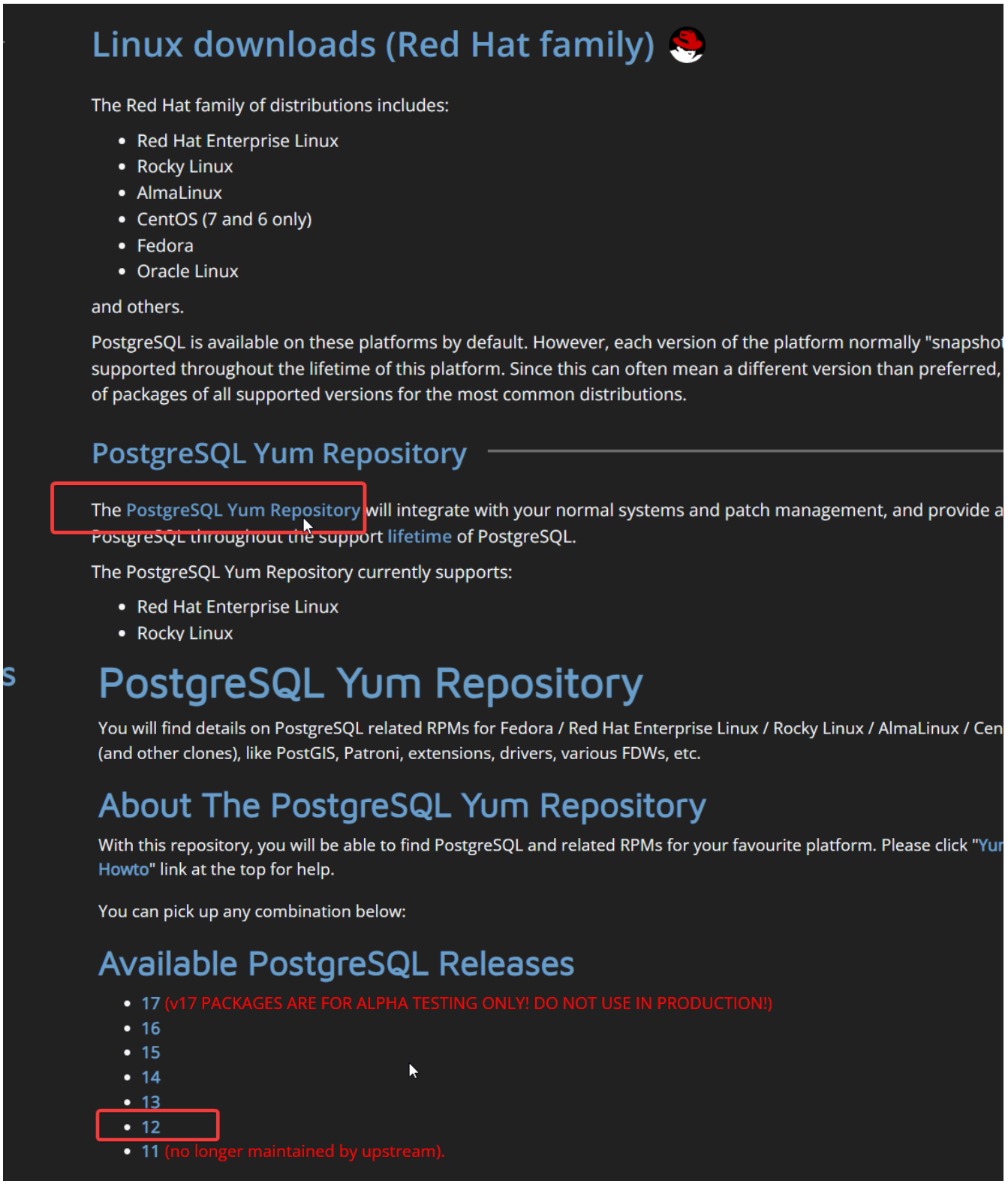
2. installing PostgreSQL on linux

below steps are done on on rocky Linux
RHEl package management

downloading package

In the realm of Red Hat, we employ RPMs (Red Hat Package Manager) as a fundamental tool. RPM files serve as the architects of software repositories, facilitating the seamless installation of software packages. The process unfolds as follows:

1. Selecting the Operating System (OS) Version: This step is akin to choosing the foundation upon which our digital infrastructure will be built.
2. Utilizing the Yum Repository Method: We opt for the Yum repository method, a robust approach akin to gaining access to a comprehensive software library.
3. PostgreSQL Version Selection: A pivotal decision arises as we must specify the PostgreSQL version. For our purposes, let's opt for PostgreSQL version 12.



use the below command `wget` to download file directory from website

`wget [link to download ]`

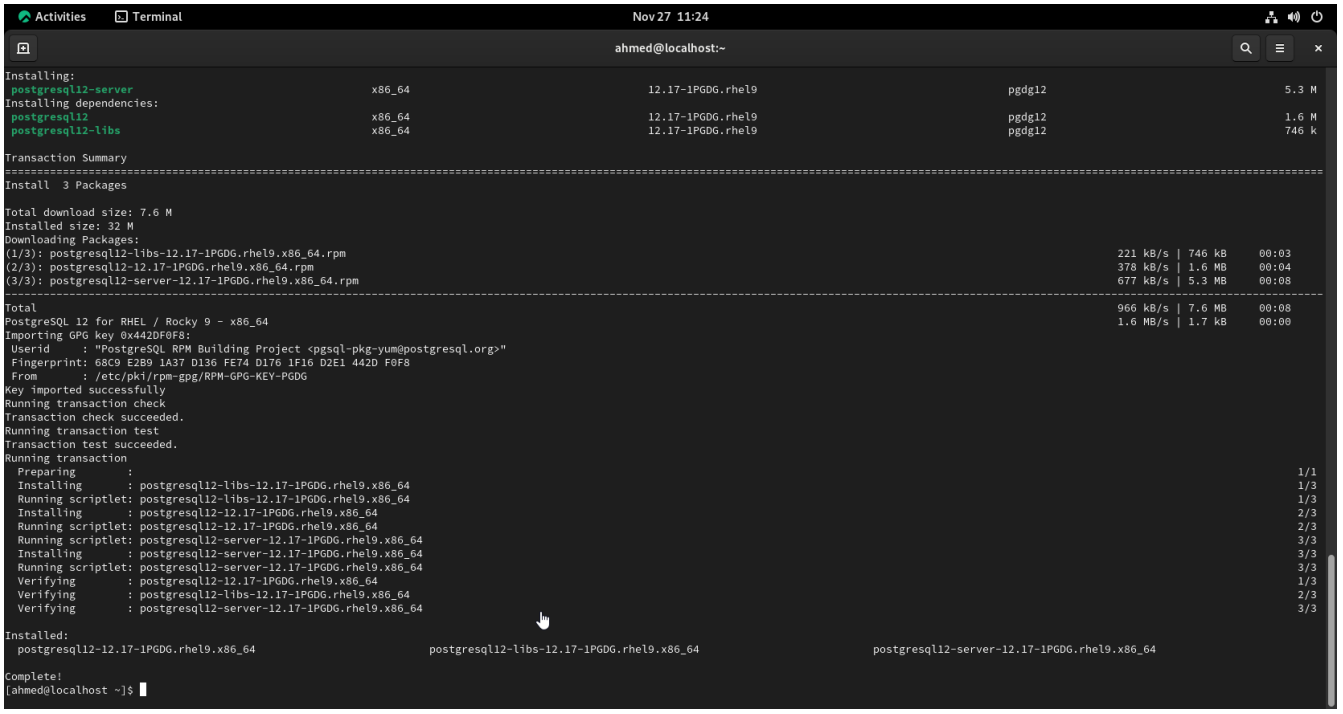
`wget [https://yum.postgresql.org/packages/#pg12]`

or use the below script which you can get from website its self

```
sudo dnf install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-9-x86_64/pgdg-redhat-repo-latest.noarch.rpm sudo dnf -qy module disable postgresql sudo dnf install -y postgresql12-server sudo /usr/pgsql-12/bin/postgresql-12-setup initdb sudo systemctl enable postgresql-12 sudo systemctl start postgresql-12
```

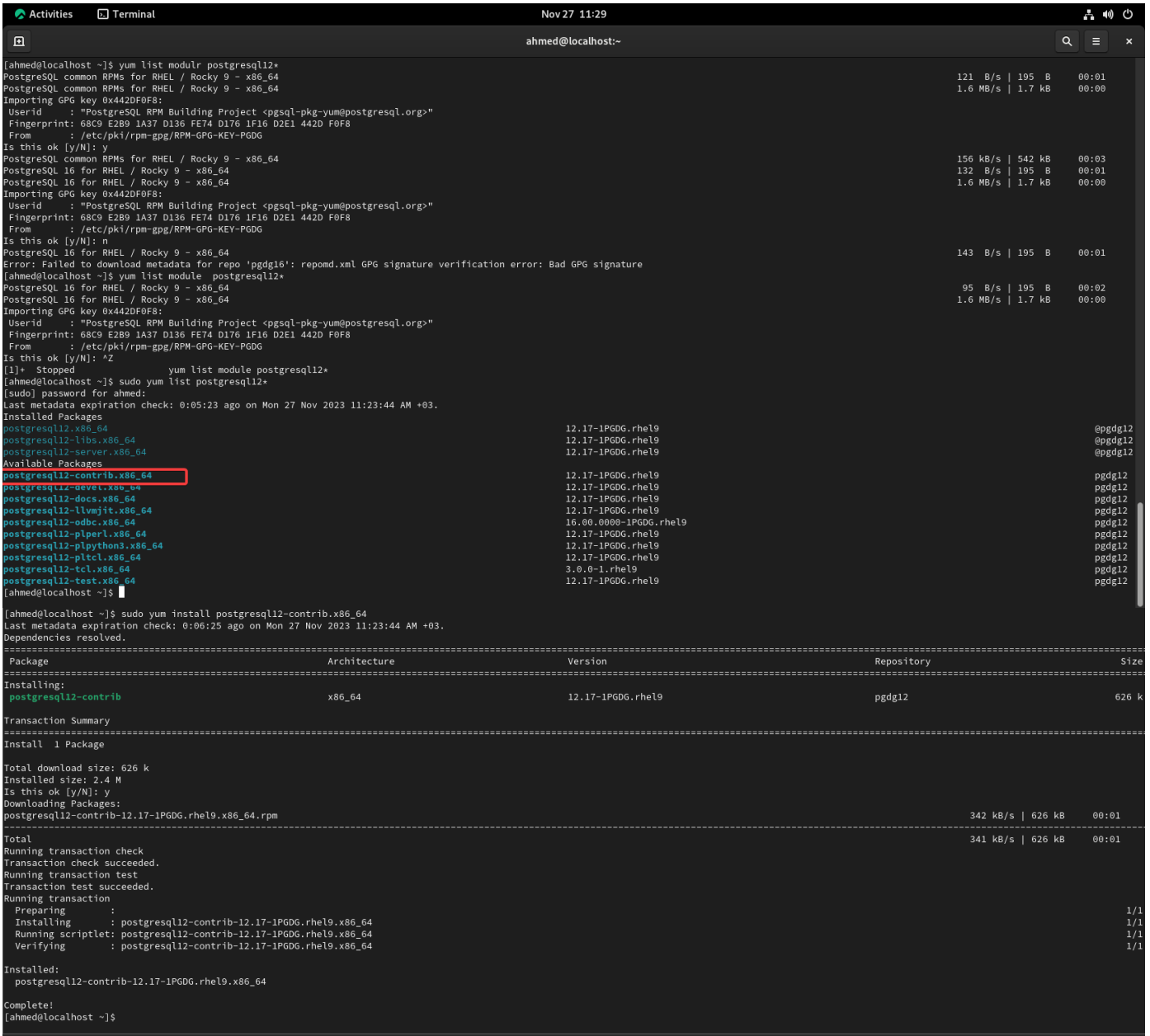
- on the script you will download rpm repo package

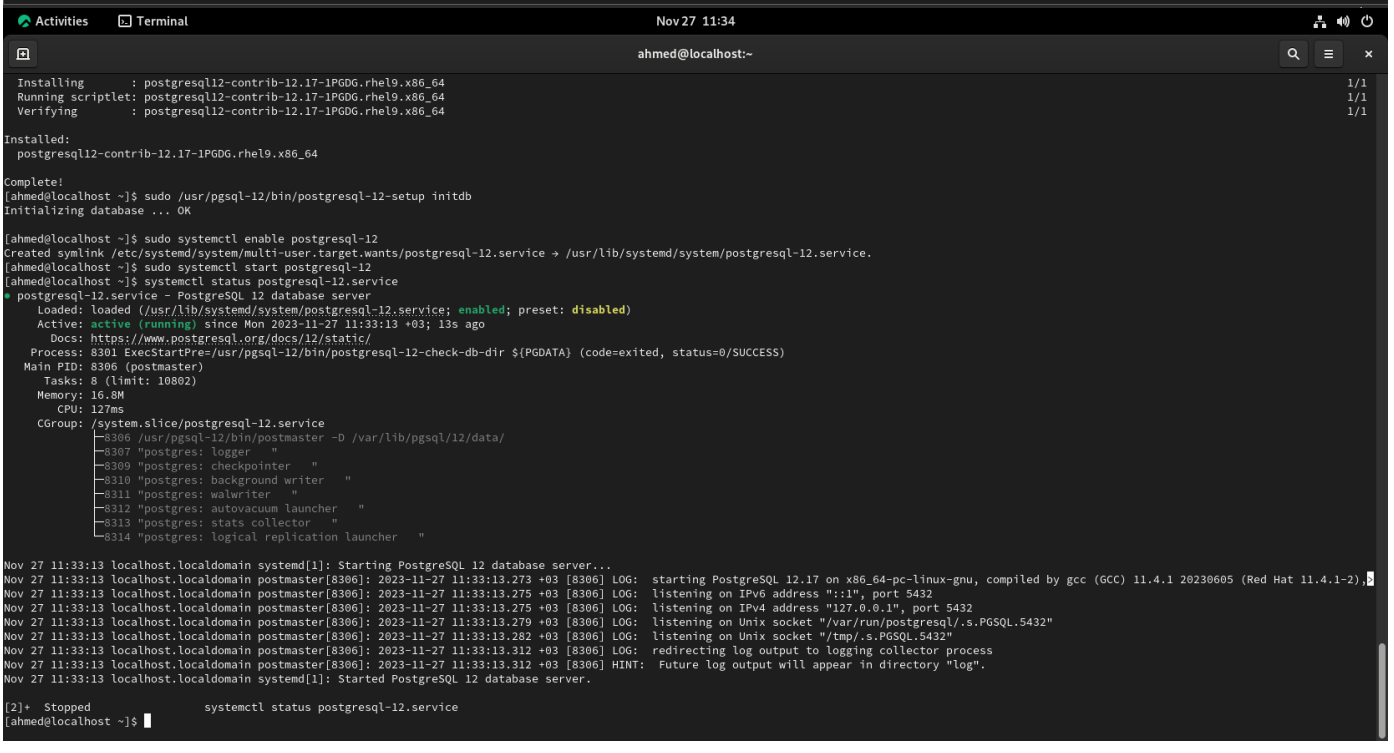
- disable the PostgreSQL module available on redhat (only redhat)
- `sudo dnf -qy module disable postgresql` This command ensures that the PostgreSQL module will not be active or installed on the system. It can be useful when you want to prevent the installation or use of a specific PostgreSQL module or version
- `/usr/pgsql-12/bin/postgresql-12-setup`: This part of the command specifies the path to the PostgreSQL version 12 setup utility. It's a script or program responsible for configuring and managing various aspects of PostgreSQL, including initializing the database cluster.
- `initdb`: The `"initdb"` command is short for "initialize database." When executed, it creates a new PostgreSQL database cluster, which is the core directory structure and configuration files required to run a PostgreSQL instance. It sets up the essential components and directory structure for a fresh database installation.
  In summary, the provided command, "sudo /usr/pgsql-12/bin/postgresql
- setup initdb," is used to initialize a PostgreSQL version 12 database cluster on the system with elevated privileges, creating the necessary infrastructure for PostgreSQL to operate effectively.



additional package we need to install postgree-contrib , this will include all additional features

to locate this the syntax for package use `yum list postgresql12*`



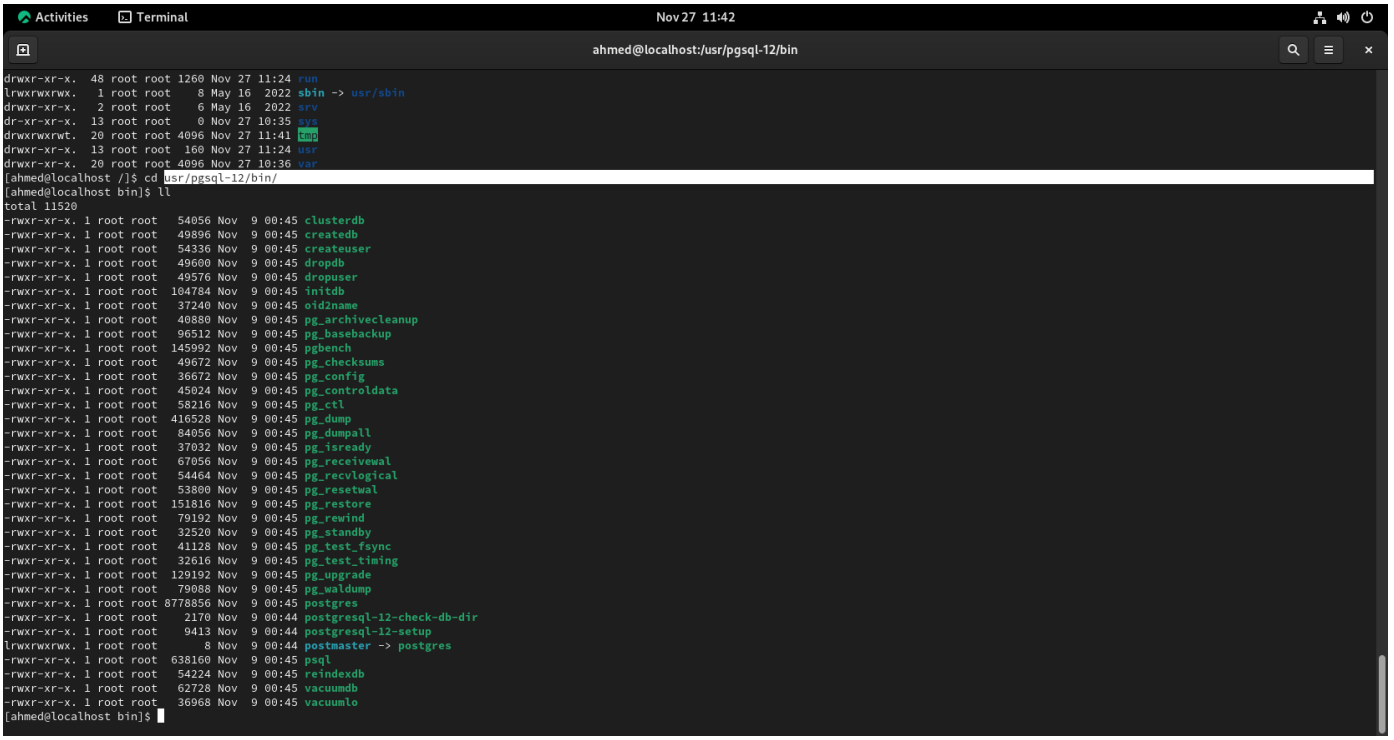


configuring PostgreSQL post installation

During the installation of PostgreSQL on a Linux system, you may have noticed that the installation process lacks certain user interactions that are typically associated with software installations. Unlike some other software, PostgreSQL installation on Linux doesn't ask you for important details such as the data directory, installation directory, port number, or even the PostgreSQL password.

installation and data directory in Linux

the installation directory will be in `usr/pgsql-12/bin/`



the data directory will be locating in `/var/lib/pgsql/`

create superuser for PostgreSQL

to check the superuser for PostgreSQL is configured

type `su - postgres`

if it dint ask you for password means the password was not setup for PostgreSQL

```
[root@localhost ~]# su - postgres
[postgres@localhost ~]$
```

you cannot login to PostgreSQL using normal only root can access

then once you login just Chang the password using `passwd` command using the root user

make sure the services are running

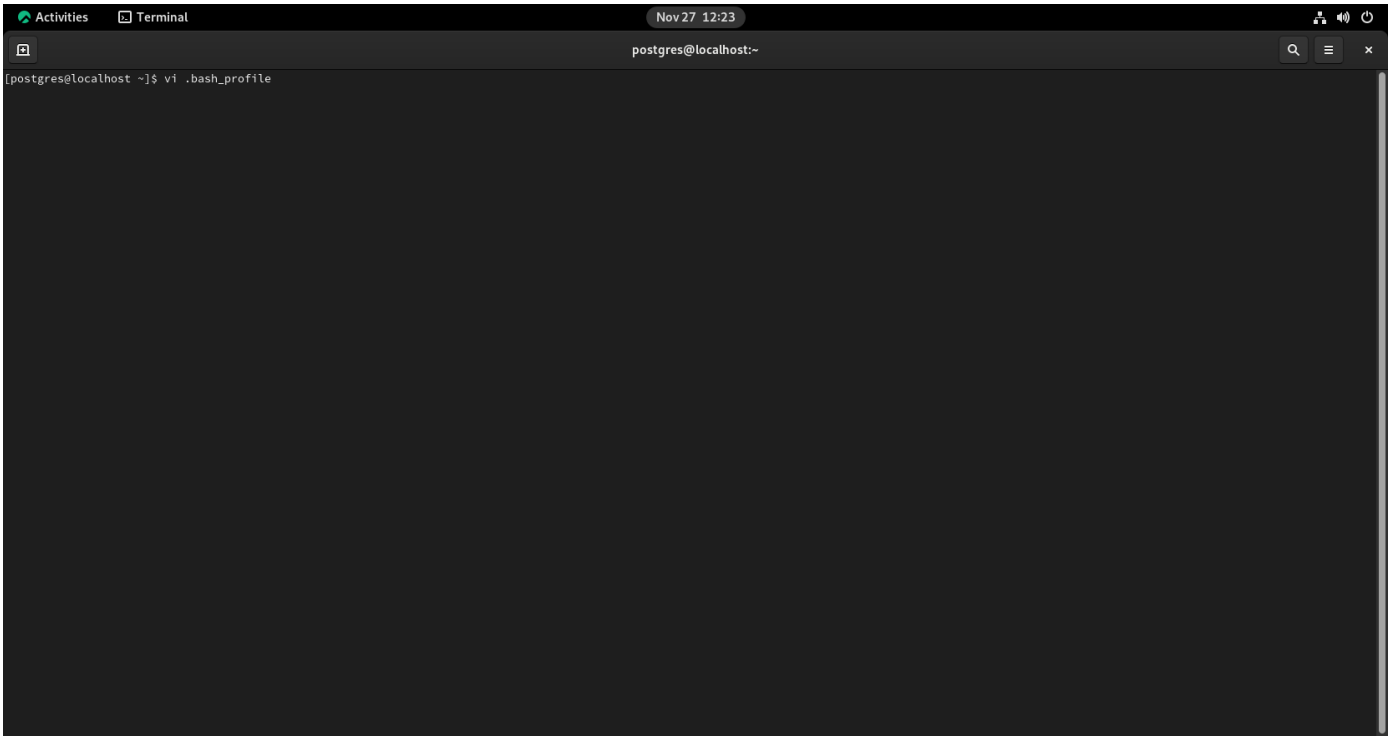
make sure the below command is installed

`sudo systemctl enable postgresql-12 sudo systemctl start postgresql-12`

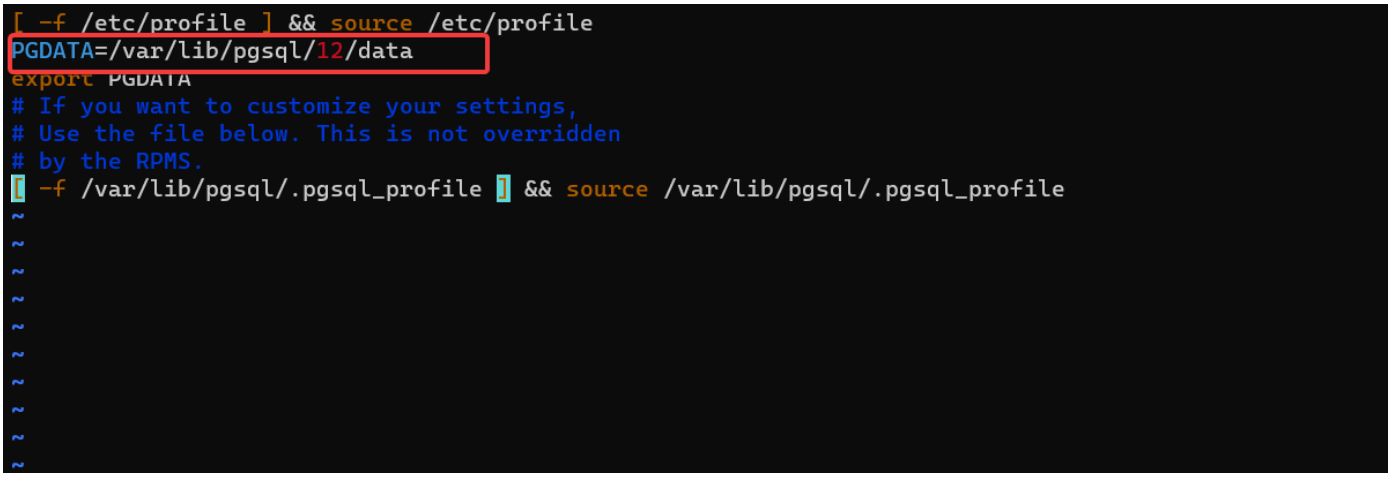
setup environment variables

first login using postgres user

the use VIM to edit the

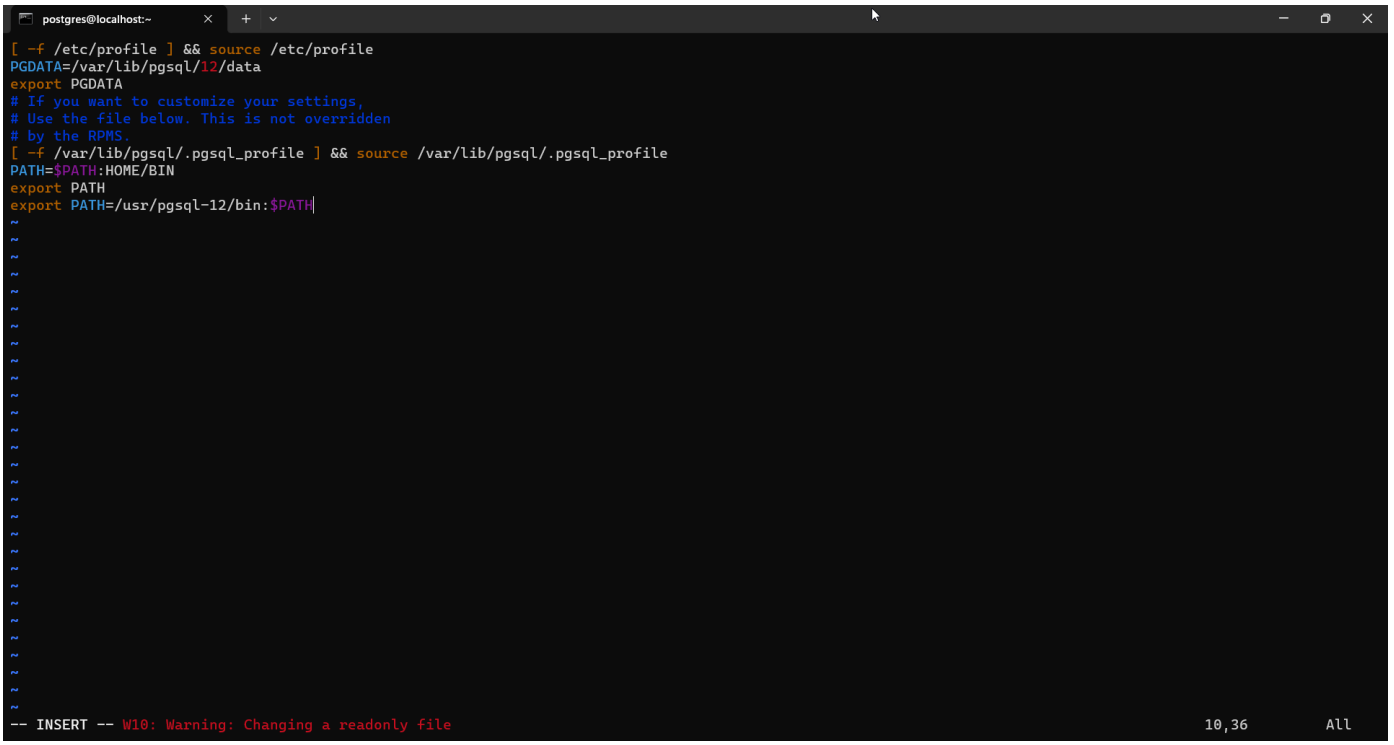


the installer has already specify the pgdata path for the user



you just need to specify the path for binary to be used by user

`PATH=$PATH:HOME/BIN export PATH export PATH=/usr/pgsql-12/bin:$PATH`



when you want to use deterrent user then you will have to edit environment path for that user to .

