What is PostgreSQL?

PostgreSQL (pronounced as **post-gress-Q-L**) is an open source relational database management system (DBMS) developed by a worldwide team of volunteers. PostgreSQL is not controlled by any corporation or other private entity and the source code is available free of charge.

A Brief History of PostgreSQL

PostgreSQL, originally called Postgres, was created at UCB by a computer science professor named Michael Stonebraker. Stonebraker started Postgres in 1986 as a follow-up project to its predecessor, Ingres, now owned by Computer Associates.

1. 1977-1985: A project called INGRES was developed.
   - Proof-of-concept for relational databases
   - Established the company Ingres in 1980
   - Bought by Computer Associates in 1994
2. 1986-1994: POSTGRES
   - Development of the concepts in INGRES with a focus on object orientation and the query language - Quel
   - The code base of INGRES was not used as a basis for POSTGRES
   - Commercialized as Illustra (bought by Informix, bought by IBM)
3. 1994-1995: Postgres95
   - Support for SQL was added in 1994

Key Features of PostgreSQL

PostgreSQL runs on all major operating systems, including Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), and Windows. It supports text, images, sounds, and video, and includes programming interfaces for C / C++, Java, Perl, Python, Ruby, Tcl and Open Database Connectivity (ODBC).

PostgreSQL supports a large part of the SQL standard and offers many modern features including the following:

- Complex SQL queries
- SQL Sub-selects
- Foreign keys
- Trigger
- Views
- Transactions
- Multiversion concurrency control (MVCC)
- Streaming Replication (as of 9.0)
- Hot Standby (as of 9.0)

You can check official documentation of PostgreSQL to understand the above-mentioned features. PostgreSQL can be extended by the user in many ways. For example, by adding new:

- Data types
- Functions
- Operators
- Aggregate functions
- Index methods

Installing PostgreSQL:

Ubuntu-Linux:

Open terminal and follow the below steps:

- sudo apt update
- apt install postgresql postgresql-contrib
- sudo systemctl start postgresql.service (for ensure server is running)

Accessing Postgres and Creating Data Base:

Open terminal and follow the below steps:

- psql postgres -- (to log into postgres)
- \l -- (to see the list of data-base or info of postgres)

Creating an **employee** data-base with **empuser** as user and **emppass** as password and with all permisions

- create database employee;
- create user empuser with encrypted password 'emppass';
- grant all privileges on database employee to empuser;

After above steps the changes will be similar to below image.

```
Last login: Mon Jul 25 21:03:04 on ttys011
[redacted] postgres % psql postgres
psql (14.4)
Type "help" for help.

postgres=# \l
                                  List of databases
   Name     |  Owner   | Encoding | Collate | Ctype |   Access privileges
-----------+----------+----------+---------+-------+-----------------------
 postgres  | [redacted] | UTF8     | C       | C     |
 template0 |          | UTF8     | C       | C     | [redacted]
           |          |          |         |       |
 template1 |          | UTF8     | C       | C     |
           |          |          |         |       |
(3 rows)

postgres=# create database employee;
CREATE DATABASE
postgres=# create user empuser with encrypted password 'emppass';
CREATE ROLE
postgres=# grant all privileges on database employee to empuser;
GRANT
postgres=# \l
                                  List of databases
   Name     |  Owner   | Encoding | Collate | Ctype |   Access privileges
-----------+----------+----------+---------+-------+-----------------------
 employee  | [redacted] | UTF8     | C       | C     | [redacted]
           |          |          |         |       |
           |          |          |         |       |
 postgres  |          | UTF8     | C       | C     |
 template0 |          | UTF8     | C       | C     |
           |          |          |         |       |
 template1 |          | UTF8     | C       | C     |
           |          |          |         |       |
(4 rows)

postgres=# \c employee
You are now connected to database "employee" as user [redacted].
employee=#
```

Command for accessing data-base: \c employee (\c data-base name)

Command to drop data-base: DROP DATABASE [IF EXISTS] employee (DROP DATABASE [IF EXISTS] data-base name)

For reference: Link

Accessing PostgreSQL through DBeaver:

