Postgress Installation

Step-1 :- sudo apt-get install postgresql postgresql-contrib

Step-2 :- service postgresql status

* importatnt Commands *

- \l :- To Show All The Database
- \du :- To Show All The User

* Postgress Users Queries *

- Create User :- create user demo with password ‘anything given’ ;

Example :- create user testuser with password 'test1234';

- Grant User :- Grant all on database database_name to user;

Example :- Grant all on database demo to testuser

Postgress Operations

Step-1 :- To login postgress database enter sudo -u postgres psql

Step-2 :- To create a Database

➔create database database_name;

Step-3 :- To show all the databases

➔\l

Step-4 :- Create User :- create user demo with password ‘anything given’ ;

➔ create user testuser with password 'test1234';

Step-5 :- Grant User :- Grant all on database database_name to user;

➔Grant all on database demo to testuser

Strp-6 :- Add Priviilages

alter user username with SUPERUSER;

Step-7 :- Remove User

drop user user_name;

Postgress Drop Database

Step-1 :- drop database database_name;

Step-2 :- is the error is generate

Error :- `database "demo" is being accessed by other users`

Detail :- `There is 1 other session using the database.`

Step-3 :- first you switch postgres database

Put :- \c postgres

Step4 :- write, query the `pg_stat_activity` view to find what activities are taking place against the `databasename` database

```
SELECT
    *
FROM
    pg_stat_activity
WHERE
    datname = 'demo';
```

The `databasename` database has 1 connection from `localhost` therefore it is safe to terminate this connection and remove the database.

Second, terminate the connection to the `databasename` database by using the following statement:

*Step5 :- in postgres type

```
SELECT
    pg_terminate_backend (pg_stat_activity.pid)
FROM
    pg_stat_activity
WHERE
    pg_stat_activity.datname = 'demo';
```

Then Fire The drop database query

drop database databasename;

database dropped

Postgress Alter Database

1) Rename database

=> alter database target_database RENAME TO new_database;

2) Change owner

Only the `superuser` or owner of the database can change the database's owner. The database owner must also have the `CREATEDB` privilege to rename the database.

```
ALTER DATABASE target_database OWNER TO new_onwer;
```

Exa : ALTER DATABASE TEST OWNER TO TESTUSER

TESTUSER :- it is a name of created or existing user. (\du)

3) Change Password Of owner

=> alter user testuser with password 'abc';

* restart the server:

```
sudo service postgresql restart
```

database Backup/Restore :- https://www.youtube.com/watch?v=KUHD4YGXseI

Postgres Backup Database

pg_dump database_name > new_backup_database_name.sql

pg_dump test > my_database_backup.sql

Postgres Restore Database

psql database_name < your_backup_database_name

psql test < my_database_backup.sql

Postgres Create Table

Create Table :-

create table table_name (field_name1 datatype [constraint],

field_name2 datatype [constraint], field_name3 datatype [constraint], ...);

Query :- CREATE TABLE COMPANY (ID INT PRIMARY KEY NOT NULL, NAME TEXT NOT NULL, AGE INT NOT NULL, ADDRESS CHAR(50), SALARY REAL NOT NULL);

Query2 :- CREATE TABLE DEPARTMENT (ID INT PRIMARY KEY NOT NULL, DEPT_NAME TEXT NOT NULL, EMP_ID INT REFERENCES COMPANY(ID));

Varify Table is succesfully created

Ans :- \d

Use **\d *tablename*** to describe each table as shown below -

Ans :- \d company

DROP TABLE :-

DROP TABLE TABLE_NAME;

DROP TWO OR MORE TABLE

DROP TABLE TABLE_NAME1, TABLE_NAME2;

Postgres Insert Records

INSERT INTO COMPANY VALUES (1, 'Paul', 32, 'California', 20000.00, '2001-07-13');

NOTE :- If Id is Autoincrement then specified all the table fields insted of id field after the name of table name or we can say that before values keyword.