3. database cluster

PostgreSQL+Cluster.docx

what is database cluster

database cluster is collection of database managed by single instance
don't get confused by cluster that tied up by high availability

initdb

initdb : is utility used to create database cluster
initdb create setup directory called (data directory) where we store our data .

initdb need to be installed , and once installed you have to initialise storage area

data directory path

typically the location of data directory in Linux will be

`/var/lib/pgsql/data/`

initdb syntax

you have to execute initdb using the postgres user

syntax:

`initdb -D /usr/var/lib/pgsql/main`

`pg_ctl -D /usr/var/lib/pgsql/main`

`-D` this option refer to data directory

`-W` we can use this option to force the super user to provide password before initialise DB

before you start make sure add biniray of postgreasql commmand

`export PATH=/usr/lib/postgresql/13/bin/:$PATH`

to test this we will create directory in root folder called `/postgresql/data/`

and we will assigned the permission

`chmod 770 /postgresql/data/`

```
root@PostgreSQLSTG /# mkdir -p /postgresql/data/
root@PostgreSQLSTG /# chmod 700 /postgresql/data/
root@PostgreSQLSTG /#
```

the we will run the command

```
root@PostgreSQLSTG /# initdb -D /postgresql/data/
-bash: initdb: command not found
root@PostgreSQLSTG /#
```

if get this error means the `initdb` is not added in environment of OS

usually the bainiray are located at `ls /usr/lib/postgresql/13/bin/`

```
root@PostgreSQLSTG /# ls /usr/lib/postgresql/13/bin/
clusterdb    dropuser           pg_basebackup   pg_ctl      pg_receivewal   pg_rewind       pg_upgrade       postgres    vacuumdb
createdb     initdb             pg_checksums    pg_dump     pg_recvlogical  pg_standby      pg_verifybackup  postmaster  vacuumlo
createuser   oid2name           pg_config       pg_dumpall  pg_resetwal     pg_test_fsync   pg_waldump       psql
dropdb       pg_archivecleanup  pg_controldata  pg_isready  pg_restore      pg_test_timing  pgbench          reindexdb
root@PostgreSQLSTG /#
```

to run the command you will have to put the full path of `initdb` binary

with follow up with the option and directory

make sure that command is not executed using the root user

```
root@PostgreSQLSTG /# /usr/lib/postgresql/13/bin/initdb -D /postgresql/data/
initdb: error: cannot be run as root
Please log in (using, e.g., "su") as the (unprivileged) user that will
own the server process.
root@PostgreSQLSTG /# su - postgres
]postgres@PostgreSQLSTG:~$ /usr/lib/postgresql/13/bin/initdb  -D /postgresql/data/
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.

The database cluster will be initialized with locale "C".
The default database encoding has accordingly been set to "SQL_ASCII".
The default text search configuration will be set to "english".

Data page checksums are disabled.

initdb: error: could not access directory "/postgresql/data": Permission denied
postgres@PostgreSQLSTG:~$
```

if you ran to the below issue you also have to change the owner of the folder using `chown`

`chown postgres:postgres /postgresql/data/`

```
root@PostgreSQLSTG ~# chown postgres:postgres /postgresql/data/
root@PostgreSQLSTG ~# ls
root@PostgreSQLSTG ~# cd /
root@PostgreSQLSTG /# ls -l
```

then try to run the command again and it should execute correctly

`/usr/lib/postgresql/13/bin/initdb -D /postgresql/data/`

```
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.

The database cluster will be initialized with locale "C".
The default database encoding has accordingly been set to "SQL_ASCII".
The default text search configuration will be set to "english".

Data page checksums are disabled.

fixing permissions on existing directory /postgresql/data ... ok
creating subdirectories ... ok
selecting dynamic shared memory implementation ... posix
selecting default max_connections ... 100
selecting default shared_buffers ... 128MB
selecting default time zone ... Etc/UTC
creating configuration files ... ok
running bootstrap script ... ok
performing post-bootstrap initialization ... ok
syncing data to disk ... ok

initdb: warning: enabling "trust" authentication for local connections
You can change this by editing pg_hba.conf or using the option -A, or
--auth-local and --auth-host, the next time you run initdb.

Success. You can now start the database server using:

    /usr/lib/postgresql/13/bin/pg_ctl -D /postgresql/data/ -l logfile start

postgres@PostgreSQLSTG:~$
```

now go to the file you created and you will see there files created on new directory

```
postgres@PostgreSQLSTG:~$ cd /postgresql/data/
postgres@PostgreSQLSTG:/postgresql/data$ ls
PG_VERSION  pg_commit_ts  pg_ident.conf  pg_notify    pg_snapshots  pg_subtrans  pg_wal               postgresql.conf
base        pg_dynshmem   pg_logical     pg_replslot  pg_stat       pg_tblspc    pg_xact
global      pg_hba.conf   pg_multixact   pg_serial    pg_stat_tmp   pg_twophase  postgresql.auto.conf
postgres@PostgreSQLSTG:/postgresql/data$
```

starting the cluster

once we created the cluster it wont start automatically , we need to start the cluster using the command same issue as `INITDB` command you have to give the full path of the command or edit the environment to include the pg_ctl path

`pg_ctl start`

`/usr/lib/postgresql/13/bin/initdb -D /postgresql/data/`

if will ran to error as showing below , because the cluster is using the same port 5432 as default cluster

```
postgres@PostgreSQLSTG:/postgresql/data$ /usr/lib/postgresql/13/bin/pg_ctl start -D /postgresql/data/
waiting for server to start....2023-12-09 15:57:01.599 UTC [6246] LOG:  starting PostgreSQL 13.8 (Debian 13.8-0+deb11u1) on x86_6
-pc-linux-gnu, compiled by gcc (Debian 10.2.1-6) 10.2.1 20210110, 64-bit
2023-12-09 15:57:01.600 UTC [6246] LOG:  could not bind IPv4 address "127.0.0.1": Address already in use
2023-12-09 15:57:01.600 UTC [6246] HINT:  Is another postmaster already running on port 5432? If not, wait a few seconds and retr
.
2023-12-09 15:57:01.600 UTC [6246] WARNING:  could not create listen socket for "localhost"
2023-12-09 15:57:01.600 UTC [6246] FATAL:  could not create any TCP/IP sockets
2023-12-09 15:57:01.602 UTC [6246] LOG:  database system is shut down
 stopped waiting
pg_ctl: could not start server
```

you need to change it to random one

edit the postgresql.conf using nano and edit the port and enable listening to local host

```
# - Connection Settings -

listen_addresses = 'localhost'          # what IP address(es) to listen on;
                                        # comma-separated list of addresses;
                                        # defaults to 'localhost'; use '*' for all
                                        # (change requires restart)
port = 5444                             # (change requires restart)
max_connections = 100                   # (change requires restart)
#superuser_reserved_connections = 3     # (change requires restart)
#unix_socket_directories = '/var/run/postgresql'        # comma-separated list of directories
                                        # (change requires restart)
#unix_socket_group = ''                 # (change requires restart)
#unix_socket_permissions = 0777         # begin with 0 to use octal notation
                                        # (change requires restart)
#bonjour = off                          # advertise server via Bonjour
                                        # (change requires restart)
#bonjour_name = ''                      # defaults to the computer name
                                        # (change requires restart)

# - TCP settings -
# see "man tcp" for details
```

now if you ran the command again it will execute successfully

```
root@PostgreSQLSTG /postgresql/data# su - postgres
postgres@PostgreSQLSTG:~$ /usr/lib/postgresql/13/bin/initdb  -D /postgresql/data/
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.

The database cluster will be initialized with locale "C".
The default database encoding has accordingly been set to "SQL_ASCII".
The default text search configuration will be set to "english".

Data page checksums are disabled.

initdb: error: directory "/postgresql/data" exists but is not empty
If you want to create a new database system, either remove or empty
the directory "/postgresql/data" or run initdb
with an argument other than "/postgresql/data".
postgres@PostgreSQLSTG:~$ /usr/lib/postgresql/13/bin/pg_c
pg_checksums    pg_config       pg_controldata  pg_ctl
postgres@PostgreSQLSTG:~$ /usr/lib/postgresql/13/bin/pg_ctl start -D /postgresql/data/
waiting for server to start....2023-12-09 16:01:31.326 UTC [6354] LOG:  starting PostgreSQL 13.8 (Debian 13.8-0+deb11u1) on x86_64
-pc-linux-gnu, compiled by gcc (Debian 10.2.1-6) 10.2.1 20210110, 64-bit
2023-12-09 16:01:31.327 UTC [6354] LOG:  listening on IPv4 address "127.0.0.1", port 5444
2023-12-09 16:01:31.328 UTC [6354] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5444"
2023-12-09 16:01:31.330 UTC [6355] LOG:  database system was shut down at 2023-12-09 15:45:16 UTC
2023-12-09 16:01:31.334 UTC [6354] LOG:  database system is ready to accept connections
 done
server started
postgres@PostgreSQLSTG:~$
```

how to connect to the cluster

to connect to the new cluster we have created use the below syntax , and make sure to provide the correct port for the cluster

`psql -U postgres [or any user you want] -p 5444[port number]`

```
postgres@PostgreSQLSTG:~$ psql -U postgresql -p 5444
2023-12-09 16:06:39.011 UTC [6409] FATAL:  role "postgresql" does not exist
psql: error: FATAL:  role "postgresql" does not exist
postgres@PostgreSQLSTG:~$ psql -U postgres -p 5444
psql (13.8 (Debian 13.8-0+deb11u1))
Type "help" for help.

postgres=# ls
postgres-# \l
                              List of databases
   Name    |  Owner   | Encoding  | Collate | Ctype |   Access privileges
-----------+----------+-----------+---------+-------+-----------------------
 postgres  | postgres | SQL_ASCII | C       | C     |
 template0 | postgres | SQL_ASCII | C       | C     | =c/postgres          +
           |          |           |         |       | postgres=CTc/postgres
 template1 | postgres | SQL_ASCII | C       | C     | =c/postgres          +
           |          |           |         |       | postgres=CTc/postgres
(3 rows)

postgres-#
```

\q

how to check the status of cluster

you can check if the cluster is runinng by using the below command

`/usr/lib/postgresql/13/bin/pg_ctl status -D /postgresql/data/`

```
postgres@PostgreSQLSTG:~$ /usr/lib/postgresql/13/bin/pg_ctl status  -D /postgresql/data/
pg_ctl: server is running (PID: 6354)
/usr/lib/postgresql/13/bin/postgres "-D" "/postgresql/data"
postgres@PostgreSQLSTG:~$
```

shutdown option

below command is related to shutdown the cluster

1- **smart** : the server disallow new connection , but the existent session work normally , its shutdown only after all session terminated.

2- **fast (default)** : the server disallow new connection and abort their current transaction and exit gracefully.

3- **immediate** : quits/abort without proper shutdown which lead to recovery on next startup

syntax

for the propose i will use smart option in shutdown the new cluster we have created , for the rest of shutdown option is basically the same .

`pg_ctl stope -m smart[also you can uses immediate or fast] -D[data directory]`

```
postgres@PostgreSQLSTG:~$ pg_ctl stop -m smart -D /postgresql/data/
2023-12-09 16:37:17.071 UTC [6354] LOG:  received smart shutdown request
waiting for server to shut down....2023-12-09 16:37:17.077 UTC [6354] LOG:  background worker "logical replication launcher" (PID
6361) exited with exit code 1
2023-12-09 16:37:17.077 UTC [6356] LOG:  shutting down
2023-12-09 16:37:17.086 UTC [6354] LOG:  database system is shut down
 done
server stopped
postgres@PostgreSQLSTG:~$ pg_ctl status -D /postgresql/data/
pg_ctl: no server running
postgres@PostgreSQLSTG:~$
```

reload vs restart

reload is used when we did some changes on configuration files to , reload only load new config with restarting the services .
Some changes in config wont reflect unless we restart the services
restart : gracefully shutdown all activity , close all open files and start again with new configuration .

reload & restart syntax

reload : `system reload postgresql-11`
restart : `systectl restart postgresql`

pg_controldata

its used to provide information about cluster
such as version number , last checkpoint

syntax

`pg_contrldata /var/lib/pgsql/main`

```
  -?, --help            show this help, then exit

If no data directory (DATADIR) is specified, the environment variable PGDATA
is used.

Report bugs to <pgsql-bugs@lists.postgresql.org>.
PostgreSQL home page: <https://www.postgresql.org/>
postgres@PostgreSQLSTG:~$ pg_controldata /postgresql/data/
pg_control version number:            1300
Catalog version number:               202007201
Database system identifier:           7310621525282367391
Database cluster state:               in production
pg_control last modified:             Sat Dec  9 16:44:05 2023
Latest checkpoint location:           0/15CFAF0
Latest checkpoint's REDO location:    0/15CFAB8
Latest checkpoint's REDO WAL file:    000000010000000000000001
Latest checkpoint's TimeLineID:       1
Latest checkpoint's PrevTimeLineID:   1
Latest checkpoint's full_page_writes: on
Latest checkpoint's NextXID:          0:486
Latest checkpoint's NextOID:          16385
Latest checkpoint's NextMultiXactId:  1
Latest checkpoint's NextMultiOffset:  0
Latest checkpoint's oldestXID:        478
Latest checkpoint's oldestXID's DB:   1
Latest checkpoint's oldestActiveXID:  486
Latest checkpoint's oldestMultiXid:   1
Latest checkpoint's oldestMulti's DB: 1
Latest checkpoint's oldestCommitTsXid:0
```

see data directory

to see the data directory connect psql client and use the below command

`show data_directory;`

4- PostgreSQL Directory layout

installation directory layout

PostgreSQL is typicality installed to `/usr/lib/postgresql/`
the same directory structure is the same both on windows and Linux.,
now we will discssed the content of instalation

bin folder

here you will find all PostgreSQL utility files such as `initdb` a& `bg_ctl`

```
postgres@PostgreSQLSTG:/usr/lib/postgresql/13$ ls
bin  lib
postgres@PostgreSQLSTG:/usr/lib/postgresql/13$
```

data

the folder is located in default cluster and might change depend on if there mulitable cluster created , but by default its located in /var/lib/postgresql/13/main

here you will find all data related to database and the configuration file for the cluster

log folder

the log of the PostgreSQL is very important it can support you to troubleshoot the status of PostgreSQL

the log are typically stored in below path
`/var/log/postgresql/`

```
root@PostgreSQLSTG ~# cd /var/log/postgresql/
root@PostgreSQLSTG .../log/postgresql# ls
postgresql-13-main.log
root@PostgreSQLSTG .../log/postgresql#
```

5- configuration files

- postgresql.conf
  - check parameter of PostgreSQL inside pgSQL console
  - view parameter using pg_setting
  - view history of changes using pg_file_setting
- postgresql.autoconf
  - example 1 , edit of the parameter work mem
  - check if changes require restart from psql cli
  - does change in parameter reflect on postgresql.conf file .
  - reset all
- pg_ident.conf
- pg_hba.conf

postgresql.conf

- file contain parameter to help configure and manage performance of the database server
- when we run `initd` command to create cluster it create `postgresql.conf` file default
- the file allow one parameter per the line format
- parameter that require restart are clearly marked on the file
  many parameter need a server restart to take affect

the file is located in data directory of the cluster

default is located in `/etc/postgresql/13/main/`

```
root@PostgreSQLSTG .../13/main# cd /etc/postgresql/13/main/
root@PostgreSQLSTG .../13/main# ls
conf.d  environment  pg_ctl.conf  pg_hba.conf  pg_ident.conf  postgresql.conf  start.conf
root@PostgreSQLSTG .../13/main#
```

```
  GNU nano 5.4                                    postgresql.conf
# - Connection Settings -

listen_addresses = '*'          # what IP address(es) to listen on;
                                # comma-separated list of addresses;
                                # defaults to 'localhost'; use '*' for all
                                # (change requires restart)
port = 5432                     # (change requires restart)
max_connections = 100           # (change requires restart)
#superuser_reserved_connections = 3     # (change requires restart)
unix_socket_directories = '/var/run/postgresql' # comma-separated list of directories
                                # (change requires restart)
#unix_socket_group = ''         # (change requires restart)
#unix_socket_permissions = 0777         # begin with 0 to use octal notation
                                # (change requires restart)
#bonjour = off                  # advertise server via Bonjour
                                # (change requires restart)
#bonjour_name = ''              # defaults to the computer name
                                # (change requires restart)


# - TCP settings -
# see "man tcp" for details

#tcp_keepalives_idle = 0        # TCP_KEEPIDLE, in seconds;
                                # 0 selects the system default
#tcp_keepalives_interval = 0    # TCP_KEEPINTVL, in seconds;

^G Help         ^O Write Out    ^W Where Is     ^K Cut          ^T Execute      ^C Location     M-U Undo        M-A Set Mark
^X Exit         ^R Read File    ^\ Replace      ^U Paste        ^J Justify      ^_ Go To Line   M-E Redo        M-6 Copy
```

check parameter of PostgreSQL inside pgSQL console

the parameter you saw in the file can also be viewed in the psql console itself
to check and verified whether they enabled or not , and what is the parameter set for it
for example in the file there parameter called max_connection , i can see the value of it inside psql console by using `show max_connection`

```
  GNU nano 5.4                                    postgresql.conf
external_pid_file = '/var/run/postgresql/13-main.pid'                   # write an extra PID file
                                # (change requires restart)


#------------------------------------------------------------------------------
# CONNECTIONS AND AUTHENTICATION
#------------------------------------------------------------------------------

# - Connection Settings -

listen_addresses = '*'          # what IP address(es) to listen on;
                                # comma-separated list of addresses;
                                # defaults to 'localhost'; use '*' for all
                                # (change requires restart)
port = 5432                     # (change requires restart)
max_connections = 100           # (change requires restart)
#superuser_reserved_connections = 3     # (change requires restart)
unix_socket_directories = '/var/run/postgresql' # comma-separated list of directories
                                # (change requires restart)
#unix_socket_group = ''         # (change requires restart)
#unix_socket_permissions = 0777         # begin with 0 to use octal notation
                                # (change requires restart)
#bonjour = off                  # advertise server via Bonjour
                                # (change requires restart)
#bonjour_name = ''              # defaults to the computer name

^G Help         ^O Write Out    ^W Where Is     ^K Cut          ^T Execute      ^C Location     M-U Undo        M-A Set Mark
^X Exit         ^R Read File    ^\ Replace      ^U Paste        ^J Justify      ^_ Go To Line   M-E Redo        M-6 Copy
```

```
root=# show max_connections ;
 max_connections
-----------------
 100
(1 row)

root=#
```

```
                                # (change requires restart)
#bonjour = off                  # advertise server via Bonjour
                                # (change requires restart)
#bonjour_name = ''              # defaults to the computer name


root@PostgreSQLSTG .../13/main# psql
psql (13.8 (Debian 13.8-0+deb11u1))
Type "help" for help.

root=# show max_connections ;
 max_connections
-----------------
 100
(1 row)

root=# show port ;
 port
------
 5432
(1 row)

root=# show  bonjour;
 bonjour
---------
 off
(1 row)

root=#
```

view parameter using pg_setting

pg_setting is table contain all the infromation about config setup of postgresql.conf
you can query it to get parameters set
for example the below query will show parameter that recently eddied and pending restart to apply

```
select name ,source,boot_val,pending_restart from pg_settings where name = 'max_connections';
```

to get the list of column in pg_setting we can use the below option , useful in case you want to build query for yourself

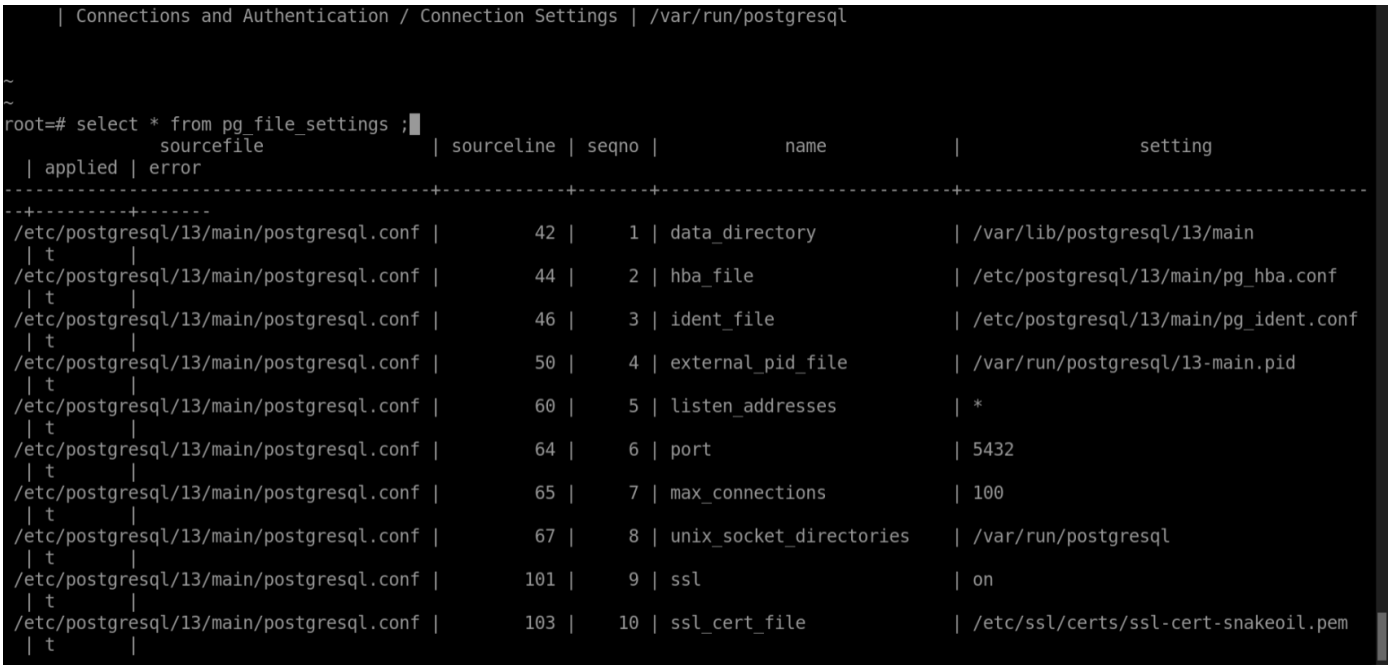
```
\d pg_setting
```

```
LINE 1: ... setting , unit , category from pg_settings where namme = "p...
                                                             ^
HINT:  Perhaps you meant to reference the column "pg_settings.name".
root=# select setting , unit , category from pg_settings where name = "postgresql.conf";
ERROR:  column "postgresql.conf" does not exist
LINE 1: ...g , unit , category from pg_settings where name = "postgresq...
                                                             ^
root=# \d pg_settings
              View "pg_catalog.pg_settings"
     Column      |  Type   | Collation | Nullable | Default
-----------------+---------+-----------+----------+---------
 name            | text    |           |          |
 setting         | text    |           |          |
 unit            | text    |           |          |
 category        | text    |           |          |
 short_desc      | text    |           |          |
 extra_desc      | text    |           |          |
 context         | text    |           |          |
 vartype         | text    |           |          |
 source          | text    |           |          |
 min_val         | text    |           |          |
 max_val         | text    |           |          |
 enumvals        | text[]  |           |          |
 boot_val        | text    |           |          |
 reset_val       | text    |           |          |
 sourcefile      | text    |           |          |
 sourceline      | integer |           |          |
 pending_restart | boolean |           |          |

root=#
```