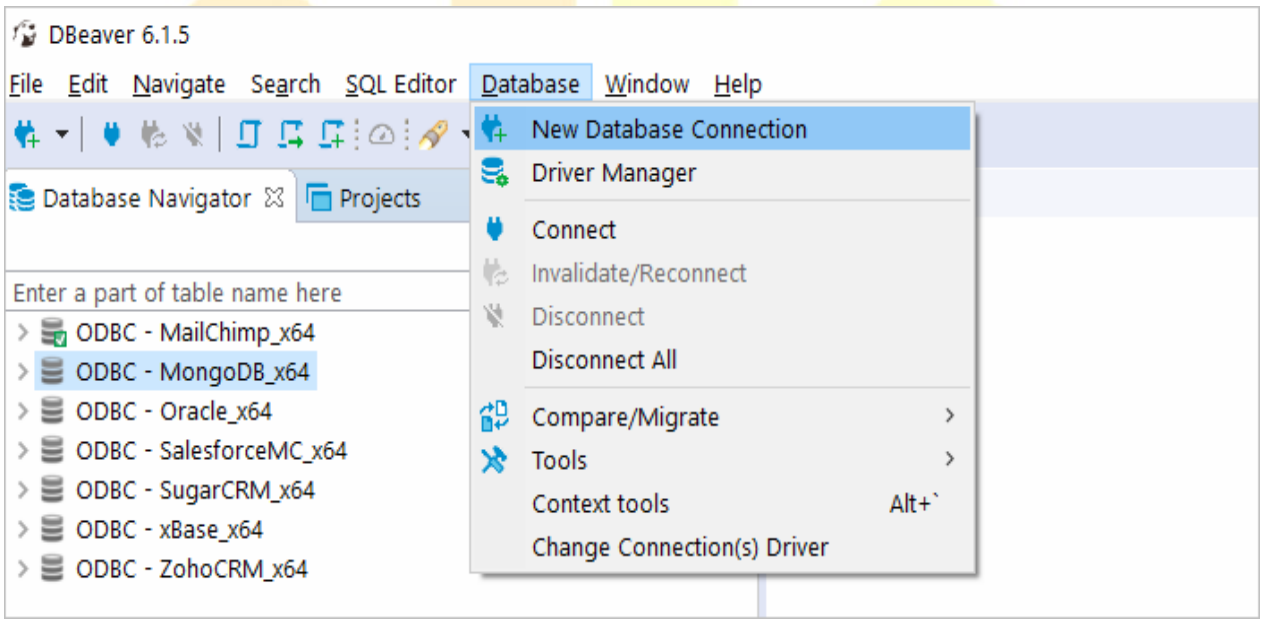
About DBeaver :

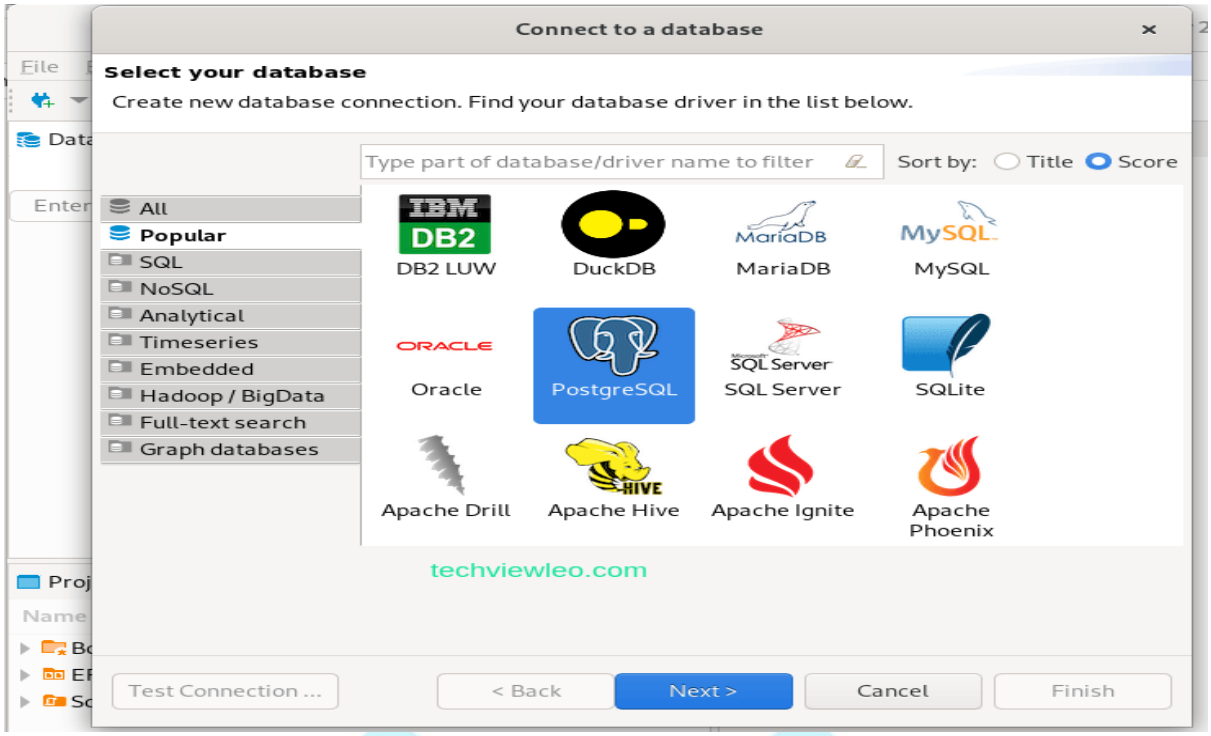
- Free multi-platform database tool for developers, database administrators, analysts and all people who need to work with databases. Supports all popular databases: MySQL, PostgreSQL, SQLite, Oracle, DB2, SQL Server, Sybase, MS Access, Teradata, Firebird, Apache Hive, Phoenix, Presto, etc.