The following example uses the DEFAULT clause for the JOIN_DATE column rather than specifying a value −

INSERT INTO COMPANY VALUES (3, 'Teddy', 23, 'Norway', 20000.00, DEFAULT);

The following example inserts multiple rows using the multirow VALUES syntax −

DEPARTMENT TABLE

INSERT INTO DEPARTMENT VALUES (10, ’SALES’, 1), (20, ’PURCHASE’, 2), (30, ’ACCOUNT’, 3), (10, ’PRODUCT’, 4), (15, ’PARLE’, 5);

Postgres ON DELETE CASCADE

CREATE TABLE DEMO(ID INT PRIMARY KEY NOT NULL, NAME TEXT NOT NULL);

CREATE TABLE TEST(TID INT PRIMARY KEY NOT NULL, TNAME TEXT NOT NULL, DID INT REFERENCES DEMO(ID) ON DELETE CASCADE);

INSERT

INSERT INTO DEMO VALUES (1, 'ABC'), (2, 'XYZ'), (3, 'PQR');

INSERT INTO TEST VALUES (1, 'AAA', 1), (2, 'bbb', 2), (3, 'CCC', 1), (4, 'DDD', 2), (5, 'EEE', 1), (6, 'FFF', 2);

SELECT * FROM DEMO;

SELECT * FROM TEST;

DELETE

DELETE FROM DEMO WHERE ID = 1;

SELECT * FROM DEMO;

SELECT * FROM TEST;

Postgres ON DELETE SET NULL

CREATE TABLE DEMO(ID INT PRIMARY KEY NOT NULL, NAME TEXT NOT NULL);

CREATE TABLE TEST(TID INT PRIMARY KEY NOT NULL, TNAME TEXT NOT NULL, DID INT REFERENCES DEMO(ID) ON DELETE SET NULL);

INSERT

INSERT INTO DEMO VALUES (1, 'ABC'), (2, 'XYZ'), (3, 'PQR');

INSERT INTO TEST VALUES (1, 'AAA', 1), (2, 'bbb', 2), (3, 'CCC', 1), (4, 'DDD', 2), (5, 'EEE', 1), (6, 'FFF', 2);

SELECT * FROM DEMO;

SELECT * FROM TEST;

DELETE

DELETE FROM DEMO WHERE ID = 1;

SELECT * FROM DEMO;

SELECT * FROM TEST;

UPDATE RECORDS

UPDATE TABLE_NAME SET FIELD = VALUE, ... WHERE CONDITION;

UPDATE DEMO SET NAME = ‘XXX’ WHERE ID =2;

DELETE RECORDS

DELETE FROM TABLE_NAME WHERE [CONDITION];

DELETE FROM DEMO WHERE ID = 1;

ALTER TABLE COMMAND

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

You would also use ALTER TABLE command to add and drop various constraints on an existing table.

create table test1(id int,name text);

The basic syntax of **ALTER TABLE** to add a new column in an existing table is as follows −

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

Ans :- ALTER TABLE TEST1 ADD SAL REAL;

The basic syntax of ALTER TABLE to **DROP COLUMN** in an existing table is as follows −

```
ALTER TABLE table_name DROP COLUMN column_name;
```

Ans :- ALTER TABLE TEST1 DROP COLUMN SAL;

The basic syntax of ALTER TABLE to change the **DATA TYPE** of a column in a table is as follows −

```
ALTER TABLE table_name ALTER COLUMN column_name TYPE datatype;
```

Ans :- ALTER TABLE TEST1 ALTER COLUMN SAL TYPE INT;

The basic syntax of ALTER TABLE to add a **NOT NULL** constraint to a column in a table is as follows −

```
ALTER TABLE table_name MODIFY column_name datatype NOT NULL;
```

The basic syntax of ALTER TABLE to **ADD UNIQUE CONSTRAINT** to a table is as follows −

```
ALTER TABLE table_name
ADD CONSTRAINT MyUniqueConstraint UNIQUE(column1, column2...);
```

ALTER TABLE TEST1 ADD CONSTRAINT MyUniqueConstraint Unique(id);

The basic syntax of ALTER TABLE to **ADD CHECK CONSTRAINT** to a table is as follows −

```
ALTER TABLE table_name
ADD CONSTRAINT MyUniqueConstraint CHECK (CONDITION);
```

Ans :- ALTER TABLE TEST1 ADD CONSTRAINT MyCheckConstraint check(sal > 0);

The basic syntax of ALTER TABLE to **ADD PRIMARY KEY** constraint to a table is as follows −

```
ALTER TABLE table_name
ADD CONSTRAINT MyPrimaryKey PRIMARY KEY (column1, column2...);
```