`select name , setting , category , boot_val from pg_settings where sourcefile = '/etc/postgresql/13/main/postgresql.conf';`

view history of changes using pg_file_setting

this table contain log of any changes done on postgresql.conf file



```
    | Connections and Authentication / Connection Settings | /var/run/postgresql


~
~
root=# select * from pg_file_settings ;
              sourcefile               | sourceline | seqno |           name            |                 setting
  | applied | error
---------------------------------------+------------+-------+---------------------------+-----------------------------------
--+---------+-------
 /etc/postgresql/13/main/postgresql.conf |         42 |     1 | data_directory            | /var/lib/postgresql/13/main
  | t       |
 /etc/postgresql/13/main/postgresql.conf |         44 |     2 | hba_file                  | /etc/postgresql/13/main/pg_hba.conf
  | t       |
 /etc/postgresql/13/main/postgresql.conf |         46 |     3 | ident_file                | /etc/postgresql/13/main/pg_ident.conf
  | t       |
 /etc/postgresql/13/main/postgresql.conf |         50 |     4 | external_pid_file         | /var/run/postgresql/13-main.pid
  | t       |
 /etc/postgresql/13/main/postgresql.conf |         60 |     5 | listen_addresses          | *
  | t       |
 /etc/postgresql/13/main/postgresql.conf |         64 |     6 | port                      | 5432
  | t       |
 /etc/postgresql/13/main/postgresql.conf |         65 |     7 | max_connections           | 100
  | t       |
 /etc/postgresql/13/main/postgresql.conf |         67 |     8 | unix_socket_directories   | /var/run/postgresql
  | t       |
 /etc/postgresql/13/main/postgresql.conf |        101 |     9 | ssl                       | on
  | t       |
 /etc/postgresql/13/main/postgresql.conf |        103 |    10 | ssl_cert_file             | /etc/ssl/certs/ssl-cert-snakeoil.pem
  | t       |
```

postgresql.autoconf

this file have parameter that allow to modify the postgresql.conf parameter from cli rather then editing the file itself

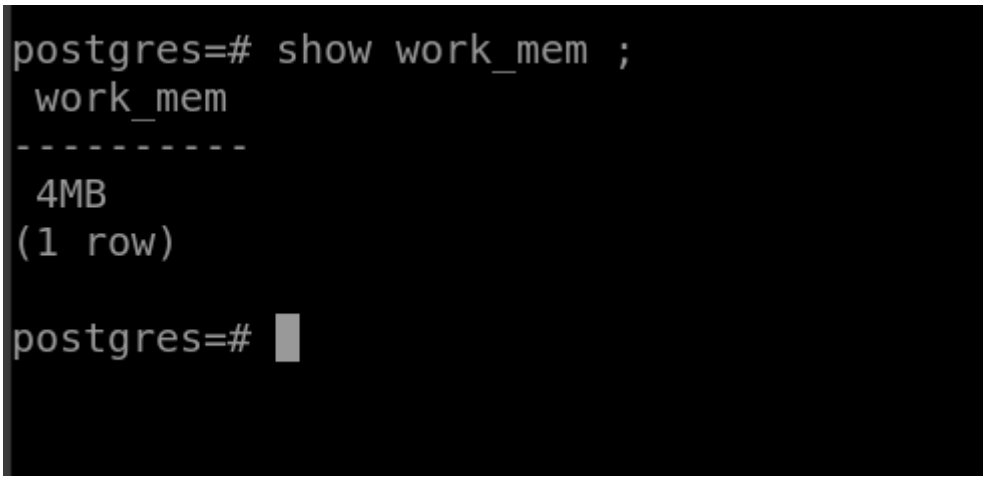
the editing of postgresql.conf is very critical and better to edit it from psql command line using `alter command`
keeping in mind some parameter require restart to reflect in the postgresql

example 1 , edit of the parameter work mem

this parameter we will edit it form pgSQL cli

to view the current value use the show command and the parameter
syntax

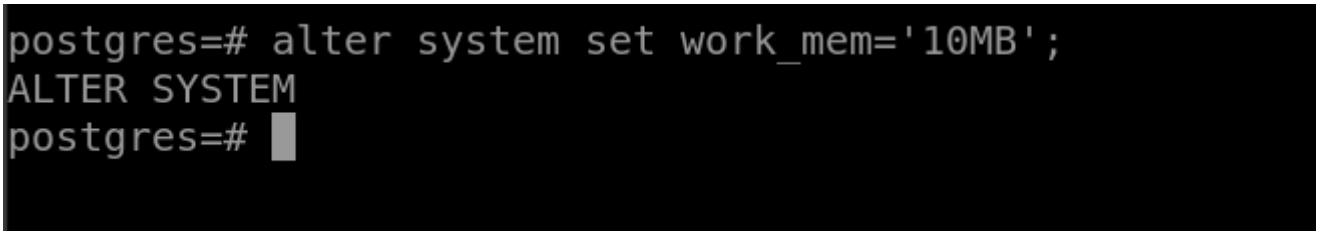
```
show [paramter in postgresql.conf];
show work_mem ;
```



to alter command we use `alter system set`

syntax :

`alter system set [parameter] = '[new value ]'; alter system set work_mem='10MB';`

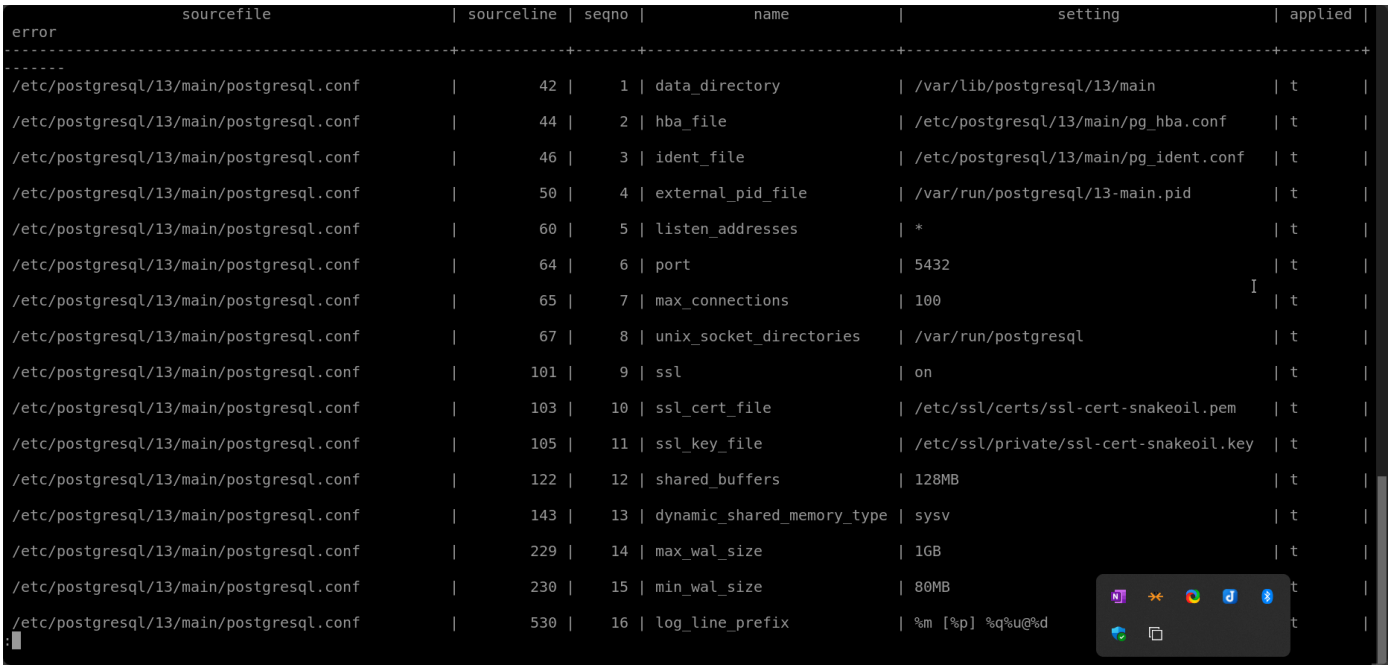


check if changes require restart from psql cli

as mentioned before some changes require restart , for instance work_mem dosnt require restart .
to check if the change require restart go to postgresql.conf file and in there you will file comment 'restart required '

or from cli itself you can check it by using the below query .

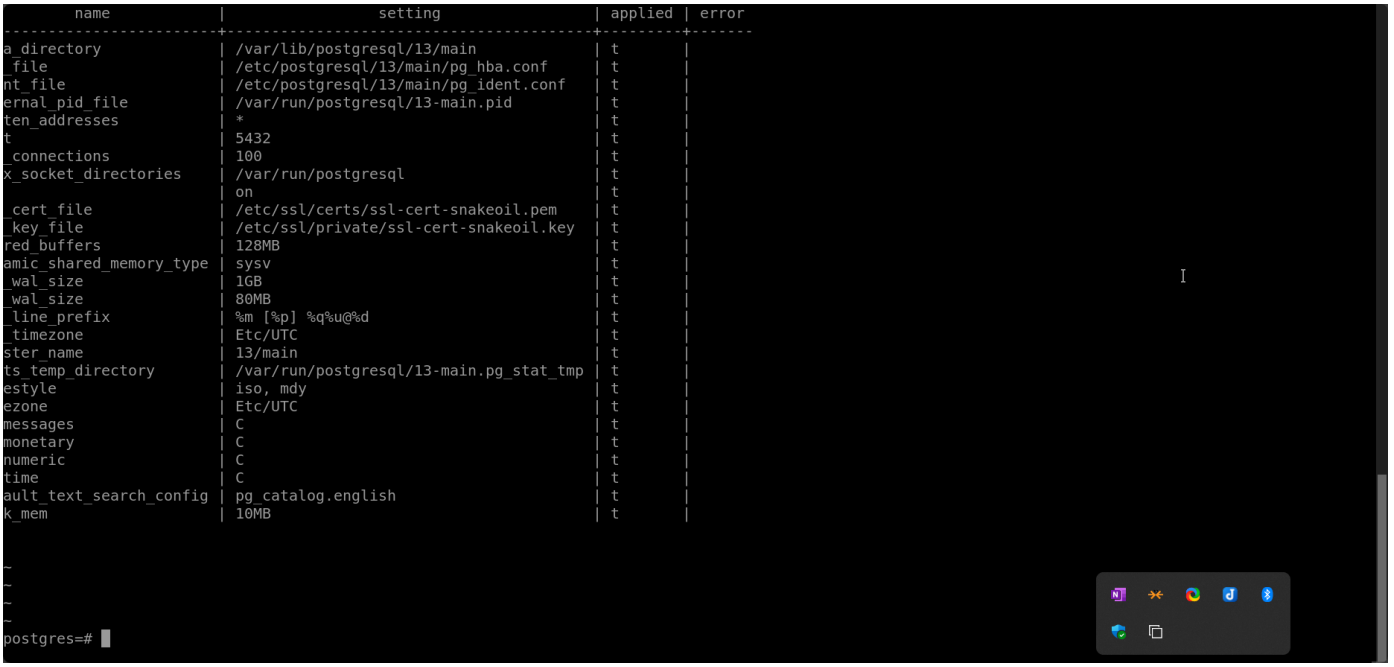
```
select * from pg_file_settings;
```



```
                sourcefile                  | sourceline | seqno |           name            |                setting                 | applied |
 error
--------------------------------------------+------------+-------+---------------------------+----------------------------------------+---------+
-------
 /etc/postgresql/13/main/postgresql.conf    |         42 |     1 | data_directory            | /var/lib/postgresql/13/main            | t       |
 /etc/postgresql/13/main/postgresql.conf    |         44 |     2 | hba_file                  | /etc/postgresql/13/main/pg_hba.conf    | t       |
 /etc/postgresql/13/main/postgresql.conf    |         46 |     3 | ident_file                | /etc/postgresql/13/main/pg_ident.conf  | t       |
 /etc/postgresql/13/main/postgresql.conf    |         50 |     4 | external_pid_file         | /var/run/postgresql/13-main.pid        | t       |
 /etc/postgresql/13/main/postgresql.conf    |         60 |     5 | listen_addresses          | *                                      | t       |
 /etc/postgresql/13/main/postgresql.conf    |         64 |     6 | port                      | 5432                                   | t       |
 /etc/postgresql/13/main/postgresql.conf    |         65 |     7 | max_connections           | 100                                    | t       |
 /etc/postgresql/13/main/postgresql.conf    |         67 |     8 | unix_socket_directories   | /var/run/postgresql                    | t       |
 /etc/postgresql/13/main/postgresql.conf    |        101 |     9 | ssl                       | on                                     | t       |
 /etc/postgresql/13/main/postgresql.conf    |        103 |    10 | ssl_cert_file             | /etc/ssl/certs/ssl-cert-snakeoil.pem   | t       |
 /etc/postgresql/13/main/postgresql.conf    |        105 |    11 | ssl_key_file              | /etc/ssl/private/ssl-cert-snakeoil.key | t       |
 /etc/postgresql/13/main/postgresql.conf    |        122 |    12 | shared_buffers            | 128MB                                  | t       |
 /etc/postgresql/13/main/postgresql.conf    |        143 |    13 | dynamic_shared_memory_type | sysv                                   | t       |
 /etc/postgresql/13/main/postgresql.conf    |        229 |    14 | max_wal_size              | 1GB                                    | t       |
 /etc/postgresql/13/main/postgresql.conf    |        230 |    15 | min_wal_size              | 80MB                                   | t       |
 /etc/postgresql/13/main/postgresql.conf    |        530 |    16 | log_line_prefix           | %m [%p] %q%u@%d                        | t       |
:
```

applying column is showing 't' meaning the changes is applied

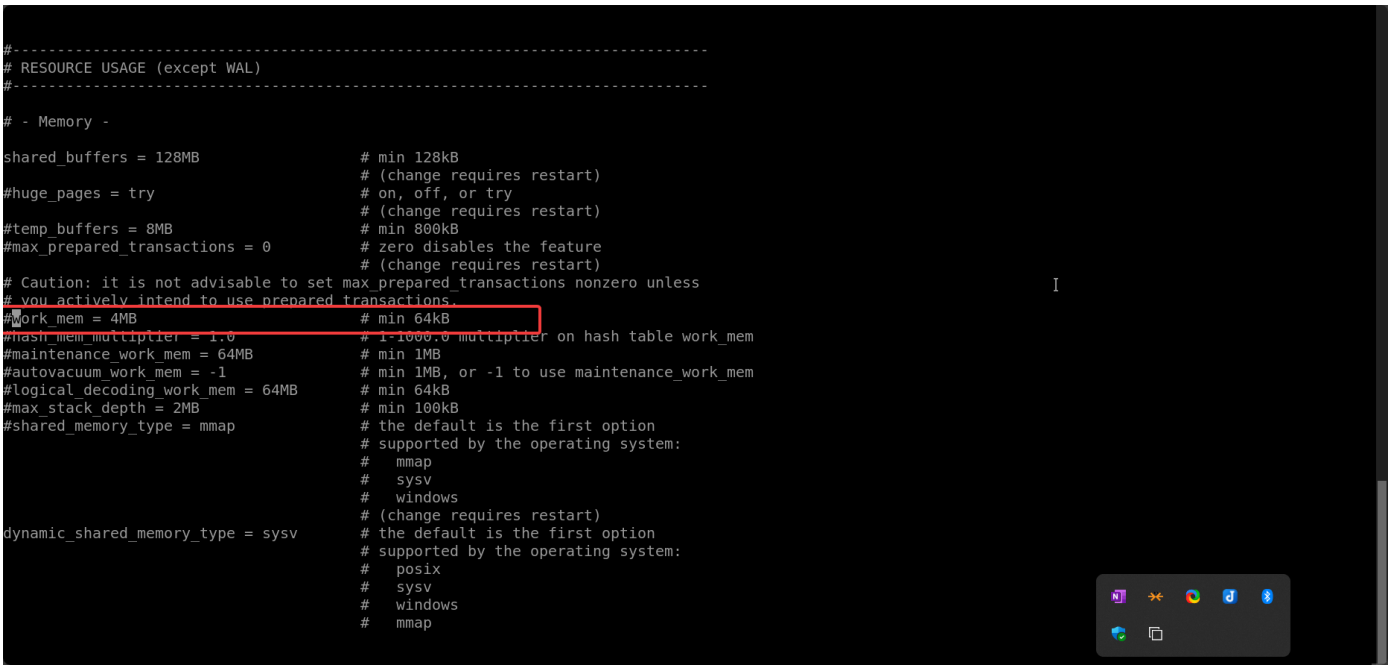
there is column called error if its empty means no restart required if there s enters in the column means restart is required .



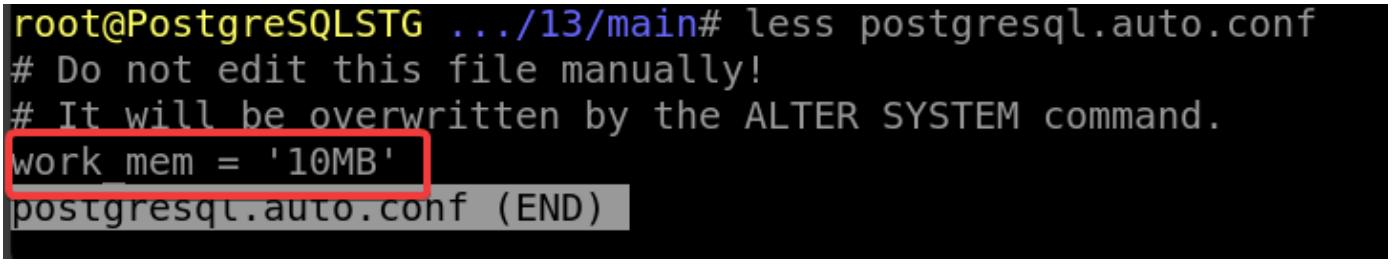
```
       name          |                setting                 | applied | error
---------------------+----------------------------------------+---------+-------
a_directory          | /var/lib/postgresql/13/main            | t       |
_file                | /etc/postgresql/13/main/pg_hba.conf    | t       |
nt_file              | /etc/postgresql/13/main/pg_ident.conf  | t       |
ernal_pid_file       | /var/run/postgresql/13-main.pid        | t       |
ten_addresses        | *                                      | t       |
t                    | 5432                                   | t       |
_connections         | 100                                    | t       |
x_socket_directories | /var/run/postgresql                    | t       |
                     | on                                     | t       |
_cert_file           | /etc/ssl/certs/ssl-cert-snakeoil.pem   | t       |
_key_file            | /etc/ssl/private/ssl-cert-snakeoil.key | t       |
red_buffers          | 128MB                                  | t       |
amic_shared_memory_type | sysv                                | t       |
_wal_size            | 1GB                                    | t       |
_wal_size            | 80MB                                   | t       |
_line_prefix         | %m [%p] %q%u@%d                        | t       |
_timezone            | Etc/UTC                                | t       |
ster_name            | 13/main                                | t       |
ts_temp_directory    | /var/run/postgresql/13-main.pg_stat_tmp | t      |
estyle               | iso, mdy                               | t       |
ezone                | Etc/UTC                                | t       |
messages             | C                                      | t       |
monetary             | C                                      | t       |
numeric              | C                                      | t       |
time                 | C                                      | t       |
ault_text_search_config | pg_catalog.english                  | t       |
k_mem                | 10MB                                   | t       |


~
~
~
~
postgres=#
```

does change in parameter reflect on postgresql.conf file .

we have changed the parameter fo work_mem from 3 to 10 let's check if its reflected on postgresql.cnf file



the value hasn't changed why ?
acuity the changes is not edited in postgresql.conf file , it will be available in postgresql.autoconf
if you checked it you will find the changes you did for work_mem
the file will be located in the data directory of the db-cluster
/var/lib/postgresql/13/main/
if you check the content of the file you will find the new value for work_mem



when postgresql services start it will load config from postgresql.conf

then it will load config in the postgresql.auto.conf

postgresql will load the value in postgresql.auto.conf and avoid load the value in postgresql.conf

reset all

if you want to reset all the changes you did in psql cli using alter system
command
you can use the below query

```
alter  system reset all ;
```

this means all values in postgresql.auto.onf will be removed
but keep in mind it will not reset values in postgresql.conf

only changes done using alter system command will be reset

```
psql (13.8 (Debian 13.8-0+deb11u1))
Type "help" for help.

postgres=# alter system reset all ;
ALTER SYSTEM
postgres=#
```

```
postgres=# show work_mem
postgres-# ;
 work_mem
----------
 4MB
(1 row)

postgres=#
```

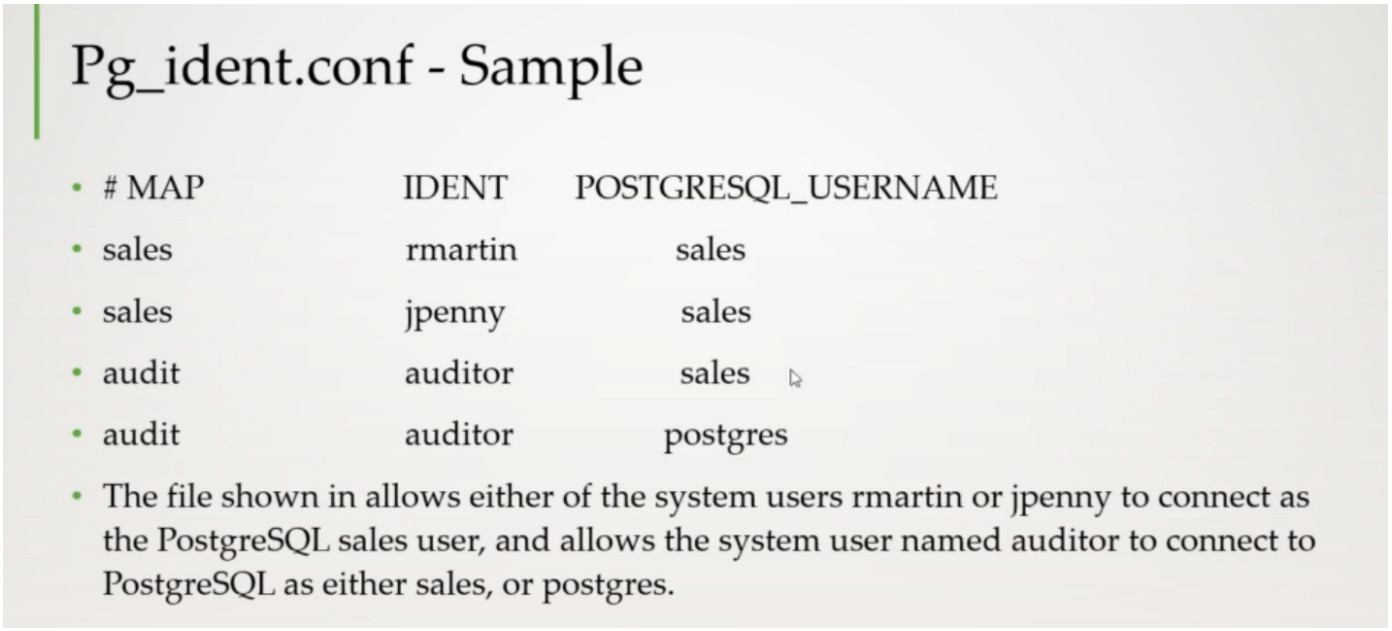
```
root@PostgreSQLSTG .../13/main# less postgresql.auto.conf
# Do not edit this file manually!
# It will be overwritten by the ALTER SYSTEM command.
postgresql.auto.conf (END)
```

- in short postgresql.auto.conf is useful to edit config of PostgreSQL without touching the postgresql.conf file

pg _ident.conf

the file is part of authenticating section file of PostgreSQL and will allow to map OS user with PostgreSQL user
this file allow to match the user in PostgreSQL database with username in OS level

any changes to the file require a reload only



the file is located in /etc/postgresql/13/main/

i have PostgreSQL user called ahmed , but the user is not there in OS so i will create it and map it to user called postres in PostgreSQL

in the file you will asked to give friendly name for map , then add the OS username in Identity , then postgresql_username add PostgreSQL username
after that you need to reload PostgreSQL

```
# Put your actual configuration here
# ----------------------------------

# MAPNAME       SYSTEM-USERNAME         PG-USERNAME
superuser       ahmed                   ahmed


root@PostgreSQLSTG .../13/main# systemctl reload postgresql
root@PostgreSQLSTG .../13/main# su - ahmed
su: warning: cannot change directory to /home/ahmed: No such file or directory
$ psql
psql (13.8 (Debian 13.8-0+deb11u1))
Type "help" for help.

ahmed=#
```

pg_hba.conf

enables clients authentication between PostgreSQL server and the client application

HBA means host based authenticating .