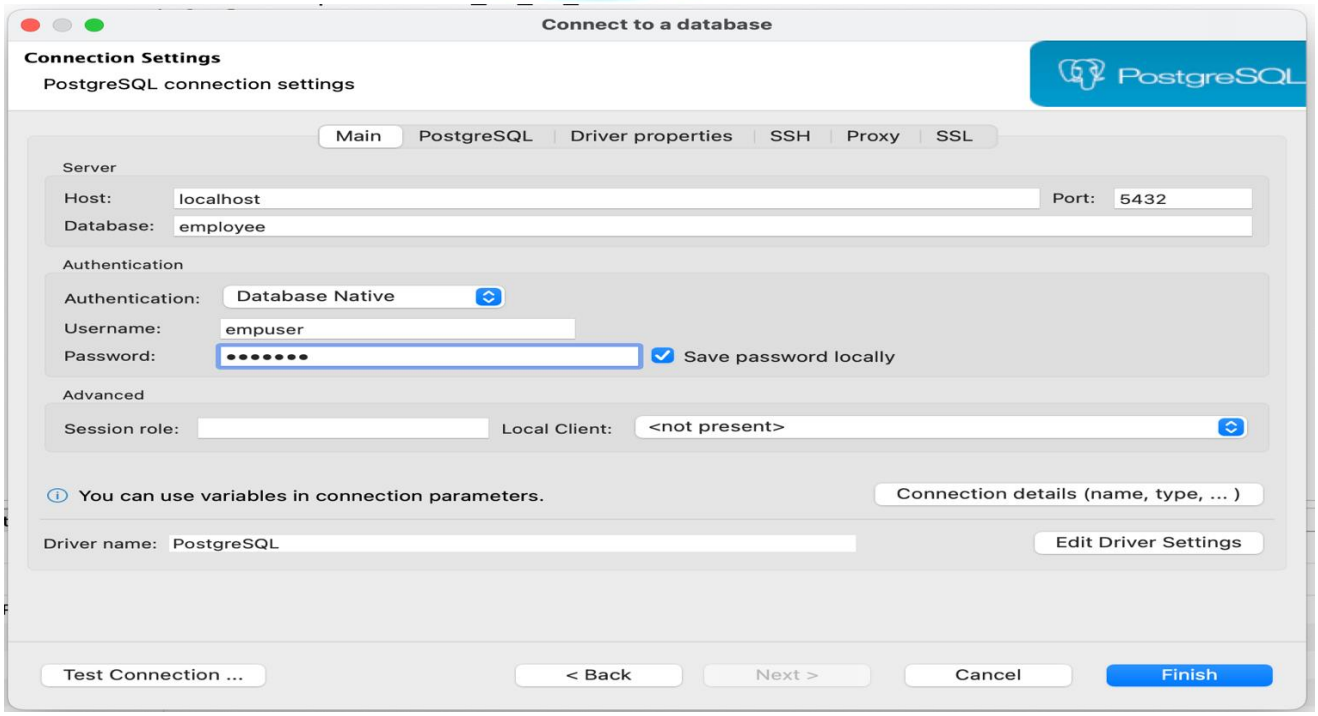
Installing DBeaver:

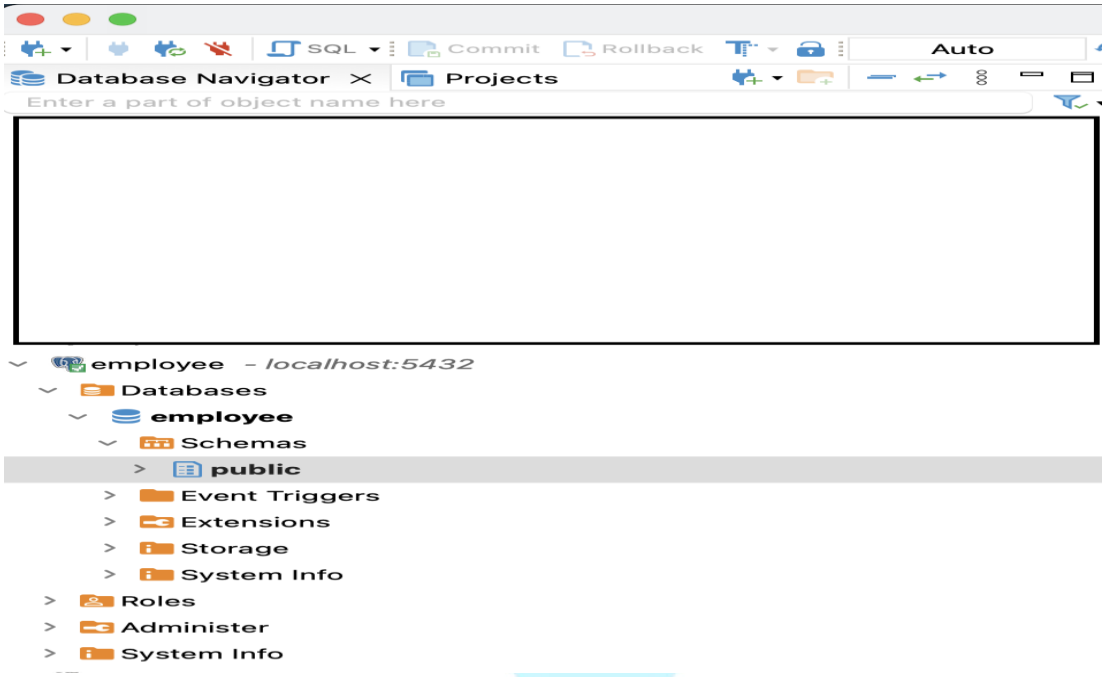
Open terminal and follow the below steps:

- sudo apt update
- sudo snap install dbeaver-ce (note : this will take some time to install)
- Open DBeaver once install and follow the below steps shown in images