ALTER TABLE TEST1 ADD CONSTRAINT MyPrimaryKey primary key(id);

The basic syntax of ALTER TABLE to **DROP CONSTRAINT** from a table is as follows −

```
ALTER TABLE table_name
DROP CONSTRAINT MyUniqueConstraint;
```

Ans :- alter table test1 drop constraint MyUniqueConstraint;

If you are using MySQL, the code is as follows −

```
ALTER TABLE table_name
DROP INDEX MyUniqueConstraint;
```

The basic syntax of ALTER TABLE to **DROP PRIMARY KEY** constraint from a table is as follows −

```
ALTER TABLE table_name
DROP CONSTRAINT MyPrimaryKey;
```

alter table test1 drop constraint MyPrimaryKey;

Postgresql JOIN

Join Types in PostgreSQL are −

- The CROSS JOIN
- The INNER JOIN
- The LEFT OUTER JOIN
- The RIGHT OUTER JOIN
- The FULL OUTER JOIN

PostgreSQL support inner join, left join, right join, full outer join, cross join, natural join, and a special kind of join called self-join.

INSERT INTO COMPANY VALUES(1, 'Paul', 32, 'California', 20000.00, '2001-07-13'), (3, 'Teddy', 23, 'Norway', 20000.00, null),

(4, 'Mark', 25, 'Rich-Mond', 65000.00, '2007-12-13'), (5, 'David', 27, 'Texas', 85000.00, '2007-07-13'),

(2, 'Allen', 25, 'Texas', null, '2007-07-13'),

(8, 'Paul', 24, 'Houston', 20000.00, '2005-07-13'), (9, 'James', 44, 'Norway', 5000.00, '2005-07-13'),

(10, 'James', 45, 'Texas', 5000.00, '2005-07-13');

insert into department values(1, 'IT Billing', 1), (2, 'Engineering', 2), (3, 'Finance', 5);

1) THE CROSS JOIN :-

A CROSS JOIN matches every row of the first table with every row of the second table. If the input tables have x and y columns, respectively, the resulting table will have x+y columns. Because CROSS JOINs have the potential to generate extremely large tables, care must be taken to use them only when appropriate.

The following is the syntax of CROSS JOIN −

```
SELECT ... FROM table1 CROSS JOIN table2 ...
```

Example :-

SELECT C.EMP_ID, C.NAME, D.DEPT_NAME FROM COMPANY C CROSS JOIN DEPARTMENT D;

2) THE INNER JOIN :-

the inner join returns a result set that contains row in the left table that match with the row in the right table.

An INNER JOIN is the most common type of join and is the default type of join. You can use INNER keyword optionally.

The following is the syntax of INNER JOIN −

```
SELECT table1.column1, table2.column2...

FROM table1
```

```
INNER JOIN table2

ON table1.common_filed = table2.common_field;
```

EXAMPLE :-

SELECT E.EMP_ID, E.NAME, D.DEPT_NAME, D.DEPT_ID FROM COMPANY E, DEPARTMENT D WHERE E.EMP_ID = D.EMP_ID;

OR

SELECT D.EMP_ID, E.NAME, D.DEPT_NAME, D.DEPT_ID FROM COMPANY E JOIN DEPARTMENT D ON E.EMP_ID = D.EMP_ID;

OR

SELECT D.EMP_ID, E.NAME, D.DEPT_NAME, D.DEPT_ID FROM COMPANY E INNER JOIN DEPARTMENT D ON E.EMP_ID = D.EMP_ID;

3) THE LEFT OUTER JOIN :-

In case of LEFT OUTER JOIN, an inner join is performed first. 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. Thus, the joined table always has at least one row for each row in T1.

The left join returns a complete set of rows from the left table with the matching rows if available from the right table. If there is no match, the right side will have null values.

The following is the syntax of LEFT OUTER JOIN −

```
SELECT ... FROM table1 LEFT OUTER JOIN table2 ON conditional_expression ...
```

EXAMPLE :-

SELECT E.EMP_ID, E.NAME, E.SALARY, D.DEPT_NAME, D.DEPT_ID FROM COMPANY E LEFT OUTER JOIN DEPARTMENT D ON E.EMP_ID = D.EMP_ID;

4) THE RIGHT OUTER JOIN :-

First, an inner join is performed. Then, for each row in table T2 that does not satisfy the join condition with any row in table T1, a joined row is added with null values in columns of T1. This is the converse of a left join; the result table will always have a row for each row in T2.