Pg_hba.conf - Sample

```
# TYPE          DATABASE      USER          ADDRESS              METHOD
# IPv4 local connections:
   host          all           all           127.0.0.1/32         md5
# IPv6 local connections:
   host          all           all           ::1/128              trust
# Allow replication connections from localhost, by a user with the
# replication privilege.
   host          replication   all           127.0.0.1/32         trust
   host          replication   all           ::1/128              trust
```

6. create object (database/user/schema)

Create.docx

creating database

in PostgreSQL you can create database through ways

1. through command line using `createdb`
2. through psql using `create database [database_name];`

createdb utility

syntax

`createdb [database name]`

you can use `--help` to check for more option

```
root@PostgreSQLSTG ~# createdb --help
createdb creates a PostgreSQL database.

Usage:
  createdb [OPTION]... [DBNAME] [DESCRIPTION]

Options:
  -D, --tablespace=TABLESPACE  default tablespace for the database
  -e, --echo                   show the commands being sent to the server
  -E, --encoding=ENCODING      encoding for the database
  -l, --locale=LOCALE          locale settings for the database
      --lc-collate=LOCALE      LC_COLLATE setting for the database
      --lc-ctype=LOCALE        LC_CTYPE setting for the database
  -O, --owner=OWNER            database user to own the new database
  -T, --template=TEMPLATE      template database to copy
  -V, --version                output version information, then exit
  -?, --help                   show this help, then exit

Connection options:
  -h, --host=HOSTNAME          database server host or socket directory
  -p, --port=PORT              database server port
  -U, --username=USERNAME      user name to connect as
  -w, --no-password            never prompt for password
  -W, --password               force password prompt
  --maintenance-db=DBNAME      alternate maintenance database

By default, a database with the same name as the current user is created.

Report bugs to <pgsql-bugs@lists.postgresql.org>.
PostgreSQL home page: <https://www.postgresql.org/>
```

to use the command you need to witch to postgres user

or use `-U` option and create database

1.using `-U` option

i will create database called asus i am login in using root user

so the syntax is below

```
~$ createdb -U postgres asus
```

```
root@PostgreSQLSTG ~# createdb -U postgres asus
Password:
root@PostgreSQLSTG ~#
```

2. using postgres user

   if you switch to postgres user no need to use -U option directy isssue the command

```
~$ createdb lenovo
```

```
postgres@PostgreSQLSTG:~$ createdb lenovo
Password:
postgres@PostgreSQLSTG:~$
```

```
postgres=# \l
                                 List of databases
   Name    |  Owner   | Encoding | Collate | Ctype |   Access privileges
-----------+----------+----------+---------+-------+-----------------------
 ahmed     | root     | UTF8     | C       | C     |
 asus      | postgres | UTF8     | C       | C     |
 empolyee  | postgres | UTF8     | C       | C     |
 lenovo    | postgres | UTF8     | C       | C     |
 postgres  | postgres | UTF8     | C       | C     |
 root      | postgres | UTF8     | C       | C     |
 template0 | postgres | UTF8     | C       | C     | =c/postgres          +
           |          |          |         |       | postgres=CTc/postgres
 template1 | postgres | UTF8     | C       | C     | =c/postgres          +
           |          |          |         |       | postgres=CTc/postgres
(8 rows)
```

create database from psql

the syntax :

```
create database [database_name] owner [username];
```

if you don't specify the owner then automatically the owner will be the user that issue the command .

```
postgres=# create database roger owner ahmed;
CREATE DATABASE
postgres=# \l
                                 List of databases
   Name    |  Owner   | Encoding | Collate | Ctype |   Access privileges
-----------+----------+----------+---------+-------+-----------------------
 ahmed     | root     | UTF8     | C       | C     |
 asus      | postgres | UTF8     | C       | C     |
 empolyee  | postgres | UTF8     | C       | C     |
 lenovo    | postgres | UTF8     | C       | C     |
 postgres  | postgres | UTF8     | C       | C     |
 roger     | ahmed    | UTF8     | C       | C     |
 root      | postgres | UTF8     | C       | C     |
 template0 | postgres | UTF8     | C       | C     | =c/postgres          +
           |          |          |         |       | postgres=CTc/postgres
 template1 | postgres | UTF8     | C       | C     | =c/postgres          +
           |          |          |         |       | postgres=CTc/postgres
(9 rows)

postgres=#
```

checking the OIDs

once you create the database it will get the OID , to heck the OID of the database use the below query

```
select datname,oid from pg_database;
```

```
postgres=# select datname,oid from pg_database;
  datname  |  oid
-----------+-------
 postgres  | 13445
 root      | 16385
 template1 |     1
 template0 | 13444
 empolyee  | 16387
 ahmed     | 16412
 asus      | 16418
 lenovo    | 16419
 roger     | 16420
(9 rows)

postgres=#
```

connect to database directory

you can connect directory to database by using the below syntax

```
$ psql -U postgre [databse_name]
```

```
root@PostgreSQLSTG ~# psql -U postgres asus
Password for user postgres:
psql (13.8 (Debian 13.8-0+deb11u1))
Type "help" for help.

asus=#
```

check what database you connected to !

you can use the below option and it will display the connection infor including data directory and port

`\conninfo`

```
ahmed=# \conninfo
You are connected to database "ahmed" as user "ahmed" via socket in "/var/run/postgresql" at port "5432".
```

drop database

to drop the database same as create you can do it from command line .

from psql

```
#drop database [database_name]
```

```
ahmed=# drop database asus;
DROP DATABASE
ahmed=# \l
                              List of databases
   Name    |  Owner   | Encoding | Collate | Ctype |   Access privileges
-----------+----------+----------+---------+-------+-----------------------
 ahmed     | root     | UTF8     | C       | C     |
 empolyee  | postgres | UTF8     | C       | C     |
 lenovo    | postgres | UTF8     | C       | C     |
 postgres  | postgres | UTF8     | C       | C     |
 roger     | ahmed    | UTF8     | C       | C     |
 root      | postgres | UTF8     | C       | C     |
 template0 | postgres | UTF8     | C       | C     | =c/postgres          +
           |          |          |         |       | postgres=CTc/postgres
 template1 | postgres | UTF8     | C       | C     | =c/postgres          +
           |          |          |         |       | postgres=CTc/postgres
(8 rows)

ahmed=#
```

dropdb utility

this utility allow you to drop database from the command line

```
root$dropdb -U postgres [database_name]
postgres$ dropdb [database_name]
```

```
root@PostgreSQLSTG ~# dropdb -U postgres lenovo
Password:
root@PostgreSQLSTG ~# psql -U postgres
Password for user postgres:
psql (13.8 (Debian 13.8-0+deb11u1))
Type "help" for help.

postgres=# \l
                              List of databases
   Name    |  Owner   | Encoding | Collate | Ctype |   Access privileges
-----------+----------+----------+---------+-------+-----------------------
 ahmed     | root     | UTF8     | C       | C     |
 empolyee  | postgres | UTF8     | C       | C     |
 postgres  | postgres | UTF8     | C       | C     |
 roger     | ahmed    | UTF8     | C       | C     |
 root      | postgres | UTF8     | C       | C     |
 template0 | postgres | UTF8     | C       | C     | =c/postgres          +
           |          |          |         |       | postgres=CTc/postgres
 template1 | postgres | UTF8     | C       | C     | =c/postgres          +
           |          |          |         |       | postgres=CTc/postgres
(7 rows)

postgres=#
```

create user

while creating the user it important to note that username must be unique
the username should not start with 'pg_'
the user postgres is created during the installation automatically

it hold the privilege of 'super user ' allow it to users with role privileges
the user postgres has all the privilege with grant option .

only super user can create users

to create user you have to way similar to create database

1. from command line using createuser utility
2. from psql console

create user using createuser utility

as usual you can use `--help` to check deferent option you can use
for this porpoise i will create user which not super user and i will specify password for it

```
root@PostgreSQLSTG ~# createuser --help
createuser creates a new PostgreSQL role.

Usage:
  createuser [OPTION]... [ROLENAME]

Options:
  -c, --connection-limit=N  connection limit for role (default: no limit)
  -d, --createdb            role can create new databases
  -D, --no-createdb         role cannot create databases (default)
  -e, --echo                show the commands being sent to the server
  -g, --role=ROLE           new role will be a member of this role
  -i, --inherit             role inherits privileges of roles it is a
                            member of (default)
  -I, --no-inherit          role does not inherit privileges
  -l, --login               role can login (default)
  -L, --no-login            role cannot login
  -P, --pwprompt            assign a password to new role
  -r, --createrole          role can create new roles
  -R, --no-createrole       role cannot create roles (default)
  -s, --superuser           role will be superuser
  -S, --no-superuser        role will not be superuser (default)
  -V, --version             output version information, then exit
  --interactive             prompt for missing role name and attributes rather
                            than using defaults
  --replication             role can initiate replication
  --no-replication          role cannot initiate replication
  -?, --help                show this help, then exit

Connection options:
  -h, --host=HOSTNAME       database server host or socket directory
  -p, --port=PORT           database server port
  -U, --username=USERNAME   user name to connect as (not the one to create)
```

so the syntax is

```
root@PostgreSQLSTG ~# createuser -U postgres -P -S [username]
postgres@PostgreSQLSTG ~# createuser  -P -S [username]
```

```
root@PostgreSQLSTG ~# createuser -U postgres -P -S superadmin
Enter password for new role:
Enter it again:
Password:
root@PostgreSQLSTG ~# psql -U postgres
Password for user postgres:
psql (13.8 (Debian 13.8-0+deb11u1))
Type "help" for help.

postgres=# \du
                                   List of roles
 Role name  |                         Attributes                         | Member of
------------+------------------------------------------------------------+-----------
 abdullah   |                                                            | {}
 ahmed      | Superuser                                                  | {}
 postgres   | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
 root       | Superuser, Create role, Create DB                          | {}
 salah      | Superuser, Create role, Create DB                          | {}
 superadmin |                                                            | {}

postgres=#
```

using --interactive

for more interactive opting using createuser utility you can specify `--interactive` option and it will ask you for deterrent parameter to which you have to specify and this option for ease of use

`createuer --interactive`

```
root@PostgreSQLSTG ~# createuser -U postgres --interactive
Enter name of role to add: messi
Shall the new role be a superuser? (y/n) n
Shall the new role be allowed to create databases? (y/n) y
Shall the new role be allowed to create more new roles? (y/n) n
Password:
root@PostgreSQLSTG ~# psql -U postgres
Password for user postgres:
psql (13.8 (Debian 13.8-0+deb11u1))
Type "help" for help.

postgres=# \du
                                   List of roles
 Role name  |                         Attributes                         | Member of
------------+------------------------------------------------------------+-----------
 abdullah   |                                                            | {}
 ahmed      | Superuser                                                  | {}
 john       | Superuser                                                  | {}
 khalid     | Superuser, Create role, Create DB                          | {}
 messi      | Create DB                                                  | {}
 postgres   | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
 root       | Superuser, Create role, Create DB                          | {}
 salah      | Superuser, Create role, Create DB                          | {}
 superadmin |                                                            | {}

postgres=#
```

```
root@PostgreSQLSTG ~# createuser -U postgres --interactive
Enter name of role to add: khalid
Shall the new role be a superuser? (y/n) y
Password:
root@PostgreSQLSTG ~# psql -U postgres
Password for user postgres:
psql (13.8 (Debian 13.8-0+deb11u1))
Type "help" for help.

postgres=# \du
                                   List of roles
 Role name  |                         Attributes                         | Member of
------------+------------------------------------------------------------+-----------
 abdullah   |                                                            | {}
 ahmed      | Superuser                                                  | {}
 john       | Superuser                                                  | {}
 khalid     | Superuser, Create role, Create DB                          | {}
 postgres   | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
 root       | Superuser, Create role, Create DB                          | {}
 salah      | Superuser, Create role, Create DB                          | {}
 superadmin |                                                            | {}

postgres=#
```

create user from psql

syntax

```
create user [username] login [privilege] password [enter the password ];


create user john login superuser password '123456789';
```

```
postgres=# create user john login superuser password '123456789';
CREATE ROLE
postgres=# \du
                                   List of roles
 Role name  |                         Attributes                         | Member of
------------+------------------------------------------------------------+-----------
 abdullah   |                                                            | {}
 ahmed      | Superuser                                                  | {}
 john       | Superuser                                                  | {}
 postgres   | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
 root       | Superuser, Create role, Create DB                          | {}
 salah      | Superuser, Create role, Create DB                          | {}
 superadmin |                                                            | {}

postgres=#
```

drop user

there two way from command line and from psql

from command line

we will use the dropuser command

```
$ dropuser -U postgres [username]
```

```
root@PostgreSQLSTG ~# dropuser -U postgres messi
Password:
root@PostgreSQLSTG ~#
```

from psql console

```
drop user [username];
```

```
postgres=# drop user  john;
DROP ROLE
postgres=# \du
                                   List of roles
 Role name  |                         Attributes                         | Member of
------------+------------------------------------------------------------+-----------
 abdullah   |                                                            | {}
 ahmed      | Superuser                                                  | {}
 khalid     | Superuser, Create role, Create DB                          | {}
 postgres   | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
 root       | Superuser, Create role, Create DB                          | {}
 salah      | Superuser, Create role, Create DB                          | {}
 superadmin |                                                            | {}

postgres=#
```

privilege

privilege is right to execute particular SQL statement , or a right to access another user objects .

there two type of privilege
1.**cluster level privilege** are granted by superuser .
2.**object level privileges** are granted by superuser or the owner of the object or someone with grant privilege

cluster level privileges

when you issue the `\du` in pgSQL you will see the users with cluster level privileges

i have user name Ahmed i want to revoke his superuser

```
postgres=# \du
                                   List of roles
 Role name  |                         Attributes                         | Member of
------------+------------------------------------------------------------+-----------
 abdullah   |                                                            | {}
 ahmed      | Superuser                                                  | {}
 khalid     | Superuser, Create role, Create DB                          | {}
 postgres   | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
 root       | Superuser, Create role, Create DB                          | {}
 salah      | Superuser, Create role, Create DB                          | {}
 superadmin |                                                            | {}

postgres=#
```

revoke superuser privilege

to revoke the privileges we will use the below syntax

```
alter user [uer_name] with nosuperuser;
```

```
postgres=# alter user ahmed  with nosuperuser ;
ALTER ROLE
postgres=# \du
                                   List of roles
 Role name  |                         Attributes                         | Member of
------------+------------------------------------------------------------+-----------
 abdullah   |                                                            | {}
 ahmed      |                                                            | {}
 khalid     | Superuser, Create role, Create DB                          | {}
 postgres   | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
 root       | Superuser, Create role, Create DB                          | {}
 salah      | Superuser, Create role, Create DB                          | {}
 superadmin |                                                            | {}

postgres=#
```

to grant him the superuser privilege again
we will use the below syntax

```
alter user [username] with superuser ;
```

```
postgres=# alter user ahmed with superuser
postgres-# ;
ALTER ROLE
postgres=# \du
                                   List of roles
 Role name  |                         Attributes                         | Member of
------------+------------------------------------------------------------+-----------
 abdullah   |                                                            | {}
 ahmed      | Superuser                                                  | {}
 khalid     | Superuser, Create role, Create DB                          | {}
 postgres   | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
 root       | Superuser, Create role, Create DB                          | {}
 salah      | Superuser, Create role, Create DB                          | {}
 superadmin |                                                            | {}

postgres=#
```

grant a role to a user

for instance we want to grant user createdb privilege

syntax

```
alter user ahmed createdb;
```

```
postgres=# alter user ahmed createdb ;
ALTER ROLE
postgres=# \du
                                   List of roles
 Role name  |                         Attributes                         | Member of
------------+------------------------------------------------------------+-----------
 abdullah   |                                                            | {}
 ahmed      | Superuser, Create DB                                       | {}
 khalid     | Superuser, Create role, Create DB                          | {}
 postgres   | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
 root       | Superuser, Create role, Create DB                          | {}
 salah      | Superuser, Create role, Create DB                          | {}
 superadmin |                                                            | {}

postgres=# alter user ahmed create role ;
ERROR:  syntax error at or near "create"
LINE 1: alter user ahmed create role ;
                         ^
postgres=# alter user ahmed createrole ;
ALTER ROLE
postgres=# \du
                                   List of roles
 Role name  |                         Attributes                         | Member of
------------+------------------------------------------------------------+-----------
 abdullah   |                                                            | {}
 ahmed      | Superuser, Create role, Create DB                          | {}
 khalid     | Superuser, Create role, Create DB                          | {}
 postgres   | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
 root       | Superuser, Create role, Create DB                          | {}
 salah      | Superuser, Create role, Create DB                          | {}
 superadmin |                                                            | {}

postgres=#
```

object level privileges

first i will connect to database directory

when you create table the owner will be the user who created the table other user will not be able to select from the table so to solve this you will have to grant select privilege to the user

```
grant select on [table_name ] to []
```

7.table inheritance , table partion

table inheritance

To convey the idea that child tables inherit all columns from the parent master table while also allowing child tables to have additional fields, with the ability to specify the "**ONLY**" keyword for queries on specific tables, you can rewrite the statement as follows:

"We enable child tables to inherit all columns from the parent master table, permitting them to include additional fields unique to each child table. Furthermore, the 'ONLY' keyword can be utilized to specify that a query should exclusively target a particular table and not affect any others.

Please note that any updates or deletions made to the parent table will automatically propagate to the child table due to their inheritance relationship.

for testing i have created a new database called testinhernttable
then i will create table called orders .

```
create table orders(orderno serial, flightname varchar(100),boarding varchar(100),status varchar(100),source varchar(100));
```

```
testinhernttable=# create table orders(orderno serial, flightname varchar(100),boarding varchar(100),status varchar(100),source varchar(100));
CREATE TABLE
testinhernttable=#
```

We will create a child table called 'online.book' and specify its inheritance using the INHERITS syntax.
`inherits(partent_table_name)`

```
create table online_booking (price int) inherits(orders);
```

```
testinhernttable=# create table online_booking (price int) inherits(orders);
CREATE TABLE
testinhernttable=#
```

To check the description of the child table and verify its inheritance from the parent "order" table, you can use the following phrase:

`\d online_booking`

"We will inspect the child table's metadata and confirm whether it inherits from the parent 'order' table."

```
testinhernttable=# \d online_booking
                                Table "public.online_booking"
   Column   |          Type          | Collation | Nullable |                 Default
------------+------------------------+-----------+----------+-----------------------------------------
 orderno    | integer                |           | not null | nextval('orders_orderno_seq'::regclass)
 flightname | character varying(100) |           |          |
 boarding   | character varying(100) |           |          |
 status     | character varying(100) |           |          |
 source     | character varying(100) |           |          |
 price      | integer                |           |          |
Inherits: orders

testinhernttable=#
```

you will also notice that child table inheritance all the column from parent table

```
testinhernttable=# \d orders
                                   Table "public.orders"
   Column   |          Type          | Collation | Nullable |                 Default
------------+------------------------+-----------+----------+-----------------------------------------
 orderno    | integer                |           | not null | nextval('orders_orderno_seq'::regclass)
 flightname | character varying(100) |           |          |
 boarding   | character varying(100) |           |          |
 status     | character varying(100) |           |          |
 source     | character varying(100) |           |          |
Number of child tables: 1 (Use \d+ to list them.)

testinhernttable=# \d online_booking
                                Table "public.online_booking"
   Column   |          Type          | Collation | Nullable |                 Default
------------+------------------------+-----------+----------+-----------------------------------------
 orderno    | integer                |           | not null | nextval('orders_orderno_seq'::regclass)
 flightname | character varying(100) |           |          |
 boarding   | character varying(100) |           |          |
 status     | character varying(100) |           |          |
 source     | character varying(100) |           |          |
 price      | integer                |           |          |
Inherits: orders

testinhernttable=#
```

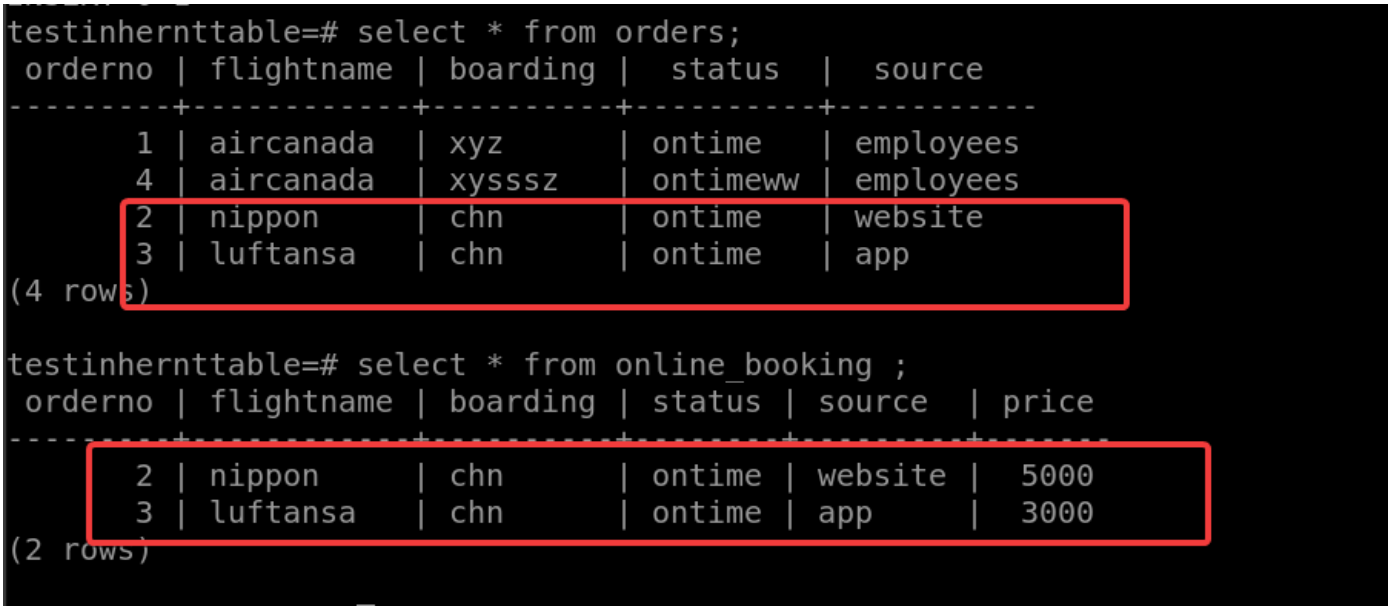
we can use the `\d+ order` on parent table and it will give more info on parent table and also will show if there child table

```
testinhernttable=# \d+ orders
                                                      Table "public.orders"
   Column   |          Type          | Collation | Nullable |                 Default                 | Storage  |
------------+------------------------+-----------+----------+-----------------------------------------+----------+-
 orderno    | integer                |           | not null | nextval('orders_orderno_seq'::regclass) | plain    |
 flightname | character varying(100) |           |          |                                         | extended |
 boarding   | character varying(100) |           |          |                                         | extended |
 status     | character varying(100) |           |          |                                         | extended |
 source     | character varying(100) |           |          |                                         | extended |
Child tables: online_booking
Access method: heap
```

now we will insert data to the table

```
insert into orders(flightname,boarding,status,source)
values('aircanada','xyz','ontime','employees');
insert into online_booking(flightname,boarding,status,source,price)
values('nippon','chn','ontime','website',5000);
insert into online_booking(flightname,boarding,status,source,price)
values('luftansa','chn','ontime','app',3000);
```