Connect to a database
Select your database
Create new database connection. Find your database driver in the list below.
Type part of database/driver name to filter
Sort by: Title Score
All
Popular
SQL
NoSQL
Analytical
Timeseries
Embedded
Hadoop / BigData
Full-text search
Graph databases
DB2 LUW
DuckDB
MariaDB
MySQL
Oracle
PostgreSQL
SQL Server
SQLite
Apache Drill
Apache Hive
Apache Ignite
Apache Phoenix
techviewleo.com
Test Connection ...
< Back
Next >
Cancel
Finish


Connect to a database
Connection Settings
PostgreSQL connection settings
PostgreSQL
Main
PostgreSQL
Driver properties
SSH
Proxy
SSL
Server
Host: localhost
Port: 5432
Database: employee
Authentication
Authentication: Database Native
Username: empuser
Password: •••••••
Save password locally
Advanced
Session role:
Local Client: <not present>
You can use variables in connection parameters.
Connection details (name, type, ...)
Driver name: PostgreSQL
Edit Driver Settings
Test Connection ...
< Back
Next >
Cancel
Finish



Employee Data-Base creation is done.

Creating Tables:

The PostgreSQL CREATE TABLE statement is used to create a new table in any of the given database.

Syntax:

Basic syntax of CREATE TABLE statement is as follows −

CREATE TABLE table_name(

column1 datatype,

column2 datatype,

column3 datatype,

.....

columnN datatype,

PRIMARY KEY(one or more columns)

);

CREATE TABLE is a keyword, telling the database system to create a new table. The unique name or identifier for the table follows the CREATE TABLE statement. Initially, the empty table in the current database is owned by the user issuing the command.

Then, in brackets, comes the list, defining each column in the table and what sort of data type it is. The syntax will become clear with an example given below.

```
CREATE TABLE employee (
   id bigint NOT NULL,
   birth_date date NOT NULL,
   first_name character varying(14) NOT NULL,
   last_name character varying(16) NOT NULL,
   gender employee_gender NOT NULL,
   hire_date date NOT NULL
);
CREATE TABLE department_employee (
   employee_id bigint NOT NULL,
   department_id character(4) NOT NULL,
   from_date date NOT NULL,
   to_date date NOT NULL
);
```

Dropping tables:

- The PostgreSQL "DROP TABLE" statement is used to remove a table definition and all associated data, indexes, rules, triggers, and constraints for that table. once a table is deleted then all the information available in the table would also be lost forever.

Syntax:

- Basic syntax of DROP TABLE statement is as follows:

  DROP TABLE table_name;

- We can drop the single or multiple tables at a time.

- Dropping single table:

  DROP TABLE employee;

- Dropping multiple tables:

  DROP TABLE employee, department;

After drop command which command, I must write.

Schema:

- A **Schema** is a named collection of tables. A schema can also contain views, indexes, sequences, data types, operators, and functions. Schemas are analogous to directories at the operating system level, except those schemas cannot be nested.

Syntax:

- The basic syntax of create schema is as follows:
  CREATE SCHEMA name;

- Note: here **name** is the name of the schema.

- The basic syntax to create table in schema is as follows:

Syntax:

```
CREATE TABLE myschema.table_name(
column1 datatype,
column2 datatype,
column3 datatype,
.....
column datatype,
PRIMARY KEY( one or more columns )
);
```

Example:

```
CREATE TABLE myschema.employee (
   id bigint NOT NULL,
   birth_date date NOT NULL,
   first_name character varying(14) NOT NULL,
   last_name character varying(16) NOT NULL,
   gender employee_gender NOT NULL,
   hire_date date NOT NULL
);
```

- To drop a schema if It is empty (all objects in it have been dropped), use the below command:

Syntax:

DROP SCHEMA schema_name;

- To drop a schema including all contained objects, use the below command:

Syntax:

DROP SCHEMA schema_name CASCADE

Inserting into tables:

The PostgrSQL **INSERT INTO** statement allows one to insert new rows into a table. One can insert a single row at a time or several rows as a result of a query.

Syntax:

INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)

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

Example:

INSERT INTO employee(id, birth_date, first_name, last_name, gender, hire_data)

VALUES (100001, 1953-02-09, 'Georgi', 'Facello', 'M', 1988-02-06);

- The following example inserts multiple rows using the multirow VALUES syntax :

Example:

INSERT INTO employee(id, birth_date, first_name, last_name, gender, hire_data)

VALUES (10001, ‘1953-02-09’, ‘Georgi’, ‘Facello’, ‘M’, ‘1988-02-06’),

(10002, ‘1964-02-06’, ‘Bezalel’, ’Simmel’, ‘F’, ‘1986-09-11’),

(10003, ‘1959-03-12’, ‘Parto’, ‘Bamford’, ‘M’, ‘1988-04-08’);

Selecting from table:

- PostgreSQL **SELECT** statement is used to fetch the data from a database table, which returns data in the form of result table. These results are called result-sets.