The right join or right outer join is a reversed version of the left join. It produces a result set that contains all rows from the right table with

matching rows from the left table. If there is no match, the left side will contain null values.

The following is the syntax of RIGHT OUTER JOIN −

```
SELECT ... FROM table1 RIGHT OUTER JOIN table2 ON conditional_expression ...
```

SELECT E.EMP_ID, E.NAME, E.SALARY, D.DEPT_NAME, D.DEPT_ID FROM COMPANY E RIGHT OUTER JOIN DEPARTMENT D ON E.EMP_ID = D.EMP_ID;

5) THE 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.

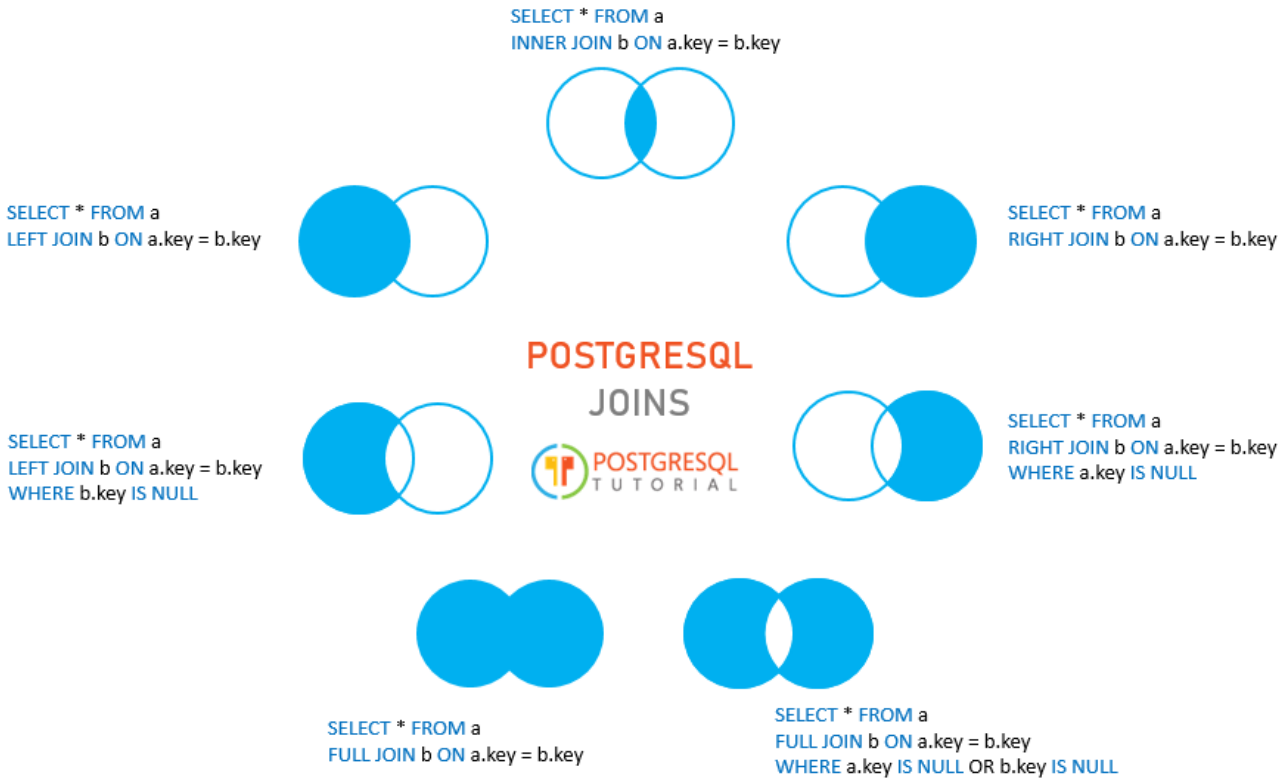
The full outer join or full join produces a result set that contains all rows from both the left and right tables, with the matching rows from both sides where available. If there is no match, the missing side contains null values.