When querying the parent table, it will display values from both the parent and child tables. However, any columns that were added exclusively by the child table and are not inherited from the parent will not be included in the query results



updating values in parent table

in child table you can update with no requirement , but for updating parent table it require to be caution due to fact that any changes will reflect on child

to only update use `only` in syntax

```
update  only order  set status='delayed';
```

```
postgres=# \c testinhernttable
You are now connected to database "testinhernttable" as user "postgres".
testinhernttable=# update  only order  set status='delayed';
ERROR:  syntax error at or near "order"
LINE 1: update  only order  set status='delayed';
                    ^
testinhernttable=# update  only orders  set status='delayed';
UPDATE 2
testinhernttable=# select * from orders;
 orderno | flightname | boarding | status  |  source
---------+------------+----------+---------+-----------
       1 | aircanada  | xyz      | delayed | employees
       4 | aircanada  | xysssz   | delayed | employees
       2 | nippon     | chn      | ontime  | website
       3 | luftansa   | chn      | ontime  | app
(4 rows)

testinhernttable=# select * from online_booking ;
 orderno | flightname | boarding | status | source  | price
---------+------------+----------+--------+---------+-------
       2 | nippon     | chn      | ontime | website |  5000
       3 | luftansa   | chn      | ontime | app     |  3000
(2 rows)

testinhernttable=#
```

copy table

copy is used to copy structure of table alone with the data to another table

the way you implement this is by creating new table and using as table followed by existing table name below is the syntax with data

```
create table new_table as table exesting_table;
```

below is syntax without the data

```
create table new_table as table exesting_table with no data ;
```

```
testinhernttable=# \d orders
                                  Table "public.orders"
   Column   |          Type          | Collation | Nullable |                Default
------------+------------------------+-----------+----------+---------------------------------------
 orderno    | integer                |           | not null | nextval('orders_orderno_seq'::regclass)
 flightname | character varying(100) |           |          |
 boarding   | character varying(100) |           |          |
 status     | character varying(100) |           |          |
 source     | character varying(100) |           |          |
Number of child tables: 1 (Use \d+ to list them.)

testinhernttable=# create table filights as orde

testinhernttable=# create table filights as orde

testinhernttable=# create table filights as orde

testinhernttable=# create table flight as orders;
ERROR:  syntax error at or near "orders"
LINE 1: create table flight as orders;
                               ^
testinhernttable=# create table flight as  table orders;
SELECT 4
testinhernttable=# \d flight
                    Table "public.flight"
   Column   |          Type          | Collation | Nullable | Default
------------+------------------------+-----------+----------+---------
 orderno    | integer                |           |          |
 flightname | character varying(100) |           |          |
 boarding   | character varying(100) |           |          |
 status     | character varying(100) |           |          |
 source     | character varying(100) |           |          |
testinhernttable=# \d flight
                     Table "public.flight"
   Column   |          Type          | Collation | Nullable | Default
------------+------------------------+-----------+----------+---------
 orderno    | integer                |           |          |
 flightname | character varying(100) |           |          |
 boarding   | character varying(100) |           |          |
 status     | character varying(100) |           |          |
 source     | character varying(100) |           |          |

testinhernttable=# select * from flight
testinhernttable-# ;
 orderno | flightname | boarding | status  |  source
---------+------------+----------+---------+-----------
       1 | aircanada  | xyz      | delayed | employees
       4 | aircanada  | xysssz   | delayed | employees
       2 | nippon     | chn      | ontime  | website
       3 | luftansa   | chn      | ontime  | app
(4 rows)

testinhernttable=#
```

```
testinhernttable=# create table testflight as table orders with no da

testinhernttable=# create table testflight as table orders with no da

testinhernttable=# create table testflight as table orders WITH no data;
CREATE TABLE AS
testinhernttable=# \d testflight
                    Table "public.testflight"
   Column   |          Type          | Collation | Nullable | Default
------------+------------------------+-----------+----------+---------
 orderno    | integer                |           |          |
 flightname | character varying(100) |           |          |
 boarding   | character varying(100) |           |          |
 status     | character varying(100) |           |          |
 source     | character varying(100) |           |          |

testinhernttable=# select * from testflight ;
 orderno | flightname | boarding | status | source
---------+------------+----------+--------+--------
(0 rows)

testinhernttable=#
```

8. tablespace

PostgreSQL store data logically in tablespace and physically in datafiles
PostgreSQL uses a tablespace to map a logical name to a physical location on disk

tablespace allow the user to control the disk layout of PostgreSQL
when we create table or any object it goes directory to data directly by default

to avoid load on disk we can separate the data been stored in disk to be segregated to deterrent part of the disk.

default tablespaces

default comes with two out box tablespaces namely `pg_default` & `pg_global`.
the default location for tablespaces is data directory.

pg_default

is the default tablespace for storing all user data .
is also the default tablespace for tamplate1 & tamplate2 database

al newly created database uses pg_default tablespace , unless overridden by TABLESPACE CLAUSE while create database

pg_global.

is the default tablespace for storing global data .

creating tablespace.

to show the tablespace that are in the db_cluster you can use the below command

```
select * from pg_tablespace ;
```

```
postgres=# select * from pg_tablespace ;
 oid  |  spcname   | spcowner | spcacl | spcoptions
------+------------+----------+--------+------------
 1663 | pg_default |       10 |        |
 1664 | pg_global  |       10 |        |
(2 rows)
```

to create new tablespace first create directory where the tablespace will point the data to be saved to

syntax:

```
create tablespace [tablespace_name] location '[directory]'
```

```
create tablespace table1 location '/tablepace1';
```

when creating new directory you you will need to add required permission

```
chmod 700 [path]
chown postgres:postgres [path]
```

now when you want to create any new object such as table or database or even index you can specify the table space that will store the object

```
[query for creating new object] tablespace [tablespace_name];
create database testtablespace1 tablespace table1
```

if you check the directory of the tablespace you will find there is new file created

```
postgres=# create database testtablespace1 tablespace table1 ;
CREATE DATABASE
postgres=# \q
root@PostgreSQLSTG /# ls
bin   dev  home  lib32  libx32      media  opt         proc  run   srv  tablepace1  usr
boot  etc  lib   lib64  lost+found  mnt    postgresql  root  sbin  sys  tmp         var
root@PostgreSQLSTG /# cd tablepace1/
root@PostgreSQLSTG /tablepace1# ls
PG_13_202007201
root@PostgreSQLSTG /tablepace1#
```

```
root@PostgreSQLSTG /tablepace1# ls
PG_13_202007201
root@PostgreSQLSTG /tablepace1# ls -l
total 4
drwx------ 3 postgres postgres 4096 Dec 21 07:06 PG_13_202007201
root@PostgreSQLSTG /tablepace1# ls
PG_13_202007201
root@PostgreSQLSTG /tablepace1# cd PG_13_202007201/16456/
root@PostgreSQLSTG .../PG_13_202007201/16456# ls
112        13300      2187      2605      2612      2620  2668  2691      2833      3081      3431      3574      3604      4149  4168  827
113        13300_fsm  2224      2605_fsm  2612_fsm  2650  2669  2692      2834      3085      3433      3575      3605      4150  4169  828
1247       13300_vm   2328      2605_vm   2612_vm   2651  2670  2693      2835      3118      3439      3576      3606      4151  4170  PG_VERSION
1247_fsm   13302      2336      2606      2613      2652  2673  2696      2836      3119      3440      3596      3607      4152  4171  pg_filenode.map
1247_vm    13304      2337      2606_fsm  2615      2653  2674  2699      2837      3164      3455      3597      3608      4153  4172  pg_internal.init
1249       13305      2579      2606_vm   2615_fsm  2654  2675  2701      2838      3256      3456      3598      3609      4154  4173
1249_fsm   13305_fsm  2600      2607      2615_vm   2655  2678  2702      2838_fsm  3257      3456_fsm  3599      3712      4155  4174
1249_vm    13305_vm   2600_fsm  2607_fsm  2616      2656  2679  2703      2838_vm   3258      3456_vm   3600      3764      4156  5002
1255       13307      2600_vm   2607_vm   2616_fsm  2657  2680  2704      2839      3350      3466      3600_fsm  3764_fsm  4157  548
1255_fsm   13309      2601      2608      2616_vm   2658  2681  2753      2840      3351      3467      3600_vm   3764_vm   4158  549
1255_vm    13310      2601_fsm  2608_fsm  2617      2659  2682  2753_fsm  2840_fsm  3379      3468      3601      3766      4159  6102
1259       13310_fsm  2601_vm   2608_vm   2617_fsm  2660  2683  2753_vm   2840_vm   3380      3501      3601_fsm  3767      4160  6104
1259_fsm   13310_vm   2602      2609      2617_vm   2661  2684  2754      2841      3381      3502      3601_vm   3997      4161  6106
1259_vm    13312      2602_fsm  2609_fsm  2618      2662  2685  2755      2995      3394      3503      3602      4143      4162  6110
13295      13314      2602_vm   2609_vm   2618_fsm  2663  2686  2756      2996      3394_fsm  3534      3602_fsm  4144      4163  6111
13295_fsm  1417       2603      2610      2618_vm   2664  2687  2757      3079      3394_vm   3541      3602_vm   4145      4164  6112
13295_vm   1418       2603_fsm  2610_fsm  2619      2665  2688  2830      3079_fsm  3395      3541_fsm  3603      4146      4165  6113
13297      174        2603_vm   2610_vm   2619_fsm  2666  2689  2831      3079_vm   3429      3541_vm   3603_fsm  4147      4166  6117
13299      175        2604      2611      2619_vm   2667  2690  2832      3080      3430      3542      3603_vm   4148      4167  826
root@PostgreSQLSTG .../PG_13_202007201/16456#
```

move table between tablespaces

first we will check the table are assinge to which tablespace

`select * from pg_tables where tablename = 'staff';`

to move table from tablespace to another we use the below syntax

first i will check in which table one target table is assinge to

```
 public | film_category              | table    | postgres
 public | film_film_id_seq           | sequence | postgres
 public | film_list                  | view     | postgres
 public | inventory                  | table    | postgres
 public | inventory_inventory_id_seq | sequence | postgres
 public | language                   | table    | postgres
 public | language_language_id_seq   | sequence | postgres
 public | nicer_but_slower_film_list | view     | postgres
 public | payment                    | table    | postgres
 public | payment_payment_id_seq     | sequence | postgres
 public | rental                     | table    | postgres
 public | rental_rental_id_seq       | sequence | postgres
 public | sales_by_film_category     | view     | postgres
 public | sales_by_store             | view     | postgres
 public | staff                      | table    | postgres
 public | staff_list                 | view     | postgres
 public | staff_staff_id_seq         | sequence | postgres
 public | store                      | table    | postgres
 public | store_store_id_seq         | sequence | postgres
(35 rows)

dvdrental=# select * from pg_tables where tablename = 'store';
 schemaname | tablename | tableowner | tablespace | hasindexes | hasrules | hastriggers | rowsecurity
------------+-----------+------------+------------+------------+----------+-------------+-------------
 public     | store     | postgres   |            | t          | f        | t           | f
(1 row)

dvdrental=# select * from pg_tablespace ;
  oid  |  spcname   | spcowner | spcacl | spcoptions
-------+------------+----------+--------+------------
  1663 | pg_default |       10 |        |
  1664 | pg_global  |       10 |        |
 16455 | table1     |       10 |        |
(3 rows)

dvdrental=#
```

the table store is not assinge to any tablespace so if you found the tablespace column empty it means the table is assinge to the default tablespace

`alter table [table-name] set tablespace [tablespace-name];`

```
alter table store set tablespace table1;
```

```
dvdrental=# alter table store set tablespace table1
dvdrental-# ;
ALTER TABLE
dvdrental=# select * from pg_tables where tablename = 'store';
 schemaname | tablename | tableowner | tablespace | hasindexes | hasrules | hastriggers | rowsecurity
------------+-----------+------------+------------+------------+----------+-------------+-------------
 public     | store     | postgres   | table1     | t          | f        | t           | f
(1 row)

dvdrental=#
```

check the new path of the table

we have successfully move the table from default tablespace to new tablespace meaning that the table data is not located in default data path of PostgreSQL
to check and confirm that we can use the below query.

```
select pg_relation_filepath('store');
```

```
dvdrental=# select pg_rel

dvdrental=# select pg_relation_filepa

dvdrental=# select pg_relation_filepath('store');
           pg_relation_filepath
-----------------------------------------
 pg_tblspc/16455/PG_13_202007201/16457/16781
(1 row)

dvdrental=# \q
root@PostgreSQLSTG ~# ls
root@PostgreSQLSTG ~# cd /
root@PostgreSQLSTG /# ls
bin   dev  home  lib32  libx32      media  opt         proc  run   share  sys         tmp  var
boot  etc  lib   lib64  lost+found  mnt    postgresql  root  sbin  srv    tablepace1  usr
root@PostgreSQLSTG /# cd tablepace1/PG_13_202007201/1645
16456/ 16457/
root@PostgreSQLSTG /# cd tablepace1/PG_13_202007201/1645
16456/ 16457/
```

drop tablespace

note that you cannot drop tablespace if there is object in the tablespace
the syntax :

```
drop tablespace [table_space_name];
```

Temporary Tablespace in PostgreSQL

PostgreSQL create temporary tablespace for such action you require for completing a query example sorting query
temporary tablespace deosnt store any data and they are remove after you shutdown database .

creating temporary tablespace

syntax:

```
create tablespace [tablespace-choosen-name] owner [username] LOCATION
[filepath]
```

the owner parameter is not require to be added , incase you didn't specify the owner the temp tablespace owner will be PostgreSQL .

after creating temp tablespace there is one changes must be done in PostgreSQL configuration files there parameter called temp_tablespaces where you have to specify temp tablespace name .

```
# CLIENT CONNECTION DEFAULTS
#------------------------------------------------------------------------------

# - Statement Behavior -

#client_min_messages = notice           # values in order of decreasing detail:
                                        #   debug5
                                        #   debug4
                                        #   debug3
                                        #   debug2
                                        #   debug1
                                        #   log
                                        #   notice
                                        #   warning
                                        #   error
#search_path = '"$user", public'        # schema names
#row_security = on
#default_tablespace = ''                # a tablespace name, '' uses the default
#temp_tablespaces = ''                  # a list of tablespace names, '' uses
                                        # only default tablespace
#default_table_access_method = 'heap'
#check_function_bodies = on
#default_transaction_isolation = 'read committed'
#default_transaction_read_only = off
#default_transaction_deferrable = off
#session_replication_role = 'origin'
#statement_timeout = 0                  # in milliseconds, 0 is disabled
#lock_timeout = 0                       # in milliseconds, 0 is disabled
#idle_in_transaction_session_timeout = 0        # in milliseconds, 0 is disabled
#vacuum_freeze_min_age = 50000000
#vacuum_freeze_table_age = 150000000
#vacuum_multixact_freeze_min_age = 5000000
#vacuum_multixact_freeze_table_age = 150000000
#vacuum_cleanup_index_scale_factor = 0.1        # fraction of total number of tuples
                                                # before index cleanup, 0 always performs
```

9. backup and restore

type of backup

1. logical backup

A logical backup refers to the process of converting the data in a database into a straightforward text format. This typically involves creating scripts of the database, table, or database cluster, which are then saved as SQL files.

The scripts for tables and databases are composed of SQL statements. By executing these SQL statements, the tables and databases can be recreated

2. physical backup

A physical backup entails duplicating the actual files used for the storage and retrieval of the database. These backups are typically in binary form, which is not human-readable.
Physical backups can be categorized into two types:

- **Offline Backup**: This type of backup is performed when the system is shut down. During this time, a backup of the data directory is taken.
- **Online Backup**: This is conducted while the system is operational and running, with users actively connected to it.

logical backup

taking logical backup

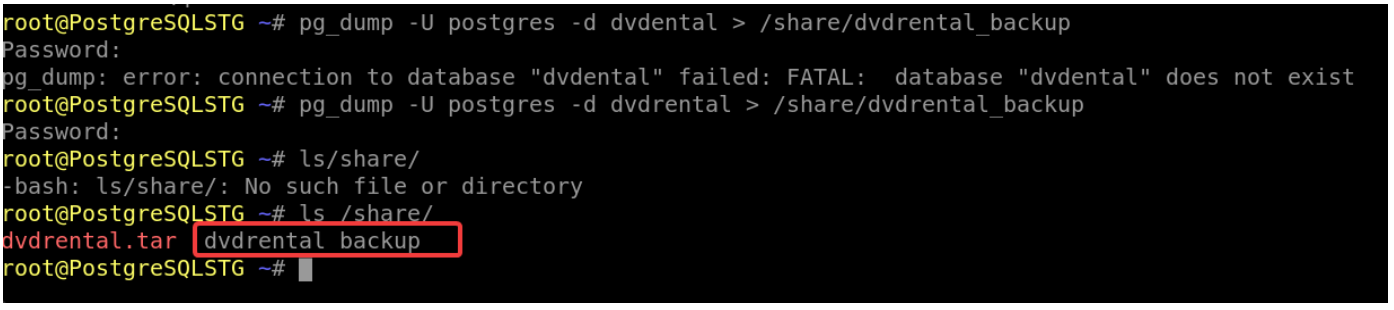
To take a logical backup in PostgreSQL, you can use the built-in utility pg_dump. The syntax for using this utility is as follows:

```
pg_dump -U [username] -d [database_name] >
[path_to_backup_file]/backup_file_name.sql
```

Replace [username] with your PostgreSQL username, [database_name] with the name of the database you want to back up, and [path_to_backup_file]/backup_file_name.sql with the path and name of the file where you want to store the backup.
For example:

```
pg_dump -U postgres -d dvdrental > /share/dvdrental_backup.sql
```

In this example, **postgres** is the username, **dvdrental** is the database name, and the backup will be stored in the specified path **/share/dvdrental_backup.sql**. Make sure that you have the necessary permissions for the folder where you intend to store the backup. If you don't specify an extension for the file, it will be automatically saved with the .sql extension, which is the standard for SQL files.



if you check the file content by using any prefers text editor such as `less` you will find SQL statement only on the file .

```
root@PostgreSQLSTG ~# less /share/dvdrental_backup
--
-- PostgreSQL database dump
--

-- Dumped from database version 13.8 (Debian 13.8-0+deb11u1)
-- Dumped by pg_dump version 13.8 (Debian 13.8-0+deb11u1)

SET statement_timeout = 0;
SET lock_timeout = 0;
SET idle_in_transaction_session_timeout = 0;
SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;
SELECT pg_catalog.set_config('search_path', '', false);
SET check_function_bodies = false;
SET xmloption = content;
SET client_min_messages = warning;
SET row_security = off;

--
-- Name: mpaa_rating; Type: TYPE; Schema: public; Owner: postgres
--

CREATE TYPE public.mpaa_rating AS ENUM (
    'G',
    'PG',
    'PG-13',
    'R',
    'NC-17'
);
```

taking backup of the entire cluster

To perform a logical backup of a PostgreSQL database cluster, you can use the built-in utility `pg_dumpall`. This utility is designed to back up all the databases in the cluster, and the user executing `pg_dumpall` needs to have full superuser privileges.

The syntax for using `pg_dumpall` is as follows:

```
pg_dumpall -U [superuser_username] >
[path_to_backup_file]/backup_file_name.sql
```

Replace [`superuser_username`] with the username of the superuser and `[path_to_backup_file]/backup_file_name.sql` with the path and name of the file where you want to store the backup.
For example:

```
pg_dumpall -U postgres > /share/dbcluster_backup.sql
```

In this example, `postgres` is the superuser, and the backup of the entire database cluster will be stored at the specified path `/share/dbcluster_backup.sql.` Remember to ensure that you have the necessary permissions for the folder where the backup file will be stored. By default, if you don't specify

an extension for the file, it will be saved with the `.sql` extension.

```
root@PostgreSQLSTG ~# pg_dumpall -U postgres > /share/clusterall_backup
Password:
Password:
Password:
Password:
Password:
Password:
Password:
Password:
Password:
Password:
root@PostgreSQLSTG ~#
```

While running **pg_dumpall** for a PostgreSQL database cluster backup, it's normal for the utility to prompt for the password multiple times. This happens because it requests the password for each database in the cluster.

This approach is suitable for smaller databases due to its straightforward nature. However, it is not recommended for large databases as it can be time-consuming to generate the backup. For larger databases, more efficient methods or tools that can handle large volumes of data more effectively might be preferable.

How to Compress and Split Dump Files

In the case of logical backups, where the dump file can become quite large, there are strategies to manage the file size and storage requirements:

1.**Compression**: You can compress the dump file to reduce its size. This is particularly useful when dealing with large databases, as it can significantly decrease the space needed for storage. Most compression tools can reduce the file size substantially, making it easier to handle and store.
syntax :

```
pg_dumpall | gzip > [filepath]/[backupfilename].gz
```

example:

```
pg_dumpall | gzip >  /share/clusterall_backup.gz
```

```
root@PostgreSQLSTG /share# pg_dumpall | gzip /share/clusterall_backup.gz
gzip: /share/clusterall_backup.gz: No such file or directory
pg_dumpall: error: pg_dump failed on database "template1", exiting
root@PostgreSQLSTG /share# pg_dumpall | gzip >  /share/clusterall_backup.gz
root@PostgreSQLSTG /share# ls
clusterall_backup  clusterall_backup.gz  dvdrental.tar  dvdrental_backup
root@PostgreSQLSTG /share#
```

```
root@PostgreSQLSTG /share# ls -ltr
total 2888
-rw------- 1 root root 2835456 Dec 21 08:41 dvdrental.tar
-rw-r--r-- 1 root root   43572 Dec 24 18:10 dvdrental_backup
-rw-r--r-- 1 root root   59960 Dec 24 18:21 clusterall_backup
-rw-r--r-- 1 root root    8550 Dec 24 18:48 clusterall_backup.gz
root@PostgreSQLSTG /share#
```

by doing compression you can see the big deterrent in size between the original dump file and compresses dump file .

2- **Splitting the File**: If you have limited space in your operating system or wish to distribute the storage of the dump across different partitions, you can split the dump file into smaller parts. This approach allows you to manage storage more effectively, especially when dealing with constraints in disk space or organizational policies on data storage.

To split the output of `pg_dumpall` into smaller files, you can indeed use the split command in Unix/Linux systems. The -b option allows you to specify the size of each split file. You can specify the size in kilobytes `(k)`, megabytes `(m)`, or gigabytes `(g).`

For example, if you want to split the dump into 1KB chunks, you would use the `1k` option. Similarly, you can use `m` for megabytes and `g` for gigabytes.

The syntax for this operation would be as follows

```
pg_dumpall  | split -b 1k - [filepath]/cluster_backup
```

```
pg_dumpall  | split -b 1k - /share/cluster_backup
```

```
root@PostgreSQLSTG /share# ls
clusterall_backup     clusterall_backupah  clusterall_backupaq  clusterall_backupaz  clusterall_backupbi  clusterall_
clusterall_backup.gz  clusterall_backupai  clusterall_backupar  clusterall_backupba  clusterall_backupbj  clusterall_
clusterall_backupaa   clusterall_backupaj  clusterall_backupas  clusterall_backupbb  clusterall_backupbk  clusterall_
clusterall_backupab   clusterall_backupak  clusterall_backupat  clusterall_backupbc  clusterall_backupbl  clusterall_
clusterall_backupac   clusterall_backupal  clusterall_backupau  clusterall_backupbd  clusterall_backupbm  clusterall_
clusterall_backupad   clusterall_backupam  clusterall_backupav  clusterall_backupbe  clusterall_backupbn  clusterall_
clusterall_backupae   clusterall_backupan  clusterall_backupaw  clusterall_backupbf  clusterall_backupbo  clusterall_
clusterall_backupaf   clusterall_backupao  clusterall_backupax  clusterall_backupbg  clusterall_backupbp  clusterall_
clusterall_backupag   clusterall_backupap  clusterall_backupay  clusterall_backupbh  clusterall_backupbq  clusterall_
root@PostgreSQLSTG /share#
```

```
-rw------- 1 root root 2835456 Dec 21 08:41 dvdrental.tar
-rw-r--r-- 1 root root   43572 Dec 24 18:10 dvdrental_backup
-rw-r--r-- 1 root root   59960 Dec 24 18:21 clusterall_backup
-rw-r--r-- 1 root root    8550 Dec 24 18:48 clusterall_backup.gz
-rw-r--r-- 1 root root    1024 Dec 24 18:56 clusterall_backupaa
-rw-r--r-- 1 root root    1024 Dec 24 18:56 clusterall_backupab
-rw-r--r-- 1 root root    1024 Dec 24 18:56 clusterall_backupaz
-rw-r--r-- 1 root root    1024 Dec 24 18:56 clusterall_backupay
-rw-r--r-- 1 root root    1024 Dec 24 18:56 clusterall_backupax
-rw-r--r-- 1 root root    1024 Dec 24 18:56 clusterall_backupaw
-rw-r--r-- 1 root root    1024 Dec 24 18:56 clusterall_backupav
-rw-r--r-- 1 root root    1024 Dec 24 18:56 clusterall_backupau
-rw-r--r-- 1 root root    1024 Dec 24 18:56 clusterall_backupat
-rw-r--r-- 1 root root    1024 Dec 24 18:56 clusterall_backupas
-rw-r--r-- 1 root root    1024 Dec 24 18:56 clusterall_backupar
-rw-r--r-- 1 root root    1024 Dec 24 18:56 clusterall_backupaq
-rw-r--r-- 1 root root    1024 Dec 24 18:56 clusterall_backupap
-rw-r--r-- 1 root root    1024 Dec 24 18:56 clusterall_backupao
-rw-r--r-- 1 root root    1024 Dec 24 18:56 clusterall_backupan
-rw-r--r-- 1 root root    1024 Dec 24 18:56 clusterall_backupam
-rw-r--r-- 1 root root    1024 Dec 24 18:56 clusterall_backupal
-rw-r--r-- 1 root root    1024 Dec 24 18:56 clusterall_backupak
-rw-r--r-- 1 root root    1024 Dec 24 18:56 clusterall_backupaj
-rw-r--r-- 1 root root    1024 Dec 24 18:56 clusterall_backupai
-rw-r--r-- 1 root root    1024 Dec 24 18:56 clusterall_backupah
-rw-r--r-- 1 root root    1024 Dec 24 18:56 clusterall_backupag
-rw-r--r-- 1 root root    1024 Dec 24 18:56 clusterall_backupaf
-rw-r--r-- 1 root root    1024 Dec 24 18:56 clusterall_backupae
```

restore logical backup using psql

to test the process i have go ahead and drop database dvdrental and then i will attempt to restore the database

```
      Name       |  Owner   | Encoding | Collate | Ctype |   Access privileges
-----------------+----------+----------+---------+-------+-----------------------
 ahmed           | root     | UTF8     | C       | C     |
 dvdrental       | postgres | UTF8     | C       | C     |
 empolyee        | postgres | UTF8     | C       | C     |
 postgres        | postgres | UTF8     | C       | C     |
 roger           | ahmed    | UTF8     | C       | C     |
 root            | postgres | UTF8     | C       | C     |
 template0       | postgres | UTF8     | C       | C     | =c/postgres          +
                 |          |          |         |       | postgres=CTc/postgres
 template1       | postgres | UTF8     | C       | C     | =c/postgres          +
                 |          |          |         |       | postgres=CTc/postgres
 testinhernttable | postgres | UTF8     | C       | C     |
 testtablespace1  | postgres | UTF8     | C       | C     |
(10 rows)

postgres=# drop database dvdrental ;
DROP DATABASE
postgres=# \l
                                 List of databases
      Name       |  Owner   | Encoding | Collate | Ctype |   Access privileges
-----------------+----------+----------+---------+-------+-----------------------
 ahmed           | root     | UTF8     | C       | C     |
 empolyee        | postgres | UTF8     | C       | C     |
 postgres        | postgres | UTF8     | C       | C     |
 roger           | ahmed    | UTF8     | C       | C     |
 root            | postgres | UTF8     | C       | C     |
 template0       | postgres | UTF8     | C       | C     | =c/postgres          +
                 |          |          |         |       | postgres=CTc/postgres
 template1       | postgres | UTF8     | C       | C     | =c/postgres          +
                 |          |          |         |       | postgres=CTc/postgres
 testinhernttable | postgres | UTF8     | C       | C     |
 testtablespace1  | postgres | UTF8     | C       | C     |
(9 rows)

postgres=#
```