Syntax:

SELECT column1, column2, columnN FROM table_name;

- Here, column1, column2...are the fields of a table, whose values you want to fetch. If you want to fetch all the fields available in the field then you can use the following syntax:

Syntax:

SELECT * FROM table_name;

Example:

SELECT id, first_name, gender, hire_date from employee;

	123 id	ABC first_name	ABC gender	hire_date
1	10,001	Georgi	M	1988-02-06
2	10,002	Bezalel	F	1986-09-11
3	10,003	Parto	M	1988-04-08
4	10,004	Chirstian	M	1986-01-12
5	10,005	Kyoichi	M	1989-12-09
6	10,006	Anneke	F	1989-02-06

Example:

SELECT * from employee;

	id	birth_date	first_name	last_name	gender	hire_date
1	10,001	1953-02-09	Georgi	Facello	M	1988-02-06
2	10,002	1964-02-06	Bezalel	Simmel	F	1986-09-11
3	10,003	1959-03-12	Parto	Bamford	M	1988-04-08
4	10,004	1954-01-05	Chirstian	Koblick	M	1986-01-12
5	10,005	1956-09-01	Kyoichi	Maliniak	M	1989-12-09
6	10,006	1954-08-04	Anneke	Preusig	F	1989-02-06

Using operators:

- An operator is a reserved word, or a character used primarily in a PostgreSQL statement's WHERE clause to perform operations, such as comparisons and arithmetic operations. Operators are used to specify conditions in a PostgreSQL statement and to serve as conjunctions for multiple conditions in a statement.

  - Arithmetic operators
  - Comparison operators
  - Logical operators
  - Bitwise operators

Arithmetic operators:

Addition:

- Addition adds the values on either side of the operator.

Example:

SELECT employee_id, from_date, amount+100 as "amount+100" FROM salary;

	employee_id	from_date	amount+100
1	10,001	1988-02-06	61,117
2	10,001	1989-02-06	63,102
3	10,001	1990-01-06	67,074
4	10,001	1991-01-06	67,596
5	10,001	1992-01-06	67,961
6	10,001	1993-01-06	72,046

Subtraction:

- Subtracts right hand operand from left hand operand.

Example:

select employee_id, from_date, amount-100 as "amount-100" from salary;

	employee_id	from_date	amount-100
1	10,001	1988-02-06	59,117
2	10,001	1989-02-06	61,102
3	10,001	1990-01-06	65,074
4	10,001	1991-01-06	65,596
5	10,001	1992-01-06	65,961
6	10,001	1993-01-06	70,046

Multiplication:

- Multiplies values on either side of the operator

Example:

select employee_id, from_date, amount*2 as "amount*2" from salary;

	employee_id	from_date	amount*2
1	10,001	1988-02-06	120,234
2	10,001	1989-02-06	124,204
3	10,001	1990-01-06	132,148
4	10,001	1991-01-06	133,192
5	10,001	1992-01-06	133,922
6	10,001	1993-01-06	142,092

Division:

- Divides left hand operand by right hand operand

Example:

```
select employee_id, from_date, amount/10 as "amount/10" from salary;
```

	employee_id	from_date	amount/10
1	10,001	1988-02-06	6,011
2	10,001	1989-02-06	6,210
3	10,001	1990-01-06	6,607
4	10,001	1991-01-06	6,659
5	10,001	1992-01-06	6,696
6	10,001	1993-01-06	7,104

,mqnegtikj90o`

Comparison operators:

Equal (=):

- Checks if the values of two operands are equal or not, if yes then condition becomes true.

Example:

```
select * from salary where amount=61000;
```

	employee_id	amount	from_date	to_date
1	14,114	61,000	1997-08-11	1998-08-11
2	18,116	61,000	2001-10-03	2002-10-03
3	18,209	61,000	2001-09-04	2002-09-04
4	18,420	61,000	1999-02-04	2000-02-04
5	23,544	61,000	2001-07-01	2002-06-01
6	23,608	61,000	1986-08-12	1987-08-12

Not equal (!=):

- Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.

Example:

```
select * from salary where amount!=61000;
```

Less than (<):

- Checks if the value of right operand is greater than the value of left operand, if yes then condition becomes true.

Example:

select * from salary where amount<70000;

	employee_id	amount	from_date	to_date
1	10,001	60,117	1988-02-06	1989-02-06
2	10,001	62,102	1989-02-06	1990-01-06
3	10,001	66,074	1990-01-06	1991-01-06
4	10,001	66,596	1991-01-06	1992-01-06
5	10,001	66,961	1992-01-06	1993-01-06
6	10,002	65,828	1996-03-08	1997-03-08

Greater than (>):

- Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.

Example:

select * from salary where amount>70000;

	employee_id	amount	from_date	to_date
1	10,001	71,046	1993-01-06	1993-12-06
2	10,001	74,333	1993-12-06	1994-12-06
3	10,001	75,286	1994-12-06	1995-12-06
4	10,001	75,994	1995-12-06	1996-12-06
5	10,001	76,884	1996-12-06	1997-11-06
6	10,001	80,013	1997-11-06	1998-11-06

Less than or equal to (<=):

- Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.

Example:

```
select * from salary where amount<=70000;
```

	employee_id	amount	from_date	to_date
1	10,001	60,117	1988-02-06	1989-02-06
2	10,001	62,102	1989-02-06	1990-01-06
3	10,001	66,074	1990-01-06	1991-01-06
4	10,001	66,596	1991-01-06	1992-01-06
5	10,001	66,961	1992-01-06	1993-01-06
6	10,002	65,828	1996-03-08	1997-03-08

Greater than or equal to (>=):

- Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.

Example:

```
select * from salary where amount>=70000;
```

	employee_id	amount	from_date	to_date
1	10,001	71,046	1993-01-06	1993-12-06
2	10,001	74,333	1993-12-06	1994-12-06
3	10,001	75,286	1994-12-06	1995-12-06
4	10,001	75,994	1995-12-06	1996-12-06
5	10,001	76,884	1996-12-06	1997-11-06
6	10,001	80,013	1997-11-06	1998-11-06

Not equal (<>):

- Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.

