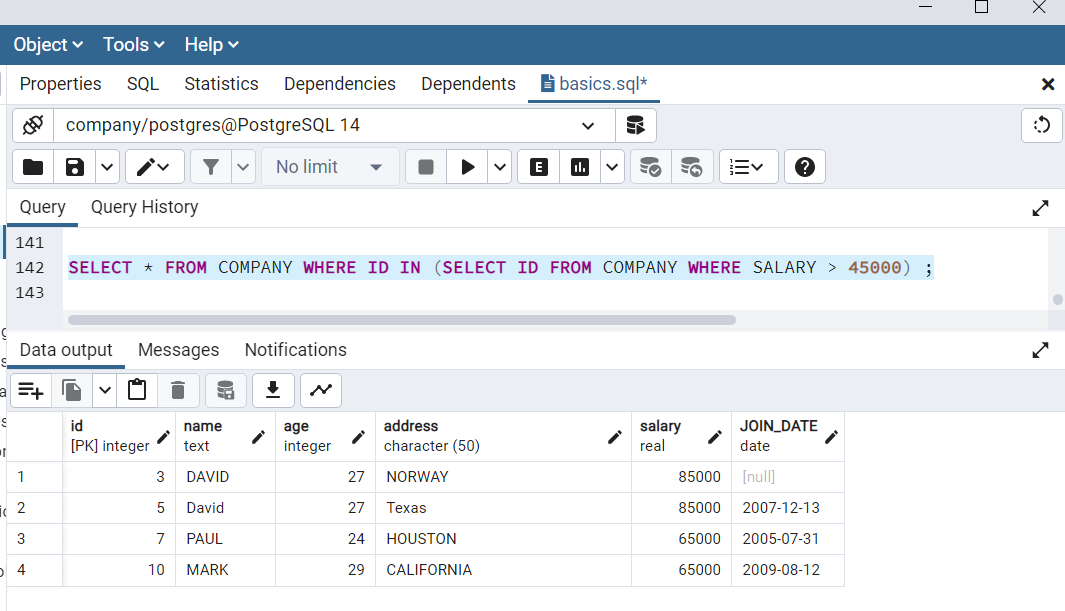
postgreSQL - locks:

Locks or exclusive locks or write locks prevent users from modifying a row or an entire table.

postgreSQL - subquery:

A subquery or inner query or nested query is a query within another postgreSQL query and embedded within the WHERE clause.

Subqueries with the SELECT statement:



Subqueries with the INSERT statement:

Syntax:

INSERT INTO COMPANY\_BKP

SELECT \* FROM COMPANY

WHERE ID IN (SELECT ID

FROM COMPANY) ;

Subqueries with the UPDATE statement:

Syntax:

UPDATE table

SET column\_name = new\_value

[ WHERE OPERATOR [ VALUE ]

(SELECT COLUMN\_NAME

FROM TABLE\_NAME)

[ WHERE) ]

Subqueries with the DELETE statement:

Syntax:

DELETE FROM TABLE\_NAME

[ WHERE OPERATOR [ VALUE ]

(SELECT COLUMN\_NAME

FROM TABLE\_NAME)

[ WHERE) ]

postgreSQL- AUTO INCREMENT:

Syntax:

CREATE TABLE table\_name (

Column SERIAL

);

Example:

CREATE TABLE COMPANY(

ID SERIAL PRIMARY KEY, --> here id will be incremented automatically

NAME TEXT NOT NULL,

AGE INT NOT NULL,

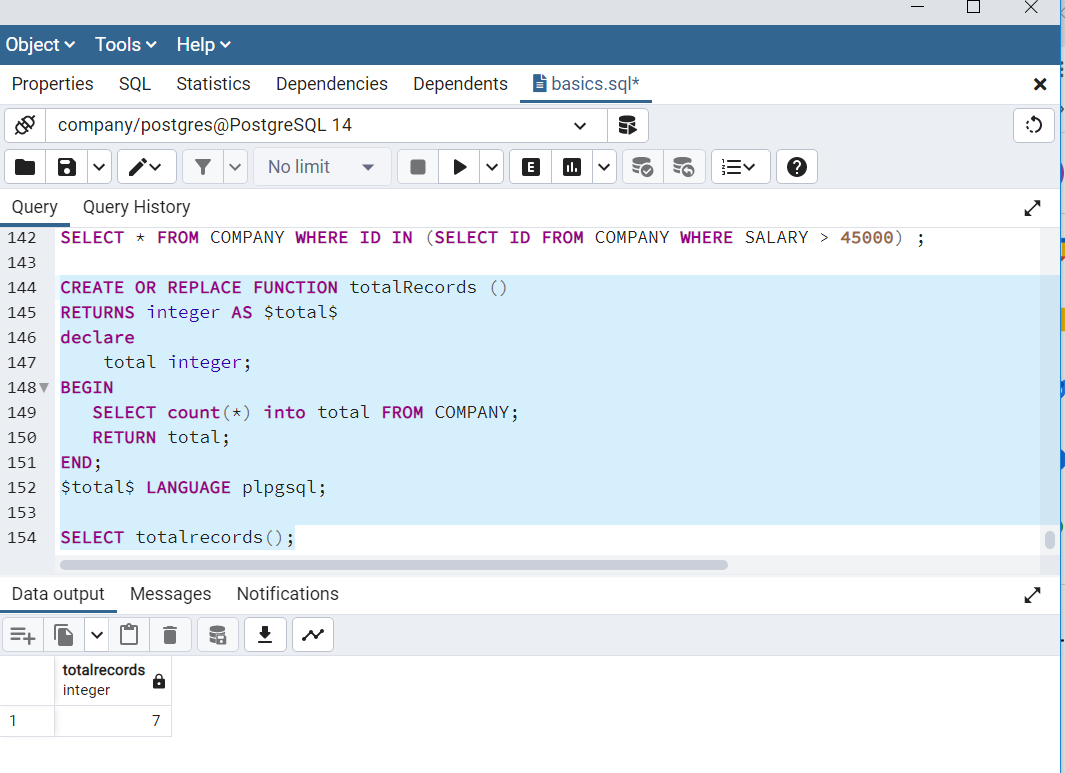
ADDRESS CHAR(50),

SALARY REAL

);

postgreSQL-Functions

They are also known as stord procedures, that allows to carry out operations that would normally take several queries and round trips in a single function within the database.



* Count function-

SELECT COUNT(\*) FROM COMPANY ;

* Max function-

SELECT MAX(salary) FROM COMPANY;

* Min function-

SELECT MIN(salary) FROM company;

* Avg function-

SELECT AVG(SALARY) FROM COMPANY;

* Sum function-

SELECT SUM(salary) FROM company;

* Array function-

SELECT ARRAY\_AGG(SALARY) FROM COMPANY;