we can use `psql` for restoring the database from backup

before we restore database we have to create empty database

syntax:

```
psql -U postgres -d dvdrental < /shar/dvdrental_bacup
```

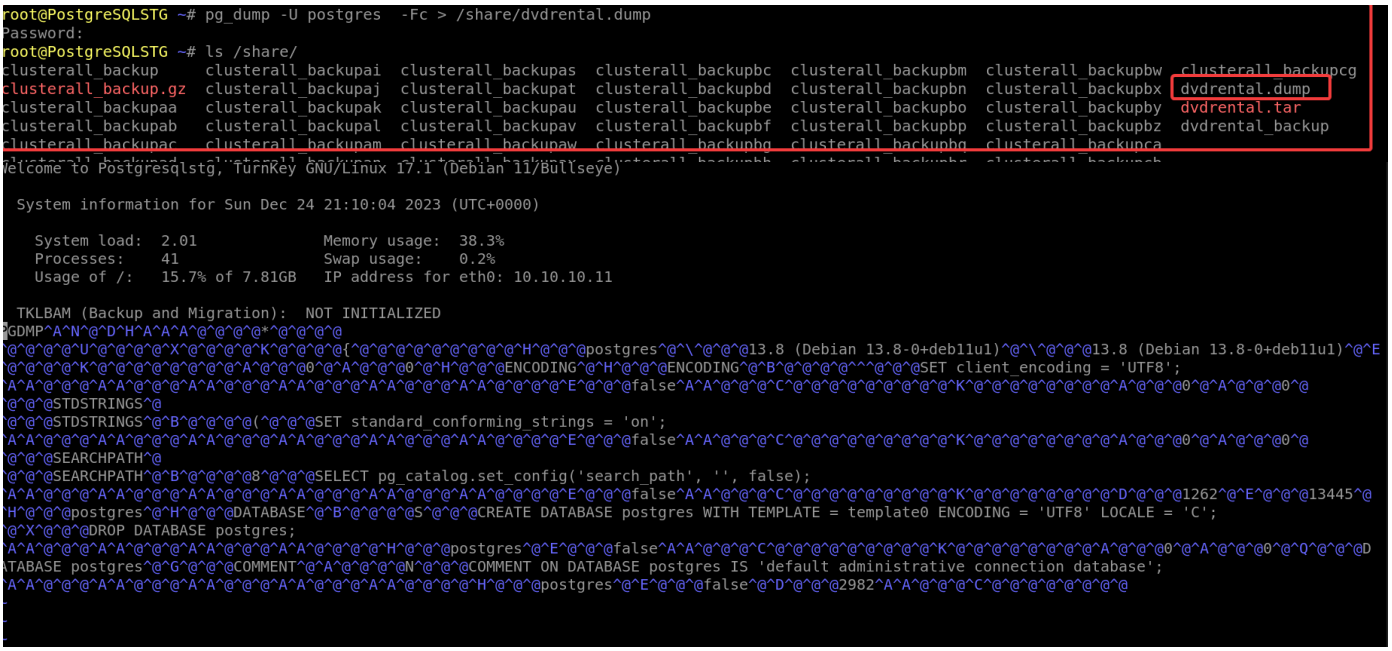
```
root@PostgreSQLSTG /share# psql -U postgres -d dvdrental < /share/dvdrental_backup
Password for user postgres:
SET
SET
SET
SET
SET
 set_config
------------

(1 row)

SET
SET
SET
SET
CREATE TYPE
ALTER TYPE
CREATE DOMAIN
ALTER DOMAIN
root@PostgreSQLSTG /share# psql -U postgres -d dvdrental
Password for user postgres:
psql (13.8 (Debian 13.8-0+deb11u1))
Type "help" for help.

dvdrental=# \d
                  List of relations
 Schema |            Name             |   Type   |  Owner
--------+-----------------------------+----------+----------
 public | actor                       | table    | postgres
 public | actor_actor_id_seq          | sequence | postgres
 public | actor_info                  | view     | postgres
 public | address                     | table    | postgres
 public | address_address_id_seq      | sequence | postgres
 public | category                    | table    | postgres
 public | category_category_id_seq    | sequence | postgres
 public | city                        | table    | postgres
 public | city_city_id_seq            | sequence | postgres
 public | country                     | table    | postgres
 public | country_country_id_seq      | sequence | postgres
 public | customer                    | table    | postgres
 public | customer_customer_id_seq    | sequence | postgres
 public | customer_list               | view     | postgres
 public | film                        | table    | postgres
 public | film_actor                  | table    | postgres
 public | film_category               | table    | postgres
 public | film_film_id_seq            | sequence | postgres
 public | film_list                   | view     | postgres
 public | inventory                   | table    | postgres
 public | inventory_inventory_id_seq  | sequence | postgres
```

restore logical backup pg_restore

pg_restore is used to restore PostgreSQL database from archive created by pg_dump in one of the non plain text format
meaning we need to create custom dump with certain format so that it can be supported by pg_restore.

to create custom dump use the below syntax

```
pg_dump -U postgres -d[database_name]dvdrental  -Fc > /share/dvdrental.dump
```



the file is in binary format not understand by the human .

restore only single table.

in this case i will drop a table and only restore this table , this one of common scenarios you will encounter during production .

```
empolyee=# \d
               List of relations
 Schema |       Name        |   Type   |  Owner
--------+-------------------+----------+----------
 public | department        | table    | postgres
 public | department_id_seq | sequence | postgres
 public | empolyees         | table    | postgres
 public | empolyees_id_seq  | sequence | postgres
(4 rows)

empolyee=# drop table department ;
DROP TABLE
empolyee=# \d
               List of relations
 Schema |       Name        |   Type   |  Owner
--------+-------------------+----------+----------
 public | empolyees         | table    | postgres
 public | empolyees_id_seq | sequence | postgres
(2 rows)

empolyee=#
```

command to restore table

syntax:

```
pg_restore -t [table-name ] department -d [database-name] empolyee -U postgres /share/dvdrental.dump
```

```
root@PostgreSQLSTG ~# pg_restore -t department -d empolyee -U postgres /share/dvdrental.dump
Password:
root@PostgreSQLSTG ~#
```

physical backup

offline backup

in here the database server must be offline in order to take backup
this type of backup is useful when we want to make changes to database directory and we want to revert back if there is any flaw in our implementation .

its important to note that when we restore the database the PostgreSQL server must be shutdown during the restore .

partial restore or singe table restore is not possible because we are backup the entire data directory and when we restore we are going to restore the entire data directory

to start i have two dbcluster
using the below command we can stop ne of them

```
pg_lscluster
pg_ctlcluster 13 ahmed stop
```

```
root@PostgreSQLSTG .../13/ahmed# pg_lsclusters
Ver Cluster Port Status Owner    Data directory              Log file
13  ahmed   5433 online postgres /postgres2/data/           /var/log/postgresql/postgresql-13-ahmed.log
13  main    5432 online postgres /var/lib/postgresql/13/main /var/log/postgresql/postgresql-13-main.log
root@PostgreSQLSTG .../13/ahmed# pg_ctlcluster 13 ahmed stop
root@PostgreSQLSTG .../13/ahmed# pg_lsclusters
Ver Cluster Port Status Owner    Data directory              Log file
13  ahmed   5433 down   postgres /postgres2/data/           /var/log/postgresql/postgresql-13-ahmed.log
13  main    5432 online postgres /var/lib/postgresql/13/main /var/log/postgresql/postgresql-13-main.log
root@PostgreSQLSTG .../13/ahmed# pg_ctlcluster 13 ahmed status
pg_ctl: no server running
root@PostgreSQLSTG .../13/ahmed# pg_ctlcluster 13 ahmed
Error: Usage: /usr/bin/pg_ctlcluster <version> <cluster> <action> [-- <pg_ctl options>]
root@PostgreSQLSTG .../13/ahmed# pg_ctlcluster 13 ahmed statsu
```

syntax for taking offline back is as follows

```
tar -cvzf  [filename]data_backup '[data directory path]'
```

the backup will happen in folder you are currently in so better to change you directory to folder you want to store the backup

```
root@PostgreSQLSTG .../13/ahmed# cd /share/backup
-bash: cd: /share/backup: No such file or directory
root@PostgreSQLSTG .../13/ahmed# mkdir  /share/backup/
root@PostgreSQLSTG .../13/ahmed# cd /share/backup/
root@PostgreSQLSTG /share/backup# ls
root@PostgreSQLSTG /share/backup# tar -cvzf ahmedcluster_backup '/postgres2/data/'
tar: Removing leading `/' from member names
/postgres2/data/
/postgres2/data/pg_xact/
/postgres2/data/pg_xact/0000
/postgres2/data/postmaster.opts
/postgres2/data/pg_serial/
/postgres2/data/pg_subtrans/
/postgres2/data/pg_subtrans/0000
/postgres2/data/pg_stat/
/postgres2/data/pg_stat/db_0.stat
/postgres2/data/pg_stat/global.stat
/postgres2/data/pg_stat/db_1.stat
/postgres2/data/pg_stat/db_13445.stat
/postgres2/data/pg_dynshmem/
/postgres2/data/pg_notify/
/postgres2/data/pg_snapshots/
/postgres2/data/postgresql.auto.conf
/postgres2/data/pg_wal/
/postgres2/data/pg_wal/000000010000000000000001
/postgres2/data/pg_wal/archive_status/
/postgres2/data/PG_VERSION
/postgres2/data/base/
/postgres2/data/base/1/
/postgres2/data/base/1/4144
/postgres2/data/base/1/3433
/postgres2/data/base/1/2603_fsm
/postgres2/data/base/1/2651
/postgres2/data/base/1/13299
/postgres2/data/base/1/2610_vm
/postgres2/data/base/1/2602_vm
root@PostgreSQLSTG /share/backup# ls
ahmedcluster_backup
root@PostgreSQLSTG /share/backup#
```

online physical backup

In this type of backup the system available and online for the user to do there
In background continuous backup is taken

In postgresql we use a continuous archiving method to enforce online backup.

Wall files : similar to MS SQL log file , here the transaction gets written on wall files upon commit before they are written to the data file .

This is done to ensure that in case of a crash we can recover the system using the wal files .

Archiver : archiver role is to copy the wall files to another safe location
To achieve online backup we will use continuous archiving of wall files or in better context continuous copying of wall files to safe locations .

why i need to copy the wall files to another location

In case of a disaster assuming i have full backup on sunday and system crash monday at 10:00AM .
And you need to recover the system till 10 Am because you don't want to lose data .
In this case you will need the full backup taken on Sunday plus all archived wal files.
This method of restore is called point in time recovery.

how to setup wall archiving

1- Enable wal archiving in the postgresql.conf files
2- We have to make base backup which is our full backup

Login to postgres and check if archive mode
Using the below command
`Show archive_mode;`
You can see the wall archiver is off still not configured

To configure wall archiving sop the cluster
`Pg_ctlcluster 13 ahmed stop`
No we need to make changes in the `postgresql.conf file`
`Vi /etc/postgresql/13/ahmed/postgresql.conf`
Look for parameter called `'wal_level` and uncomment the line

Also look for parameter`'archive_mode'`uncomment and change it from off to on

The look for archive command parameter
Before you do you need to make a directory where the wall files will be restore and ensure the correct permission are given
`Chmod 700 [directory]`
`Chown postgres:postgres [directory ]`
Now back to `archive_command` , uncomment the line in the ‘’ add the below command
`Cp -i %p [the directory you want to store the wal files ]/%f`
`Cp -i $p /share/archive/%f`

Once done start the cluster and check if archiving is enabled

By going to psql console and check the status using the below command

`Show archive_mode;`

```
postgres=# show archive_mode
postgres-# ;
 archive_mode
--------------
 on
(1 row)

postgres=#
```

test the wal level arching

We can use the which just inform postgresql that i am going to start a backup

note : this is not real backup

`Select pg_start_backup('database_name');`

```
postgres=# select pg_start_backup('major');
 pg_start_backup
-----------------
 0/2000028
(1 row)

postgres=#
```

from the restult you can find one row copyied

you can use `\! [bash command]` to run os bach command directorly from psql console .

```
postgres=# \! ls -ltr
total 16384
-rw------- 1 postgres postgres 16777216 Dec 26 21:16 000000010000000000000001
postgres=#
```

from the reslut there is one line is copyed

It's stop the command by usin the below

```
select pg_stop_backup();
```

for now we have archive wall for certain database from psql

```
postgres=# select pg_stop_backup();
NOTICE:  all required WAL segments have been archived
 pg_stop_backup
----------------
 0/2000138
(1 row)

postgres=#
```

```
root@PostgreSQLSTG /share/archive# ls -ltr
total 32772
-rw------- 1 postgres postgres 16777216 Dec 26 21:16 000000010000000000000001
-rw------- 1 postgres postgres 16777216 Dec 26 21:20 000000010000000000000002
-rw------- 1 postgres postgres      324 Dec 26 21:20 000000010000000000000002.00000028.backup
root@PostgreSQLSTG /share/archive#
```

taking full online full backup using low level api

we have setup archive backup , but you must keep in mind that archiver require full backup to be avilable
without that archiver is totaly useless.
base_backup is very

full online backup means full backup is taken while system is online

there two to take full online backup
1-low level api
2- pg_basebackup

to take low level api backup use the below syntax
in pql console
`select pg_start_backup('give lable',false , false)`
the first `false` means you are asking postgresql to take its time for doing I/O andnot return the contol to user imiddiatly .
the second `false` inform PostgreSQL to take non exeslusive backup.

```
postgres=# select pg_start_backup('backup5',false,false)
postgres-# ;
 pg_start_backup
-----------------
 0/6000028
(1 row)

postgres=#
```

now we will take `tar` o the entire data directory.
similar to offline backup but this time we will stop the cluster

```
tar -cvzf ahmed_clustr_backup.tar /postgres2/data/
```

now go back to psql console and stop the backup that we run

```
select pg_stop_backup('f');
```

we mentioned f because this non exclusive backup

online backup using base_backup

base backup is can be used for replication and point in time recovery
the base_backup don't put the database in backup mode but make the database accessible while the backup is running .
base_backup cannot take single object or single table or singe database , insist it will only take full backup of the entire DBCLUSTER .
syntax:

```
pg_basebackup -D [backup diretcory location]
```

```
root@PostgreSQLSTG /share/archive# pg_basebackup --help
pg_basebackup takes a base backup of a running PostgreSQL server.

Usage:
  pg_basebackup [OPTION]...

Options controlling the output:
  -D, --pgdata=DIRECTORY receive base backup into directory
  -F, --format=p|t       output format (plain (default), tar)
  -r, --max-rate=RATE    maximum transfer rate to transfer data directory
                         (in kB/s, or use suffix "k" or "M")
  -R, --write-recovery-conf
                         write configuration for replication
  -T, --tablespace-mapping=OLDDIR=NEWDIR
                         relocate tablespace in OLDDIR to NEWDIR
      --waldir=WALDIR    location for the write-ahead log directory
  -X, --wal-method=none|fetch|stream
                         include required WAL files with specified method
  -z, --gzip             compress tar output
  -Z, --compress=0-9     compress tar output with given compression level

General options:
  -c, --checkpoint=fast|spread
                         set fast or spread checkpointing
  -C, --create-slot      create replication slot
  -l, --label=LABEL      set backup label
```

```
pg_basebackup  -h localhost -p 5433 -U postgred -D /share/backup/ -Ft -z -P
-X
```

`-h` this host in which where the backup will be store in my case is localhost but you can store it in remote server

`-Ft` is for format you want

`-z` this will store the backup in gzip format

`-P` this will allow you to view the progress of the backup.

`-Xs` means you want backup of the entire database along with all transaction which are happening during the time of the backup

```
root@PostgreSQLSTG /share/backup# pg_basebackup  -h localhost -p 5433 -U postgres -D /share/backup/ -Ft -z -P -Xs
Password:
33031/33031 kB (100%), 1/1 tablespace
root@PostgreSQLSTG /share/backup#
```

10-Maintenance in PostgreSQL (course closure)

- explain plan and query execution cost.
- updating planner statistics / analyse
  - analyse command
  - real world examble
    - how to enable autovacum
- vacuum freeze
- vacuum
  - what is vacuum
  - what is vacuum full
    - how to check table size
  - how to vacuum database an remove data fragmentation.
    - turn off autovacuum
    - what happen when table is bloated
    - what happen when we vacuum the table
    - what happen when we vacuum full the table
- routine reindexing
  - how to locate fragmented index
    - how to install pgstattuple extension
    - how to use pgstattuple
  - rebuild index
- cluster

all database must have matininace ask to keep optimal performance

maintenance task are task performed regular
meaning we use automate these task
in Linux we use 'cronjob'
in windows we use ''task scheduler'.
PostgreSQL provide the following maintenance option.
1.updating planner statistics # analyse
2.Vacuum
3.routing reindexing
4.cluster

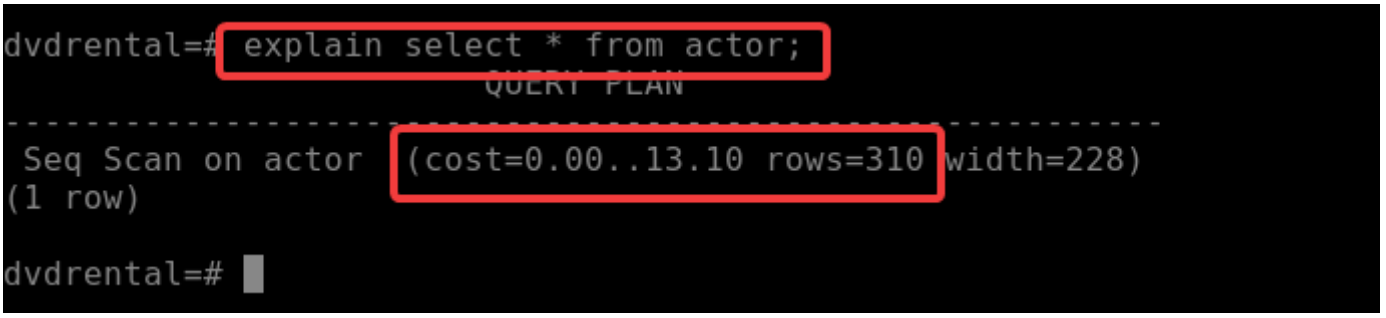
explain plan and query execution cost.

`explain` statement display execution plan chosen by the optimizer for `selec,update,insert,delete` statement .
`explain` help us understand how mush time and how mush cost it will if we execute the query , explain plan doesn't execute the query only just show execution plan .
syntax:
`explain select * from <tablename>;`

example ;
`explain select * from actor;`
the query will display the following
cost here means the cost of CPU and the number of pages read , the that i will calculated and the cost is presented
the cost equation is as following = 'umber of page read * sequnce_page_cost*row count * CPU usage time '
Cost Analysis:
the cost will display two value 1.intila cost , 2. actual cost
1.**Initial Cost:** This is the preliminary estimation of how resource-intensive the query might be.
2. **Actual Cost:** This value represents the actual resources consumed when running the query.
Row Details:

Rows Read: This indicates the number of rows that the query will read from the database.
Data Width:

Row Size: This represents the size or width of each row's data.



the explain cost will change depend on query for example the below query will have `where` clause adding flirting to the result

```
explain select * from actor where actor_id =2;
```

you can see the cost reduce because we only checking one certain row

```
dvdrental=# explain select * from actor where actor_id =2;
                              QUERY PLAN
-----------------------------------------------------------------------
 Index Scan using actor_pkey on actor  (cost=0.15..8.17 rows=1 width=228)
   Index Cond: (actor_id = 2)
(2 rows)

dvdrental=# explain select * from actor where actor_id =2;
```

updating planner statistics / analyse

- PostgreSQL query planner relays on statistical information about the contact of the table in order to generate good plans for query .
- If the statistical information about the table, including the number of rows or the number of columns, remains static and is not updated, the query "'optimizer"' may struggle to create effective execution plans. Keeping these statistics up-to-date is crucial for the optimizer to generate optimal query plans that adapt to changes in the data distribution and size.
- table in general get updated deleted new insert
  so what's is going on in the table has to be update to the "optimizer"
- inaccurate or outdate statics may lead to optimizer chosen the poor execution plan which my lead to database performance degradation
-

analyse command

- analyse command collects information about row size , row count , average row size and row sampling information
- we can run Analyze command automatcily by enable autovaccum daemon(enabled by default) or run the analyze command manually

real world examble

i have created this table

```
CREATE TABLE tel_directory (
    ID VARCHAR(5),
    Name VARCHAR(50),
    City VARCHAR(50),
    ZipCode INT
);
```

```
dvdrental=# CREATE TABLE tel_directory (
dvdrental(#     ID VARCHAR(5),
dvdrental(#     Name VARCHAR(50),
dvdrental(#     City VARCHAR(50),
dvdrental(#     ZipCode INT
dvdrental(# );
CREATE TABLE
dvdrental=#
```

then i will disable auto '"vacuum"'

the command as follow .

```
alter table [tabe-name] SEt (autovacum_enabled=false);
```

```
dvdrental=# alter table tel_directory SET (autovacuum_enabled =false);
ALTER TABLE
dvdrental=#
```

to confirm if the table not under auto vacuum

use the bellow command

```
select reloptions from pg_class where relname='tabe_name';
```

```
dvdrental=# select reloptions from pg_class where relname='tel_directory';
        reloptions
---------------------------
 {autovacuum_enabled=false}
(1 row)

dvdrental=#
```

I have upload the below insert file to server we will use psql to insert this

insert.sql

to insert we will use the below command

```
psql -U postgres -d [database-name] -f insert.sql
```

insert is completed I have run the command which count how many row are there

```
dvdrental=# select count(*) from tel_directory ;
 count
-------
  5000
(1 row)

dvdrental=#
```

now we will see what's inofmration does optimize have about the table

for insist the row are stored in pages

let's see what the optimize know about number o pages and number of row

```
select relname,reltuples,relpages from pg_class where relname ='table_name';
```

```
postgres=# select relname,reltuples,relpages from pg_class where relname ='tel_directory';
 relname | reltuples | relpages
---------+-----------+----------
(0 rows)

postgres=#
```

result cam as follow:

-reltuples : number of row , cam as 0 optimizer but there is 5000 row

-relpages : number of pages came 0 but not possible since there are rows in the table

so let's see the quey execution plan for `select * from tel_dirctory;`

```
explain select * from tel_directory;
```

```
dvdrental=# select count(*) from tel_directory;
 count
-------
  5000
(1 row)

dvdrental=# explain select * from tel_directory;
                          QUERY PLAN
---------------------------------------------------------------
 Seq Scan on tel_directory  (cost=0.00..40.64 rows=864 width=264)
(1 row)

dvdrental=#
```

the execution plan came wrong its said it will read 864 row while there is 5000 row as shown in count query .
this means that optimizer doesn't have any idea about the table.

now we will run the Analise command this now will tell the optimizer what are number row and the pages

the command is as follow

```
analyze [table-name];
```

```
dvdrental=# analyze tel_directory ;
ANALYZE
dvdrental=#
```

now we will repeat the same command
now let's see the statistics that optimizer know about the table .

```
select relname,reltuples,relpages from pg_class where relname ='table_name';
```

```
dvdrental=# select relname,reltuples,relpages from pg_class where relname ='tel_directory';
    relname    | reltuples | relpages
---------------+-----------+----------
 tel_directory |      5000 |       32
(1 row)

dvdrental=#
```

now the optimizer know that there is 5000 row and there 32 pages took to store the rows

now let's check `explain` and we will see the explain plan know the exist number of row

```
dvdrental=# explain select * from tel_directory;
                          QUERY PLAN
---------------------------------------------------------------
 Seq Scan on tel_directory  (cost=0.00..82.00 rows=5000 width=18)
(1 row)

dvdrental=#
```

how to enable autovacum

the same command todisable only we wll put true

```
alter table [tabe-name] SEt (autovacum_enabled=true);
```

to confirm if the table not under auto vacuum
use the bellow command

```
select reloptions from pg_class where relname='tabe_name';
```

```
dvdrental=# explain select * from tel_directory;
                          QUERY PLAN
---------------------------------------------------------------
 Seq Scan on tel_directory  (cost=0.00..82.00 rows=5000 width=18)
(1 row)

dvdrental=# alter table tel_directory SET (autovacuum_enabled =true);
ALTER TABLE
dvdrental=# select reloptions from pg_class where relname='tel_directory';
        reloptions
--------------------------
 {autovacuum_enabled=true}
(1 row)

dvdrental=#
```

vacuum freeze

is a critical maintenance operation in PostgreSQL. It's specifically designed to mark rows as "frozen," which means they are no longer subject to being vacuumed or removed by the system. This process is essential for ensuring the stability and performance of the database, particularly in scenarios where data changes frequently

Here's why VACUUM FREEZE is important:

- **Transaction ID Wraparound Prevention**: PostgreSQL uses Transaction IDs (XIDs) to track the age and status of transactions. Since XIDs are finite and wrap around, there's a risk of a "transaction ID wraparound" if the XID limit is reached. When this happens, it can lead to data corruption and downtime. VACUUM FREEZE helps prevent this by recycling old XIDs, making them available for reuse.
- Performance Optimization: As rows become "frozen," they are excluded from regular vacuuming processes. This reduces the overhead of vacuum operations and helps maintain consistent database performance over time.