Example:

```
select * from salary where amount <> 60000;
```

	employee_id	amount	from_date	to_date
1	10,001	60,117	1988-02-06	1989-02-06
2	10,001	62,102	1989-02-06	1990-01-06
3	10,001	66,074	1990-01-06	1991-01-06
4	10,001	66,596	1991-01-06	1992-01-06
5	10,001	66,961	1992-01-06	1993-01-06
6	10,001	71,046	1993-01-06	1993-12-06

Example:

```
select * from department_employee where department_id <> 'd005';
```

	123 employee_id	Ctrl+click to open SQL console	m_date	to_date
1	10,002	d007	1996-03-08	1999-01-01
2	10,003	d004	1995-03-12	1999-01-01
3	10,004	d004	1986-01-12	1999-01-01
4	10,005	d003	1989-12-09	1999-01-01
5	10,007	d008	1989-10-02	1999-01-01
6	10,009	d006	1986-06-02	1999-01-01

Logical operators:

- The AND, OR, and NOT keywords are PostgreSQL's Boolean operators. These keywords are mostly used to join or invert conditions in a SQL statement, specifically in the WHERE clause and the HAVING clause.

AND:

- The AND operator allows the existence of multiple conditions in a PostgresSQL statement's WHERE clause.

Example:

select * from department_employee where department_id = 'd005' AND from_date > '1988-02-06';

	123 employee_id	RBC department_id	from_date	to_date
1	10,006	d005	1990-05-08	1999-01-01
2	10,008	d005	1998-11-03	2002-07-07
3	10,012	d005	1993-06-12	1999-01-01
4	10,014	d005	1995-05-12	1999-01-01
5	10,021	d005	1988-10-02	2003-03-07
6	10,022	d005	1999-03-09	1999-01-01

NOT:

- The NOT operator reverses the meaning of the logical operator with which it is used. Eg. NOT EXISTS, NOT BETWEEN, NOT IN etc. This is negate operator.

Example:

select * from department_employee where not department_id = 'd005';

	123 employee_id	ABC department_id	from_date	to_date
1	10,002	d007	1996-03-08	1999-01-01
2	10,003	d004	1995-03-12	1999-01-01
3	10,004	d004	1986-01-12	1999-01-01
4	10,005	d003	1989-12-09	1999-01-01
5	10,007	d008	1989-10-02	1999-01-01
6	10,009	d006	1986-06-02	1999-01-01

OR:

- The OR operator is used to combine multiple conditions in a PostgresSQL statement's WHERE clause.

Example:

select * from department_employee where department_id = 'd005' or department_id = 'd004';

	123 employee_id	ABC department_id	from_date	to_date
1	10,001	d005	1988-02-06	1999-01-01
2	10,003	d004	1995-03-12	1999-01-01
3	10,004	d004	1986-01-12	1999-01-01
4	10,006	d005	1990-05-08	1999-01-01
5	10,008	d005	1998-11-03	2002-07-07
6	10,010	d004	1997-12-11	2002-02-06

Expressions:

- An expression is a combination of one or more values, operators, and PostgreSQL functions that evaluate to a value. PostgreSQL expressions are like formulas and are written in query language. You can also use to query the database for specific set of data.

Syntax:

```
SELECT column1, column2, columnN
FROM table_name
WHERE [CONDITION | EXPRESSION];
```

Example:

```
select * from department_employee where from_date > '1988-02-06' and
department_id = 'd005';
```

	123 employee_id	ABC department_id	from_date	to_date
1	10,006	d005	1990-05-08	1999-01-01
2	10,008	d005	1998-11-03	2002-07-07
3	10,012	d005	1993-06-12	1999-01-01
4	10,014	d005	1995-05-12	1999-01-01
5	10,021	d005	1988-10-02	2003-03-07
6	10,022	d005	1999-03-09	1999-01-01

WHERE clause:

- The PostgreSQL WHERE clause is used to specify a condition while fetching the data from single table or joining with multiple tables. If one condition is satisfied, only then it returns specific value from the table. You can filter out rows that you do not want included in the result-set by using the WHERE clause. The WHERE clause not only used in SELECT, but it is also used in UPDATE, DELETE statement etc.

Syntax:

```
SELECT column1, column2, columnN
FROM table_name
WHERE [search_condition]
```

Example:

```
select * from employee where gender = 'F';
```

	id	birth_date	first_name	last_name	gender	hire_date
1	10,002	1964-02-06	Bezalel	Simmel	F	1986-09-11
2	10,006	1954-08-04	Anneke	Preusig	F	1989-02-06
3	10,007	1958-11-05	Tzvetan	Zielinski	F	1989-10-02
4	10,009	1953-07-04	Sumant	Peac	F	1986-06-02
5	10,010	1963-01-06	Duangkaew	Piveteau	F	1990-12-08
6	10,011	1953-07-11	Mary	Sluis	F	1991-10-01

Example:

```
select * from employee where gender = 'M' and birth_date > '1953-02-09';
```

	id	birth_date	first_name	last_name	gender	hire_date
1	10,003	1959-03-12	Parto	Bamford	M	1988-04-08
2	10,004	1954-01-05	Chirstian	Koblick	M	1986-01-12
3	10,005	1956-09-01	Kyoichi	Maliniak	M	1989-12-09
4	10,008	1959-07-02	Saniya	Kalloufi	M	1995-03-09
5	10,012	1960-04-10	Patricio	Bridgland	M	1993-06-12
6	10,013	1963-07-06	Eberhardt	Terkki	M	1986-08-10

UPDATE query:

The PostgreSQL UPDATE query is used to modify the existing records in a table. You can use WHERE clause with UPDATE query to update the selected rows. Otherwise, all the rows would be updated.