The following is the syntax of FULL OUTER JOIN −

```
SELECT ... FROM table1 FULL OUTER JOIN table2 ON conditional_expression ...
```

EXAMPLE :-

SELECT COMPANY.EMP_ID , COMPANY.NAME, COMPANY.SALARY, DEPARTMENT.DEPT_ID, DEPARTMENT.DEPT_NAME FROM COMPANY FULL OUTER JOIN DEPARTMENT ON COMPANY.EMP_ID = DEPARTMENT.EMP_ID;



SELF JOIN :-

A self-join is a query in which a table is joined to itself. Self-joins are useful for comparing values in a column of rows within the same table.

To form a self-join, you specify the same table twice with different aliases, set up the comparison, and eliminate cases where a value would be equal to itself.

EXAMPLE :-

SELECT E.FIRST_NAME ||' ' || E.LAST_NAME AS EMPLOYEE,F.FIRST_NAME || ' ' || F.LAST_NAME AS MANAGER FROM EMPLOYEE E JOIN EMPLOYEE F ON E.MANAGER_ID=F.EMPLOYEE_ID;

self join :- https://www.youtube.com/watch?v=6DQpvfdj6EE

Date-Time In Postgres

The following table lists the behaviors of the basic arithmetic operators −

Operator	Example	Result
+	date '2001-09-28' + integer '7'	date '2001-10-05'
+	date '2001-09-28' + interval '1 hour'	timestamp '2001-09-28 01:00:00'

+	date '2001-09-28' + time '03:00'	timestamp '2001-09-28 03:00:00'
+	interval '1 day' + interval '1 hour'	interval '1 day 01:00:00'
+	timestamp '2001-09-28 01:00' + interval '23 hours'	timestamp '2001-09-29 00:00:00'
+	time '01:00' + interval '3 hours'	time '04:00:00'
-	- interval '23 hours'	interval '-23:00:00'
-	date '2001-10-01' - date '2001-09-28'	integer '3' (days)
-	date '2001-10-01' - integer '7'	date '2001-09-24'
-	date '2001-09-28' - interval '1 hour'	timestamp '2001-09-27 23:00:00'
-	time '05:00' - time '03:00'	interval '02:00:00'
-	time '05:00' - interval '2 hours'	time '03:00:00'
-	timestamp '2001-09-28 23:00' - interval '23 hours'	timestamp '2001-09-28 00:00:00'
-	interval '1 day' - interval '1 hour'	interval '1 day -01:00:00'
-	timestamp '2001-09-29 03:00' - timestamp '2001-09-27 12:00'	interval '1 day 15:00:00'
*	900 * interval '1 second'	interval '00:15:00'
*	21 * interval '1 day'	interval '21 days'
*	double precision '3.5' * interval '1 hour'	interval '03:30:00'
/	interval '1 hour' / double precision '1.5'	interval '00:40:00'

The following is the list of all important Date and Time related functions available.

S. No.	Function & Description
1	**AGE()** Subtract arguments
2	**CURRENT DATE/TIME()**

	Current date and time
3	**DATE_PART()** Get subfield (equivalent to extract)
4	**EXTRACT()** Get subfield
5	**ISFINITE()** Test for finite date, time and interval (not +/- infinity)
6	**JUSTIFY** Adjust interval

1) AGE :-

```
SELECT AGE(timestamp '2001-04-10', timestamp '1957-06-13');
select age(timestamp '1957-06-13');
```

2) CURRENT DATE/TIME()

Select now(); -> it gives date and time both

select CURRENT_DATE; -> it gives the current date

select CURRENT_TIME; -> it gives the current time

select CURRENT_TIMESTAMP; -> it gives values with timezones.

3) DATE_PART() :-

it is used to get subfield (equivalent to extract)

DATE_PART('field', source)

These functions get the subfields. The *field* parameter needs to be a string value, not a name.

The valid field names are: *century, day, decade, dow, doy, epoch, hour, isodow, isoyear, microseconds, millennium, milliseconds, minute, month, quarter, second, timezone, timezone_hour, timezone_minute, week, year.*

Example :-SELECT date_part('hour', INTERVAL '4 hours 3 minutes');

SELECT date_part('day', TIMESTAMP '2001-02-16 20:38:40');

Datetime Conversions: PostgreSQL

TO_CHAR({*timestamp*|*interval*}, *format*)
TO_DATE(*string*, *format*)
TO_TIMESTAMP(*string*, *format*)

1) TO_CHAR() :-

select emp_id,name,age,address,to_char(join_date,'dd-Mon-YYYY') from company;