Example:
Imagine you have a PostgreSQL database that's been running for several years, with frequent data

modifications. The XID counter has been steadily increasing. If you don't perform VACUUM FREEZE, you risk reaching the XID limit, which could result in a catastrophic database failure.
PostgreSQL uses the utility called `vacumdb` to vacum freeze tabel or row or entire database .

```
root@PostgreSQLSTG /share# vacuumdb --help
vacuumdb cleans and analyzes a PostgreSQL database.

Usage:
  vacuumdb [OPTION]... [DBNAME]

Options:
  -a, --all                       vacuum all databases
  -d, --dbname=DBNAME             database to vacuum
      --disable-page-skipping     disable all page-skipping behavior
  -e, --echo                      show the commands being sent to the server
  -f, --full                      do full vacuuming
  -F, --freeze                    freeze row transaction information
  -j, --jobs=NUM                  use this many concurrent connections to vacuum
      --min-mxid-age=MXID_AGE     minimum multixact ID age of tables to vacuum
      --min-xid-age=XID_AGE       minimum transaction ID age of tables to vacuum
  -P, --parallel=PARALLEL_WORKERS use this many background workers for vacuum, if available
  -q, --quiet                     don't write any messages
      --skip-locked               skip relations that cannot be immediately locked
  -t, --table='TABLE[(COLUMNS)]'  vacuum specific table(s) only
  -v, --verbose                   write a lot of output
  -V, --version                   output version information, then exit
  -z, --analyze                   update optimizer statistics
  -Z, --analyze-only              only update optimizer statistics; no vacuum
      --analyze-in-stages         only update optimizer statistics, in multiple
                                  stages for faster results; no vacuum
  -?, --help                      show this help, then exit

Connection options:
  -h, --host=HOSTNAME       database server host or socket directory
  -p, --port=PORT           database server port
  -U, --username=USERNAME   user name to connect as
  -w, --no-password         never prompt for password
  -W, --password            force password prompt
  --maintenance-db=DBNAME   alternate maintenance database

Read the description of the SQL command VACUUM for details.
```

vacuum

the utility for vacuum is built in called `vaccumdb`
syntax for normal vacuum
`vacuumdb`

- is also called **bloat** in PostgreSQL.
- **PostgreSQL doesn't UPDATE in PLACE OR DELETE a row directory form the disk .**
- meaning when you delete a record form table they are not actually delete they are just marked as deleted and they are keep as old version
- as the old version keep pilling up and they become absolute , this causes fragmentation and bloating in the table .
- so the tables are indexes will look the excite the same size even after you delete a lot of record . **to solve this we will use vacuum and vacuum full**

what is vacuum

vacuum reclaim storage occupied by dead tuples/rows

there are two type of vacuum :
-**vacuum**:the utility simply reclaim the space and make it available for reuse , this is allow to free space on table level but space will not be usable by OS , however you can do more insert on the table.
vacuum allow to reuse space in table but not to because by OS .

- during vacuum there will not be exclusive luck on table , meaning we can do vacuum during business hour and it will not effect performance .

frequently updated table are good candidate for vacuuming.

what is vacuum full

- vacuum full rewrite the entire contents of the tables into new disk file with no extra space .
- vacuum full release the space to OS occupied by delete records .
- it reduce the size of table
- vacuum full take a lot of time then vacuum ,because it going to rewrite the entire content of the table into the disk again
- vacuum full will place exclusive lock .
- its reamended not to run full vacuum during bussnies hour
  syntax for vacuumdb is
  `vacuumdb -f`

i have deleted the follow amout of row .

```
dvdrental=# delete from tel_directory where city = 'NY';
DELETE 467
dvdrental=#
```

let see the tablespace size

```
SELECT pg_size_pretty(pg_total_relation_size('your_table_name'));
or\dt [table-name]
```

```
dvdrental=# SELECT pg_size_pretty(pg_total_relation_size('tel_directory'));
 pg_size_pretty
----------------
 288 kB
(1 row)

dvdrental=#
```

```
root@PostgreSQLSTG /share# vacuumdb --help
vacuumdb cleans and analyzes a PostgreSQL database.

Usage:
  vacuumdb [OPTION]... [DBNAME]

Options:
  -a, --all                       vacuum all databases
  -d, --dbname=DBNAME             database to vacuum
      --disable-page-skipping     disable all page-skipping behavior
  -e, --echo                      show the commands being sent to the server
  -f, --full                      do full vacuuming
  -F, --freeze                    freeze row transaction information
  -j, --jobs=NUM                  use this many concurrent connections to vacuum
      --min-mxid-age=MXID_AGE     minimum multixact ID age of tables to vacuum
      --min-xid-age=XID_AGE       minimum transaction ID age of tables to vacuum
  -P, --parallel=PARALLEL_WORKERS use this many background workers for vacuum, if available
  -q, --quiet                     don't write any messages
      --skip-locked               skip relations that cannot be immediately locked
  -t, --table='TABLE[(COLUMNS)]'  vacuum specific table(s) only
  -v, --verbose                   write a lot of output
  -V, --version                   output version information, then exit
  -z, --analyze                   update optimizer statistics
  -Z, --analyze-only              only update optimizer statistics; no vacuum
      --analyze-in-stages         only update optimizer statistics, in multiple
                                  stages for faster results; no vacuum
  -?, --help                      show this help, then exit

Connection options:
  -h, --host=HOSTNAME       database server host or socket directory
  -p, --port=PORT           database server port
  -U, --username=USERNAME   user name to connect as
  -w, --no-password         never prompt for password
  -W, --password            force password prompt
  --maintenance-db=DBNAME   alternate maintenance database

Read the description of the SQL command VACUUM for details.

Report bugs to <pgsql-bugs@lists.postgresql.org>.
PostgreSQL home page: <https://www.postgresql.org/>
root@PostgreSQLSTG /share# vacuumdb -f -d dvdrental -t tel_directory -U postgres
Password:
vacuumdb: vacuuming database "dvdrental"
```

you can see the size reduced

```
dvdrental=# SELECT pg_size_pretty(pg_total_relation_size('tel_directory'))
 pg_size_pretty
----------------
 232 kB
(1 row)

dvdrental=#
```

how to check table size

it important to able able to determine the size of the table

the command below will show the size

```
/dt+ [table-name]
```

```
dvdrental=# \dt tel_directory
            List of relations
 Schema |     Name      | Type  |  Owner
--------+---------------+-------+----------
 public | tel_directory | table | postgres
(1 row)

dvdrental=# \dt+ tel_directory
                            List of relations
 Schema |     Name      | Type  |  Owner   | Persistence |  Size  | Description
--------+---------------+-------+----------+-------------+--------+-------------
 public | tel_directory | table | postgres | permanent   | 264 kB |
(1 row)

dvdrental=#
```

\dT+ [pattern]
Lists all data types or only those that match pattern. The command form \dT+ shows extra information.

how to vacuum database an remove data fragmentation.

we have create table called tel_directory with 9000 row this current size of the table

```
\dt+ tel_directory
```

```
dvdrental=# \dt+ tel_directory
                            List of relations
 Schema |     Name      | Type  |  Owner   | Persistence |  Size  | Description
--------+---------------+-------+----------+-------------+--------+-------------
 public | tel_directory | table | postgres | permanent   | 520 kB |
(1 row)

dvdrental=#
```

and this the count of rows in the table

```
vdrental=# select count(*) from tel_directory
vdrental-# ;
count
------
 9533
1 row)
```

turn off autovacuum

if vaccum is enabled by default we will not be able to test our size deferent when we do vacuum of full vacuum when we delete row

to turn off autovacuum use the below command

```
alter table tel_directory set (autovacuum_enabled=false);
```

```
dvdrental=# alter table tel_directory SET (autovacuum_enabled =false);
ALTER TABLE
dvdrental=#
```

what happen when table is bloated

now back to our example we will delete record from table

```
delete from tel_directory where state in('TX','NJ','NY');
```

```
dvdrental=# delete from tel_directory  where city  in ('CA','FL','NJ');
DELETE 3002
dvdrental=#
```

now 3002 record has been delete let's check the row count and size

```
dvdrental=# delete from tel_directory  where city  in ('CA','FL','NJ');
DELETE 3002
dvdrental=# select count(*) from tel_directory
;
 count
-------
  6531
(1 row)

dvdrental=# \dt+ tel_directory
                            List of relations
 Schema |     Name      | Type  |  Owner   | Persistence |  Size  | Description
--------+---------------+-------+----------+-------------+--------+-------------
 public | tel_directory | table | postgres | permanent   | 520 kB |
(1 row)

dvdrental=#
```

the size didn't change but it increased , because the record are not removed form table they are marked as old version , meaning they still occupy space from disk

now i will insert the record again
i have file called state.sql that contain all record we deleted we will insert it back to the table

state.sql

```
psql -U postgres -d dvdrental -f state.sql
```

let's check the size and row count

```
postgres=# \c dvdrental
You are now connected to database "dvdrental" as user "postgres".
dvdrental=# select count(*) from tel_directory
;
 count
-------
  7826
(1 row)

dvdrental=# \dt+ tel_directory
                            List of relations
 Schema |     Name      | Type  |  Owner   | Persistence |  Size  | Description
--------+---------------+-------+----------+-------------+--------+-------------
 public | tel_directory | table | postgres | permanent   | 584 kB |
(1 row)

dvdrental=#
```

the size of the table increased but this now consider bloated ("data fragmented") table containing record that are there and record that are delete but not removed only marked as old .
the "bloating" where size of the table increased even though it holding the same amount of records

what happen when we vacuum the table

now I will drop this table and start again and create table and insert the record from insert.sql file
truncate will remove all record in the table

```
truncate table tel_directory;
select * from tel_directory;
```

```
dvdrental=# truncate table tel_directory ;
TRUNCATE TABLE
dvdrental=#
dvdrental=# select count(*) from tel_directo
;
 count
-------
     0
(1 row)

dvdrental=#
```

now i will run the insert.sql again

```
dvdrental=# \q
root@PostgreSQLSTG /share# psql -U postgres -d dvdrental -f insert.sql
Password for user postgres:
postgres=# \c dvdrental
You are now connected to database "dvdrental" as user "postgres".
dvdrental=# select count(*) from tel_directory
;
 count
-------
  5000
(1 row)

dvdrental=# \dt+ tel_directory
                                List of relations
 Schema |     Name      | Type  |  Owner   | Persistence |  Size  | Description
--------+---------------+-------+----------+-------------+--------+-------------
 public | tel_directory | table | postgres | permanent   | 280 kB |
(1 row)

dvdrental=#
```

i will delete the row same as before and check the count and the size after delete

```
delete from tel_directory  where city  in ('CA','FL','NJ');
```

```
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
root@PostgreSQLSTG /share# psql -U postgres
Password for user postgres:
psql (13.13 (Debian 13.13-0+deb11u1))
Type "help" for help.

postgres=# \c dvdrental
You are now connected to database "dvdrental" as user "postgres".
dvdrental=# select count(*) from tel_directory
;
 count
-------
  5000
(1 row)

dvdrental=# \dt+ tel_directory
                                List of relations
 Schema |     Name      | Type  |  Owner   | Persistence |  Size  | Description
--------+---------------+-------+----------+-------------+--------+-------------
 public | tel_directory | table | postgres | permanent   | 280 kB |
(1 row)

dvdrental=# delete from tel_directory where state in('TX','NJ','NY');
ERROR:  column "state" does not exist
LINE 1: delete from tel_directory where state in('TX','NJ','NY');
                                        ^
dvdrental=# delete from tel_directory  where city  in ('CA','FL','NJ');
DELETE 1501
dvdrental=# select count(*) from tel_directory
;
 count
-------
  3499
(1 row)

dvdrental=# \dt+ tel_directory
                                List of relations
 Schema |     Name      | Type  |  Owner   | Persistence |  Size  | Description
--------+---------------+-------+----------+-------------+--------+-------------
 public | tel_directory | table | postgres | permanent   | 280 kB |
(1 row)

dvdrental=#
```

now we will do normal vaccum and after that check the status , below is done in psql console

```
# vacuum(verbose) tel_directory
```

the option `verbose` will show what has been and status after the command , its optional not needed you can use the below syntax if you would like

```
vacum tel_directory
```

```
postgres=# \c dvdrental
You are now connected to database "dvdrental" as user "postgres".
dvdrental=# vacuum(verbose) tel_directory ;
INFO:  vacuuming "public.tel_directory"
INFO:  "tel_directory": removed 26 row versions in 32 pages
INFO:  "tel_directory": found 26 removable, 3499 nonremovable row versions in 32 out of 32 pages
DETAIL:  0 dead row versions cannot be removed yet, oldest xmin: 17698
There were 0 unused item identifiers.
Skipped 0 pages due to buffer pins, 0 frozen pages.
0 pages are entirely empty.
CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s.
VACUUM
dvdrental=#
dvdrental=# \dt+ tel_directory
                              List of relations
 Schema |     Name      | Type  |  Owner   | Persistence |  Size  | Description
--------+---------------+-------+----------+-------------+--------+------------
 public | tel_directory | table | postgres | permanent   | 288 kB |
(1 row)

dvdrental=#
```

the size has increased why ???
reason is that vacuum has deleted the rows but has reserved some space for the insert .

now we will insert the row in state.sql and check if the space changed

```
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
root@PostgreSQLSTG /share# psql -U postgres
Password for user postgres:
psql (13.13 (Debian 13.13-0+deb11u1))
Type "help" for help.

postgres=# \c dvdrental
You are now connected to database "dvdrental" as user "postgres".
dvdrental=# \dt+ tel_directory
                              List of relations
 Schema |     Name      | Type  |  Owner   | Persistence |  Size  | Description
--------+---------------+-------+----------+-------------+--------+------------
 public | tel_directory | table | postgres | permanent   | 288 kB |
(1 row)

dvdrental=#
```

FILE TRANSFERS Clear
state.sql 81.8 KB

the space has not changed .

what happen when we vacuum full the table

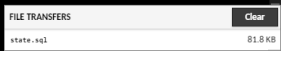
same senario i will truncate the table then reinsert the row and then check the space

after that i will delete the row and use vacuum full instied of normal vacuum and check the space .

```
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
root@PostgreSQLSTG /share# psql -U postgres
Password for user postgres:
psql (13.13 (Debian 13.13-0+deb11u1))
Type "help" for help.

postgres=# \c dvdrental
You are now connected to database "dvdrental" as user "postgres".
dvdrental=# \dt+ tel_directory
                          List of relations
 Schema |     Name      | Type  |  Owner   | Persistence |  Size  | Description
--------+---------------+-------+----------+-------------+--------+-------------
 public | tel_directory | table | postgres | permanent   | 280 kB |
(1 row)

dvdrental=# select count(*) from tel_directory
;
 count
-------
  5000
(1 row)

dvdrental=#
```

now I will delete the row and check the space

```
INSERT 0 1
INSERT 0 1
root@PostgreSQLSTG /share# psql -U postgres
Password for user postgres:
psql (13.13 (Debian 13.13-0+deb11u1))
Type "help" for help.

postgres=# \c dvdrental
You are now connected to database "dvdrental" as user "postgres".
dvdrental=# \dt+ tel_directory
                          List of relations
 Schema |     Name      | Type  |  Owner   | Persistence |  Size  | Description
--------+---------------+-------+----------+-------------+--------+-------------
 public | tel_directory | table | postgres | permanent   | 280 kB |
(1 row)

dvdrental=# select count(*) from tel_directory
;
 count
-------
  5000
(1 row)

dvdrental=# delete from tel_directory  where city  in ('CA','FL','NJ');
DELETE 1501
dvdrental=# select count(*) from tel_directory
;
 count
-------
  3499
(1 row)

dvdrental=# select count(*) from tel_directory
;
 count
-------
  3499
(1 row)

dvdrental=# \dt+ tel_directory
                          List of relations
 Schema |     Name      | Type  |  Owner   | Persistence |  Size  | Description
--------+---------------+-------+----------+-------------+--------+-------------
 public | tel_directory | table | postgres | permanent   | 280 kB |
(1 row)

dvdrental=#
```

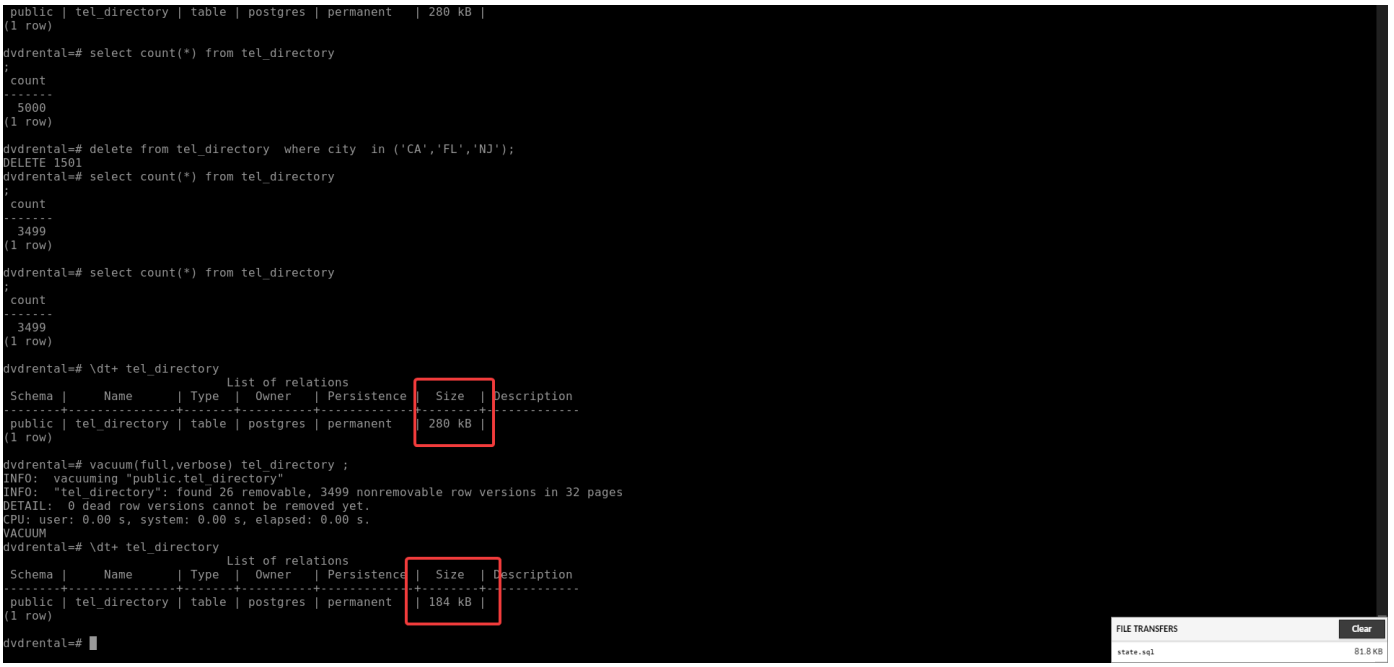


now we will do vacuum full in the psql

```
vacuum(full,verbose) tel_directory;
```

```
dvdrental=# vacuum(full,verbose) tel_directory ;
INFO:  vacuuming "public.tel_directory"
INFO:  "tel_directory": found 26 removable, 3499 nonremovable row versions in 32 pages
DETAIL:  0 dead row versions cannot be removed yet.
CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s.
VACUUM
dvdrental=#
```

the size as shown below has decreased with huge margin compare to normal vacuum



vacuum full has gone ahead and deleted all dead rows and released the space to the OS

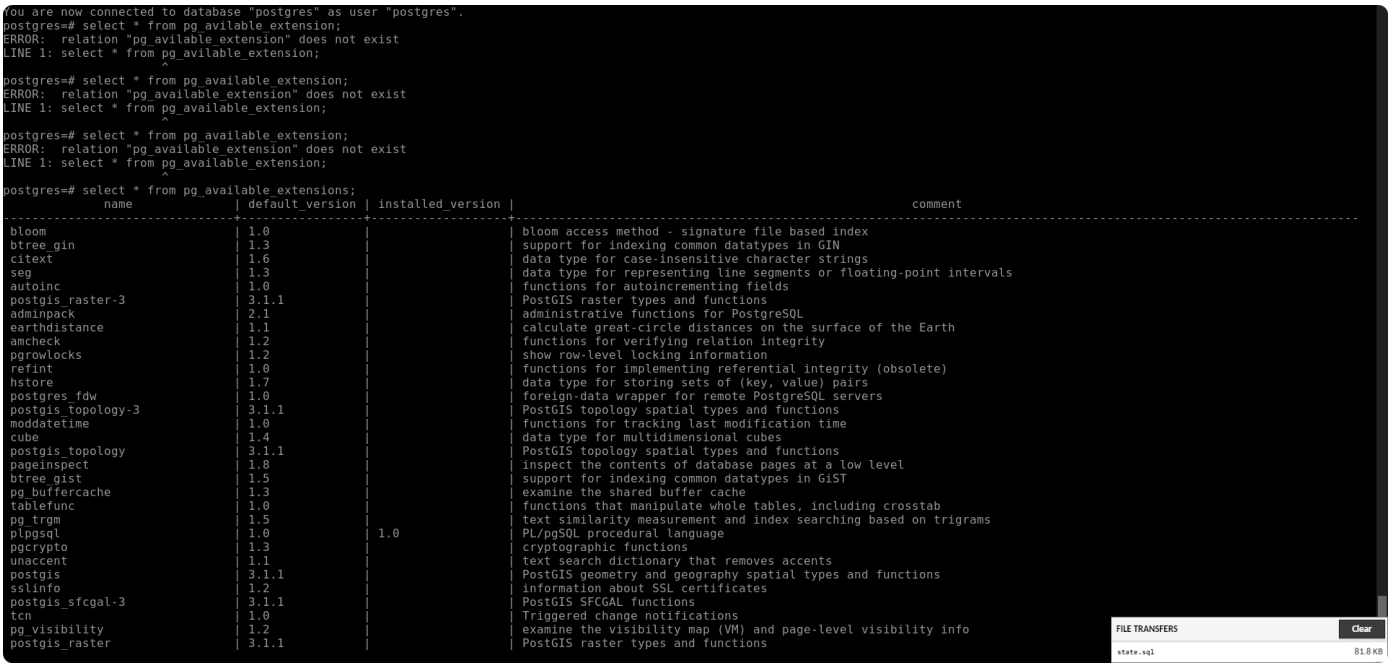
routine reindexing

- insert and update and delete overtime fragments the index
- a fragmented index will have pages where logica order based on key value differs from the physical ordering inside the data file .
- heavily fragmented index can degrade query performance because additional I/O is required to locate data to which the index point
- reindex rebuild as index using data stored in index table and eliminates empty space between pages
- syntax : `reindex index <index-name>`.