Syntax:

```
UPDATE table_name
SET column1 = value1, column2 = value2...., columnN = valueN
WHERE [condition];
```

Example:

```
update salary set amount = 20000 where employee_id = 10001;
```

Example:

```
update salary  set employee_id  = 10001, amount = 30000 where
employee_id  = 10002;
```

DELETE query:

- The PostgreSQL DELETE query is used to delete the existing records from a table. You can use WHERE clause with DELETE query to delete the selected rows. Otherwise, all the records would be deleted.

Syntax:

```
DELETE FROM table_name
WHERE [condition];
```

Note: You can combine N number of conditions using AND or OR operators.

Example:

```
delete from employee where id = '10001';
```

Example:

```
delete from salary where employee_id = 10001 and amount > 70000;
```

- If you want to DELETE all the records from table, you do not need to use WHERE clause.

Example:

Delete from employee;

LIKE clause:

- The PostgreSQL LIKE operator is used to match text values against a pattern using wildcards. If you search expression can be matched to the pattern expression, the LIKE will return true, which is 1.

- There are 2 wildcards used in conjunction with the LIKE operators

  - Percent sign(%)
  - Underscore(_)

- The percent sign represents 0 or 1 or multiple numbers or characters. The underscore represents a single number of character. These symbols can be used in combinations.

- If either of these two signs is not used in conjunction with the LIKE clause, then the LIKE acts like the equals operator.

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_'

Example:

```
select * from employee where first_name like 'G%';
```

	id	birth_date	first_name	last_name	gender	hire_date
1	10,015	1960-07-08	Guoxiang	Nooteboom	M	1987-02-07
2	10,055	1956-06-06	Georgy	Dredge	M	1994-03-04
3	10,063	1952-06-08	Gino	Leonhardt	F	1989-08-04
4	10,075	1960-09-03	Gao	Dolinsky	F	1988-07-03
5	10,121	1963-02-07	Guoxiang	Ramsay	M	1989-03-05
6	10,124	1963-11-05	Geraldo	Marwedel	M	1991-05-09
7	10,133	1963-12-12	Giri	Isaak	M	1986-03-12

Example:

```
select * from employee where first_name like 'Gen%';
```

	id	birth_date	first_name	last_name	gender	hire_date
1	10,219	1952-02-05	Genta	Kolvik	M	1995-07-03
2	id: int8 0	1963-12-08	Gen	Tiemann	F	1995-04-09
3	11,008	1962-11-07	Gennady	Menhoudj	M	1989-06-09
4	11,060	1955-06-06	Genevieve	Pokrovskii	F	1991-03-05
5	11,196	1959-01-05	Gennadi	Breugel	M	1993-05-11
6	11,474	1954-05-04	Gennady	Akazan	M	1992-02-07

Example:

```
select * from employee where first_name like '%dy';
```

	id	birth_date	first_name	last_name	gender	hire_date
1	10,255	1964-08-02	Roddy	Garnick	M	1993-12-05
2	10,750	1954-02-06	Roddy	Demeyer	F	1991-04-08
3	10,756	1959-12-06	Paddy	Brizzi	M	1996-09-11
4	11,008	1962-11-07	Gennady	Menhoudj	M	1989-06-09
5	11,099	1958-12-07	Woody	Spataro	M	1987-07-12
6	11,285	1963-11-02	Randy	Koshino	M	1997-03-03

Example:

```
select * from salary where amount::text like '6000%';
```

	employee_id	amount	from_date	to_date
1	83,486	60,005	1987-02-07	1988-02-07
2	87,940	60,001	1987-12-02	1988-12-02
3	107,605	60,008	1988-01-12	1988-12-12
4	22,883	60,005	1986-10-05	1987-10-05
5	30,168	60,000	1986-05-03	1987-05-03
6	64,657	60,000	1988-02-05	1989-01-05
7	64,861	60,004	1987-12-09	1988-12-09

Example:

```
select * from employee where first_name like '%eo%';
```

	id	birth_date	first_name	last_name	gender	hire_date
1	10,032	1960-09-08	Jeong	Reistad	F	1991-08-06
2	10,055	1956-06-06	Georgy	Dredge	M	1994-03-04
3	10,165	1961-04-06	Miyeon	Macedo	M	1989-05-05
4	10,224	1955-05-05	Leon	Trogemann	M	1988-09-01
5	10,247	1966-06-06	Heon	Riefers	F	1993-02-08
6	10,337	1957-10-12	Jeong	Sadowsky	M	1995-06-08
7	10,338	1961-12-02	Heon	Ranai	M	1988-01-09

Example:

```
select * from employee where first_name like '_e____';
```

	id	birth_date	first_name	last_name	gender	hire_date
1	10,052	1963-02-02	Heping	Nitsch	M	1989-09-05
2	10,055	1956-06-06	Georgy	Dredge	M	1994-03-04
3	10,070	1956-08-08	Reuven	Garigliano	M	1986-02-10
4	10,090	1963-06-05	Kendra	Hofting	M	1987-02-03
5	10,225	1958-01-02	Kellie	Chinen	F	1987-07-06
6	10,288	1959-02-06	Selwyn	Perri	M	1996-05-08

Example:

```
select * from employee where first_name like '_a%a';
```

	id	birth_date	first_name	last_name	gender	hire_date
1	10,008	1959-07-02	Saniya	Kalloufi	M	1995-03-09
2	10,069	1960-06-09	Margareta	Bierman	F	1989-05-11
3	10,093	1964-11-06	Sailaja	Desikan	M	1996-05-11
4	10,131	1953-07-02	Magdalena	Eldridge	M	1995-05-11
5	10,144	1960-05-06	Marla	Brendel	M	1986-02-10
6	10,164	1957-07-01	Jagoda	Braunmuhl	M	1985-12-11

Example:

```
select * from salary where amount:text like '6___1';
```

	employee_id	amount	from_date	to_date
1	69,966	66,121	1987-05-11	1988-05-11
2	71,068	63,691	1987-05-10	1988-04-10
3	71,282	60,251	1987-05-02	1988-05-02
4	71,616	61,281	1986-02-03	1987-02-03
5	71,616	62,651	1987-02-03	1988-01-03
6	71,766	62,271	1987-11-08	1988-04-05
7	71,806	64,181	1986-08-06	1987-08-06

LIMIT clause:

- The PostgreSQL LIMIT clause is used to limit the data amount returned by the SELECT statement.

Syntax:

```
SELECT column1, column2, columnN
FROM table_name
LIMIT [no of rows]
```

Example:

```
select * from salary limit 5;
```

	employee_id	amount	from_date	to_date
1	204,388	40,000	1987-11-06	1988-11-06
2	204,399	40,000	1985-11-10	1986-11-10
3	204,399	40,444	1986-11-10	1987-11-10
4	204,399	44,491	1987-11-10	1988-10-10
5	204,401	59,468	1986-01-09	1987-01-09

- The following is the syntax of LIMIT clause when it is used along with OFFSET clause:

Syntax:

```
SELECT column1, column2, columnN
FROM table_name
LIMIT [no of rows] OFFSET [row num]
```

- LIMIT and OFFSET allow you to retrieve just a portion of the rows that are generated by the rest of the query.

Example:

select * from department_employee limit 5 offset 3;

	employee_id	department_id	from_date	to_date
1	10,004	d004	1986-01-12	1999-01-01
2	10,005	d003	1989-12-09	1999-01-01
3	10,006	d005	1990-05-08	1999-01-01
4	10,007	d008	1989-10-02	1999-01-01
5	10,008	d005	1998-11-03	2002-07-07

ORDER BY clause:

- The PostgreSQL ORDER BY clause is used to sort the data in ascending or descending order, based on one or more columns.

Syntax:

SELECT column-list

FROM table_name

[WHERE condition]

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

- You can use more than one column in the ORDER BY clause. Make sure whatever column you are using to sort, that column should be available in column list.

Example:

select * from employee order by birth_date;

	id	birth_date	first_name	last_name	gender	hire_date
1	87,461	1952-01-02	Moni	Decaestecker	M	1986-06-10
2	65,308	1952-01-02	Jouni	Pocchiola	M	1985-10-03
3	237,571	1952-01-02	Ronghao	Schaad	M	1988-10-07
4	91,374	1952-01-02	Eishiro	Kuzuoka	M	1992-12-02
5	207,658	1952-01-02	Kiyokazu	Whitcomb	M	1990-02-07
6	406,121	1952-01-02	Supot	Remmele	M	1991-03-01

Example:

select * from employee order by hire_date desc;

	id	birth_date	first_name	last_name	gender	hire_date
25	225,657	1960-10-08	Xiaoheng	Chenoweth	F	2001-07-01
26	103,208	1954-04-10	Charmane	Cannard	M	2001-07-01
27	203,299	1953-02-07	Girolamo	Binkley	F	2001-07-01
28	94,646	1956-08-05	Mahendra	Kawashimo	F	2001-07-01
29	71,159	1953-07-07	Manton	Ghemri	F	2001-06-12
30	73,925	1959-11-08	Vasilii	Stavenow	M	2001-06-12

Example:

select * from salary order by amount, from_date;

	employee_id	amount	from_date	to_date
1	10,001	20,000	1988-02-06	1989-02-06
2	209,812	20,000	1988-02-07	1989-02-07
3	210,226	20,000	1988-02-07	1989-02-07
4	80,894	20,000	1988-02-07	1989-02-07
5	81,172	20,000	1988-02-07	1989-01-07

GROUP BY clause:

- 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. This is done to eliminate redundancy in the output and/or compute aggregates that apply to these groups.

- The GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

- The basic syntax of GROUP BY clause is given below. The GROUP BY clause must follow the conditions in the WHERE clause and must precede the ORDER BY clause if one is used.

Syntax:

```
SELECT column-list
FROM table_name
WHERE [ conditions ]
GROUP BY column1, column2....columnN
ORDER BY column1, column2....columnN
```

- You can use more than one column in the GROUP BY clause. Make sure whatever column you are using to group, that column should be available in column list.