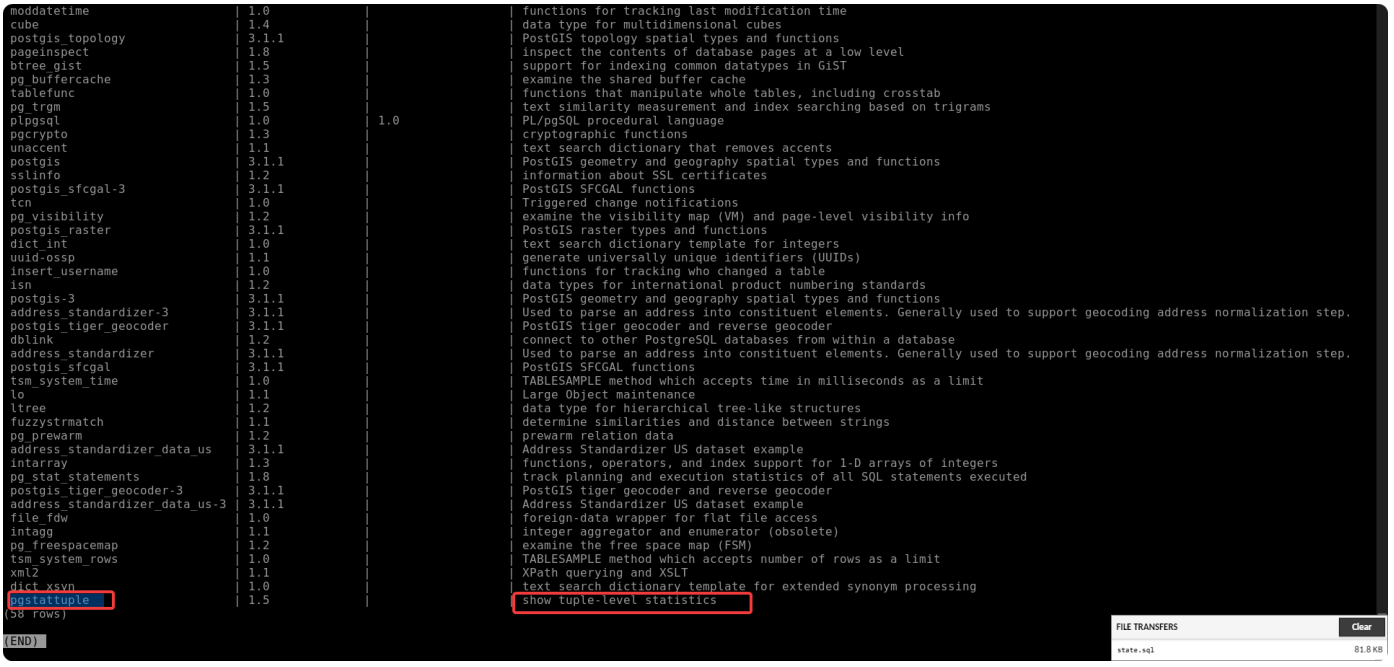
how to locate fragmented index

there internal veiw in PostgreSQL that will show statics about index , the view will show the extension that you have to install in order to add more feature , wear looking for extension that will show table statics

```
select * from pg_available_extensions;
```



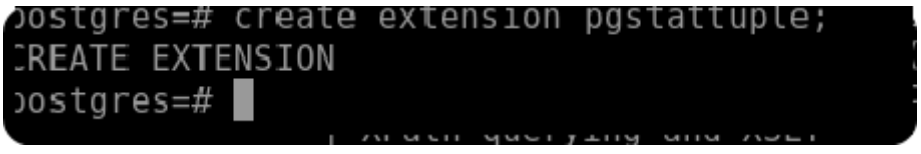
the extension we are looking for is called `pgstattuple`



how to install pgstattuple extension

to install the extension we will go to psql console and type the below command

```
create extension pgstattuple
```



run the below command again to confirm the extension is installed

```
select * from pg_available_extensions;
```

```
 intagg                        | 1.1             |                   | integer aggregator and enumerator (obsolete)
 pg_freespacemap               | 1.2             |                   | examine the free space map (FSM)
 tsm_system_rows               | 1.0             |                   | TABLESAMPLE method which accepts number of rows as a limit
 xml2                          | 1.1             |                   | XPath querying and XSLT
 dict_xsyn                     | 1.0             |                   | text search dictionary template for extended synonym processing
 pgstattuple                   | 1.5             |                   | show tuple-level statistics
(58 rows)

postgres=# create extension pgstattuple;
CREATE EXTENSION
postgres=# select * from pg_available_extensions;
             name              | default_version | installed_version |                                                  comment
-------------------------------+-----------------+-------------------+-----------------------------------------------------------------------------------------------------------
 bloom                         | 1.0             |                   | bloom access method - signature file based index
 btree_gin                     | 1.3             |                   | support for indexing common datatypes in GIN
 citext                        | 1.6             |                   | data type for case-insensitive character strings
 seg                           | 1.3             |                   | data type for representing line segments or floating-point intervals
 autoinc                       | 1.0             |                   | functions for autoincrementing fields
 postgis_raster-3              | 3.1.1           |                   | PostGIS raster types and functions
 adminpack                     | 2.1             |                   | administrative functions for PostgreSQL
 earthdistance                 | 1.1             |                   | calculate great-circle distances on the surface of the Earth
 amcheck                       | 1.2             |                   | functions for verifying relation integrity
 pgrowlocks                    | 1.2             |                   | show row-level locking information
 refint                        | 1.0             |                   | functions for implementing referential integrity (obsolete)
 hstore                        | 1.7             |                   | data type for storing sets of (key, value) pairs
 postgres_fdw                  | 1.0             |                   | foreign-data wrapper for remote PostgreSQL servers
 postgis_topology-3            | 3.1.1           |                   | PostGIS topology spatial types and functions
 moddatetime                   | 1.0             |                   | functions for tracking last modification time
 cube                          | 1.4             |                   | data type for multidimensional cubes
 postgis_topology              | 3.1.1           |                   | PostGIS topology spatial types and functions
 pageinspect                   | 1.8             |                   | inspect the contents of database pages at a low level
 btree_gist                    | 1.5             |                   | support for indexing common datatypes in GiST
 pg_buffercache                | 1.3             |                   | examine the shared buffer cache
 tablefunc                     | 1.0             |                   | functions that manipulate whole tables, including crosstab
 pg_trgm                       | 1.5             |                   | text similarity measurement and index searching based on trigrams
 plpgsql                       | 1.0             | 1.0               | PL/pgSQL procedural language
 pgcrypto                      | 1.3             |                   | cryptographic functions
 unaccent                      | 1.1             |                   | text search dictionary that removes accents
 postgis                       | 3.1.1           |                   | PostGIS geometry and geography spatial types and functions
 sslinfo                       | 1.2             |                   | information about SSL certificates
 postgis_sfcgal-3              | 3.1.1           |                   | PostGIS SFCGAL functions
 tcn                           | 1.0             |                   | Triggered change notifications
 pg_visibility                 | 1.2             |                   | examine the visibility map (VM) and page-level visibility info
 postgis_raster                | 3.1.1           |                   | PostGIS raster types and functions
 dict_int                      | 1.0             |                   | text search dictionary template for integers
 uuid-ossp                     | 1.1             |                   | generate universally unique identifiers (UUIDs)
 insert_username               | 1.0             |                   | functions for tracking who changed a table
```

how to use pgstattuple

to use the extension first check how many index you have in the database.

```
\di
```

```
 dict_xsyn                     | 1.0             |                   | text search dictionary template for extended synonym processing
 pgstattuple                   | 1.5             | 1.5               | show tuple-level statistics
(58 rows)

postgres=# \di
Did not find any relations.
postgres=# \c dvdrental
You are now connected to database "dvdrental" as user "postgres".
dvdrental=# \di
                                 List of relations
 Schema |                      Name                       | Type  |  Owner   |     Table
--------+-------------------------------------------------+-------+----------+---------------
 public | actor_pkey                                      | index | postgres | actor
 public | address_pkey                                    | index | postgres | address
 public | category_pkey                                   | index | postgres | category
 public | city_pkey                                       | index | postgres | city
 public | country_pkey                                    | index | postgres | country
 public | customer_pkey                                   | index | postgres | customer
 public | film_actor_pkey                                 | index | postgres | film_actor
 public | film_category_pkey                              | index | postgres | film_category
 public | film_fulltext_idx                               | index | postgres | film
 public | film_pkey                                       | index | postgres | film
 public | idx_actor_last_name                             | index | postgres | actor
 public | idx_fk_address_id                               | index | postgres | customer
 public | idx_fk_city_id                                  | index | postgres | address
 public | idx_fk_country_id                               | index | postgres | city
 public | idx_fk_customer_id                              | index | postgres | payment
 public | idx_fk_film_id                                  | index | postgres | film_actor
 public | idx_fk_inventory_id                             | index | postgres | rental
 public | idx_fk_language_id                              | index | postgres | film
 public | idx_fk_rental_id                                | index | postgres | payment
 public | idx_fk_staff_id                                 | index | postgres | payment
 public | idx_fk_store_id                                 | index | postgres | customer
 public | idx_last_name                                   | index | postgres | customer
 public | idx_store_id_film_id                            | index | postgres | inventory
 public | idx_title                                       | index | postgres | film
 public | idx_unq_manager_staff_id                        | index | postgres | store
 public | idx_unq_rental_rental_date_inventory_id_customer_id | index | postgres | rental
 public | inventory_pkey                                  | index | postgres | inventory
 public | language_pkey                                   | index | postgres | language
 public | payment_pkey                                    | index | postgres | payment
 public | rental_pkey                                     | index | postgres | rental
 public | staff_pkey                                      | index | postgres | staff
 public | store_pkey                                      | index | postgres | store
(32 rows)

dvdrental=#
```

now i will see what is the status of one of the index , whether this index is bloated or its fragmented . or its working fine

```
select * from pgstatindex('[index-name]');
```

note : the extension must be installed on database you want to check the index .

```
 public | idx_fk_inventory_id                             | index | postgres | rental
 public | idx_fk_language_id                             | index | postgres | film
 public | idx_fk_rental_id                               | index | postgres | payment
 public | idx_fk_staff_id                                | index | postgres | payment
 public | idx_fk_store_id                                | index | postgres | customer
 public | idx_last_name                                  | index | postgres | customer
 public | idx_store_id_film_id                           | index | postgres | inventory
 public | idx_title                                      | index | postgres | film
 public | idx_unq_manager_staff_id                       | index | postgres | store
 public | idx_unq_rental_rental_date_inventory_id_customer_id | index | postgres | rental
 public | inventory_pkey                                 | index | postgres | inventory
 public | language_pkey                                  | index | postgres | language
 public | payment_pkey                                   | index | postgres | payment
 public | rental_pkey                                    | index | postgres | rental
 public | staff_pkey                                     | index | postgres | staff
 public | store_pkey                                     | index | postgres | store
 public | test                                           | index | postgres | tel_directory
(33 rows)

dvdrental=# select * from pgstatindex('test');
ERROR:  function pgstatindex(unknown) does not exist
LINE 1: select * from pgstatindex('test');
                      ^
HINT:  No function matches the given name and argument types. You might need to add explicit type casts.
dvdrental=# SELECT * FROM pgstattuple ('tel_directory');
ERROR:  function pgstattuple(unknown) does not exist
LINE 1: SELECT * FROM pgstattuple ('tel_directory');
                      ^
HINT:  No function matches the given name and argument types. You might need to add explicit type casts.
dvdrental=# create extension pgstattuple
dvdrental-# ;
CREATE EXTENSION
dvdrental=# select * from pgstatindex('test');
ERROR:  relation "test" is not a btree index
dvdrental=# select * from pgstatindex('actor_pkey');
 version | tree_level | index_size | root_block_no | internal_pages | leaf_pages | empty_pages | deleted_pages | avg_leaf_density | leaf_fragmentation
---------+------------+------------+---------------+----------------+------------+-------------+---------------+------------------+--------------------
       4 |          0 |       8192 |             0 |              0 |          0 |           0 |             0 |              NaN |                NaN
(1 row)

dvdrental=# select * from pgstatindex('actor_pkey');
 version | tree_level | index_size | root_block_no | internal_pages | leaf_pages | empty_pages | deleted_pages | avg_leaf_density | leaf_fragmentation
---------+------------+------------+---------------+----------------+------------+-------------+---------------+------------------+--------------------
       4 |          0 |       8192 |             0 |              0 |          0 |           0 |             0 |              NaN |                NaN
(1 row)

dvdrental=#
```

the most important thing here is **leaf _fragmentation .**
if the leaf fragmentation is about 50 this is considered as fragmented index

rebuild index

syntax :

`reindex index [index-name]`

```
dvdrental=# reindex index actor_pkey;
REINDEX
```

cluster

- `cluster` instruct PostgreSQL to cluster the table base on the index specified by the index_name
- when table is cluster , it physically reorder base on the index information
- cluster will organize the data physically according to a particular index in the data file .
- we need to cluster because the user will not insert a data in a specific order.
- cluster is one-time operation , when table receive consent changes , the changes will not be clustered so you will have to to do cluster again
- an exclusive lock is required , so cluster should be executed during peak hours .
- cluster lower the disk access and speed up query .
- syntax for cluster `cluster [table-name] using [index_name];`

now to demonstrate the cluster i will create table and i will insert row in random order .

```
create table ahmed (id numeric , name varchar(10));
```

i have inserted values but not in order even after we do select we will check the record are not in order

```
dvdrental=# create table ahmed (id numeric , name varchar(1
CREATE TABLE
dvdrental=# insert into ahmed values (2,'B');
INSERT 0 1
dvdrental=# insert into ahmed values (1,'A');
INSERT 0 1
dvdrental=# insert into ahmed values (3,'A');
INSERT 0 1
dvdrental=# insert into ahmed values (5,'C');
INSERT 0 1
dvdrental=# insert into ahmed values (4,'D');
INSERT 0 1
dvdrental=# select * from ahmed ;
 id | name
----+------
  2 | B
  1 | A
  3 | A
  5 | C
  4 | D
(5 rows)

dvdrental=#
```

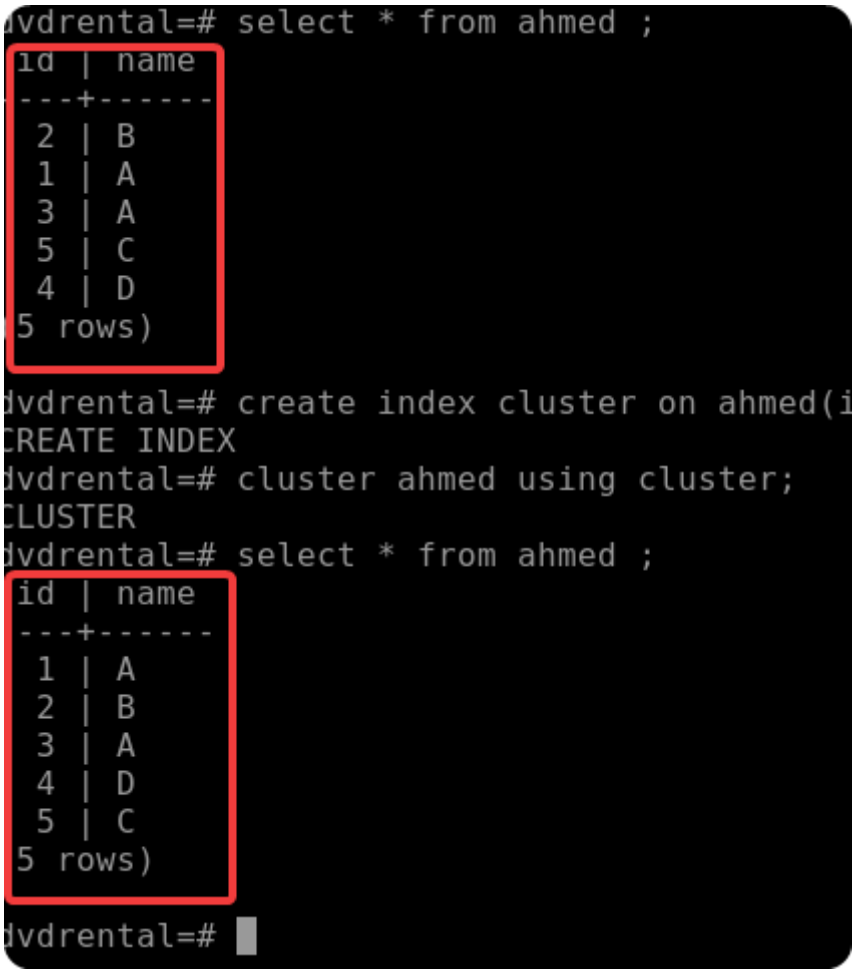
to cluster it i have to create index arrange base on id

```
create index cluster on ahmed(id);
```

```
dvdrental=# create index cluster on ahmed(id);
CREATE INDEX
dvdrental=#
```

now I will cluster the table using the index i have created , after that we will do simple query and check if the record are in order after cluster .

```
cluster ahmed using cluster;

select * from ahmed;
```



now the rows are sorted base on index on data file

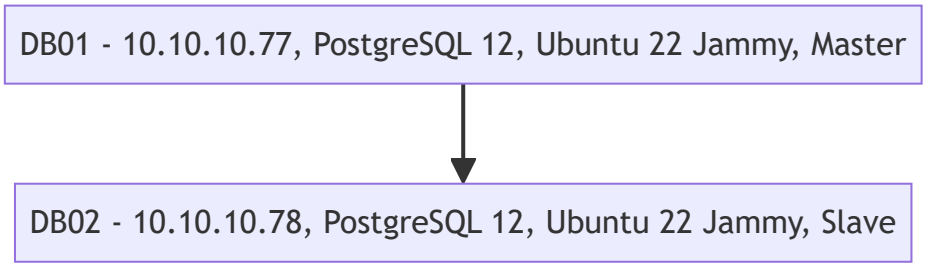
11. master-slave replication

prerequisite :

1. Ensure internet connectivity is established for the installation of PostgreSQL.
2. Nano or vim is required for editing config files
3. PostgreSQL, version 12, is to be installed on both nodes.
4. Administrative privileges (sudo access) are required for modifying configuration files.
5. Temporarily disable the firewall to facilitate the necessary configurations.
6. Root access is essential for instances where deletion of slave data files is necessary.

master slave hardware specs

we will be implement master-slave streaming replication using two VMS running ubuntu 22 jammy
both node will have PostgreSQL 12 installed , and configured with ip in same subnet
below are VM details .
1.postgresqlDB01 : 10.10.10.77
2.postgresqlDB02 : 10.10.10.78



installing PostgreSQL

first check if PostgreSQL 12 is available in Ubuntu repositories using the below command
`apt-cache madison postgresql-12`

```
ahmed@postgresqlDB01:~$ apt-cache madison postgresql-12
N: Unable to locate package postgresql-12
ahmed@postgresqlDB01:~$
```

If PostgreSQL 12 isn't available in our current repository, the next step involves adding the PostgreSQL 12 repository to our Ubuntu machine. This requires importing the GPG key and then adding the repository. To accomplish this, we'll use the following command.
Note: Internet access on the VM is a prerequisite for these steps

`wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add -`

```
ahmed@postgresqlDB01:~$ wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add -
[sudo] password for ahmed:
Warning: apt-key is deprecated. Manage keyring files in trusted.gpg.d instead (see apt-key(8)).
OK
ahmed@postgresqlDB01:~$
```

next update the ubuntu repository to add links for PostgreSQL 12

echo "deb http://apt.postgresql.org/pub/repos/apt/ `lsb_release -cs`-pgdg main" |sudo tee /etc/apt/sources.list.d/pgdg.list

```
ahmed@postgresqlDB01:~$ echo "deb http://apt.postgresql.org/pub/repos/apt/ `lsb_release -cs`-pgdg main" |sudo tee  /etc/apt/sources.list.d/pgdg.list
deb http://apt.postgresql.org/pub/repos/apt/ jammy-pgdg main
ahmed@postgresqlDB01:~$
```

We've successfully updated the repository. Now, we'll proceed with the installation of PostgreSQL 12. The following command will be used for this purpose

`sudo apt update sudo apt -y install postgresql-12 postgresql-client-12`

```
Setting up postgresql-common (256.pgdg22.04+1) ...
Adding user postgres to group ssl-cert

Creating config file /etc/postgresql-common/createcluster.conf with new version
Building PostgreSQL dictionaries from installed myspell/hunspell packages...
  en_us
Removing obsolete dictionary files:
'/etc/apt/trusted.gpg.d/apt.postgresql.org.gpg' -> '/usr/share/postgresql-common/pgdg/apt.postgresql.org.gpg'
Created symlink /etc/systemd/system/multi-user.target.wants/postgresql.service → /lib/systemd/system/postgresql.service.
Setting up postgresql-12 (12.17-1.pgdg22.04+1) ...
Creating new PostgreSQL cluster 12/main ...
/usr/lib/postgresql/12/bin/initdb -D /var/lib/postgresql/12/main --auth-local peer --auth-host md5
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.

The database cluster will be initialized with locales
  COLLATE:  en_US.UTF-8
  CTYPE:    en_US.UTF-8
  MESSAGES: en_US.UTF-8
  MONETARY: ar_SA.UTF-8
  NUMERIC:  ar_SA.UTF-8
  TIME:     ar_SA.UTF-8
The default database encoding has accordingly been set to "UTF8".
The default text search configuration will be set to "english".

Data page checksums are disabled.

fixing permissions on existing directory /var/lib/postgresql/12/main ... ok
creating subdirectories ... ok
selecting dynamic shared memory implementation ... posix
selecting default max_connections ... 100
selecting default shared_buffers ... 128MB
selecting default time zone ... Asia/Riyadh
creating configuration files ... ok
running bootstrap script ... ok
performing post-bootstrap initialization ... ok
syncing data to disk ... ok

Success. You can now start the database server using:

    pg_ctlcluster 12 main start

Processing triggers for man-db (2.10.2-1) ...
Processing triggers for libc-bin (2.35-0ubuntu3) ...
ahmed@postgresqlDB01:~$
```

```
ahmed@postgresqlDB01:~$ sudo apt update
Get:1 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Hit:2 http://sa.archive.ubuntu.com/ubuntu jammy InRelease
Get:3 http://sa.archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Hit:4 http://sa.archive.ubuntu.com/ubuntu jammy-backports InRelease
Get:5 http://sa.archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [1212 kB]
Get:6 http://apt.postgresql.org/pub/repos/apt jammy-pgdg InRelease [123 kB]
Get:7 http://sa.archive.ubuntu.com/ubuntu jammy-updates/main i386 Packages [537 kB]
Get:8 http://sa.archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages [1010 kB]
Get:9 http://sa.archive.ubuntu.com/ubuntu jammy-updates/universe i386 Packages [671 kB]
Get:10 http://apt.postgresql.org/pub/repos/apt jammy-pgdg/main amd64 Packages [296 kB]
Fetched 4077 kB in 3s (1516 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
469 packages can be upgraded. Run 'apt list --upgradable' to see them.
W: http://apt.postgresql.org/pub/repos/apt/dists/jammy-pgdg/InRelease: Key is stored in legacy trusted.gpg keyring (/etc/apt/trusted.gpg), see the DEPRECATION section in apt-key(8)
for details.
N: Skipping acquire of configured file 'main/binary-i386/Packages' as repository 'http://apt.postgresql.org/pub/repos/apt jammy-pgdg InRelease' doesn't support architecture 'i386'
ahmed@postgresqlDB01:~$ sudo apt -y install postgresql-12 postgresql-client-12
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following package was automatically installed and is no longer required:
  systemd-hwe-hwdb
Use 'sudo apt autoremove' to remove it.
The following additional packages will be installed:
  libcommon-sense-perl libjson-perl libjson-xs-perl libllvm15 libpq5 libtypes-serialiser-perl postgresql-client-common postgresql-common sysstat
Suggested packages:
  postgresql-doc-12 isag
The following NEW packages will be installed:
  libcommon-sense-perl libjson-perl libjson-xs-perl libllvm15 libpq5 libtypes-serialiser-perl postgresql-12 postgresql-client-12 postgresql-client-common postgresql-common sysstat
0 upgraded, 11 newly installed, 0 to remove and 469 not upgraded.
Need to get 43.5 MB of archives.
After this operation, 177 MB of additional disk space will be used.
Get:1 http://apt.postgresql.org/pub/repos/apt jammy-pgdg/main amd64 postgresql-client-common all 256.pgdg22.04+1 [94.0 kB]
Get:2 http://sa.archive.ubuntu.com/ubuntu jammy/main amd64 libjson-perl all 4.04000-1 [81.8 kB]
Get:3 http://sa.archive.ubuntu.com/ubuntu jammy/main amd64 libcommon-sense-perl amd64 3.75-2build1 [21.1 kB]
Get:4 http://apt.postgresql.org/pub/repos/apt jammy-pgdg/main amd64 postgresql-common all 256.pgdg22.04+1 [238 kB]
Get:5 http://sa.archive.ubuntu.com/ubuntu jammy/main amd64 libtypes-serialiser-perl all 1.01-1 [11.6 kB]
Get:6 http://sa.archive.ubuntu.com/ubuntu jammy/main amd64 libjson-xs-perl amd64 4.030-1build3 [87.2 kB]
Get:7 http://sa.archive.ubuntu.com/ubuntu jammy-updates/main amd64 libllvm15 amd64 1:15.0.7-0ubuntu0.22.04.3 [25.4 MB]
Get:8 http://apt.postgresql.org/pub/repos/apt jammy-pgdg/main amd64 libpq5 amd64 16.1-1.pgdg22.04+1 [213 kB]
Get:9 http://apt.postgresql.org/pub/repos/apt jammy-pgdg/main amd64 postgresql-client-12 amd64 12.17-1.pgdg22.04+1 [1438 kB]
Get:10 http://apt.postgresql.org/pub/repos/apt jammy-pgdg/main amd64 postgresql-12 amd64 12.17-1.pgdg22.04+1 [15.5 MB]
Get:11 http://sa.archive.ubuntu.com/ubuntu jammy-updates/main amd64 sysstat amd64 12.5.2-2ubuntu0.2 [487 kB]
Fetched 43.5 MB in 5s (8719 kB/s)
```

once the installation is completed , next steps is to verify if the PostgreSQL services are running.

`systemctl status postgresql`

```
ahmed@postgresqlDB01:~$ systemctl status postgresql
● postgresql.service - PostgreSQL RDBMS
     Loaded: loaded (/lib/systemd/system/postgresql.service; enabled; vendor preset: enabled)
     Active: active (exited) since Fri 2023-12-01 16:34:16 +03; 2min 7s ago
   Main PID: 5751 (code=exited, status=0/SUCCESS)
        CPU: 2ms

16:34:16 01 ديسمبر postgresqlDB01 systemd[1]: Starting PostgreSQL RDBMS...
16:34:16 01 ديسمبر postgresqlDB01 systemd[1]: Finished PostgreSQL RDBMS.
ahmed@postgresqlDB01:~$
```

Configuring Master

Now we will create a database user “replication” who will carry out the task of replication.
switch to postgres user that will be automatically added to ubuntu once we installed PostgreSQL

`su - postgres`

if you are unable to switch to postgres user then reset its password using the below command

`sudo passwd postgres`

```
ahmed@postgresqlDB01:~$ su - postgres
Password:
su: Authentication failure
ahmed@postgresqlDB01:~$ sudo passwd postgres
New password:
Retype new password:
passwd: password updated successfully
ahmed@postgresqlDB01:~$
```

```
ahmed@postgresqlDB01:~$ su - postgres
Password:
postgres@postgresqlDB01:~$
```

then enter PostgreSQL t-sql configuration suing `psql`

```
ahmed@postgresqlDB01:~$ su - postgres
Password:
postgres@postgresqlDB01:~$ psql
psql (12.17 (Ubuntu 12.17-1.pgdg22.04+1))
Type "help" for help.

postgres=#
```

create a database user “replication” who will carry out the task of replication using the below command , also note down the password and username you will need it in the slave configuration

`CREATE USER replication REPLICATION LOGIN CONNECTION LIMIT 1 ENCRYPTED PASSWORD '123456789';`

```
postgres=# CREATE USER replication REPLICATION LOGIN CONNECTION LIMIT 1 ENCRYPTED PASSWORD '123456789';
CREATE ROLE
postgres=#
```

check if the role is created successfully by using the below command

`\du`

```
postgres=# \du
                                   List of roles
  Role name  |                         Attributes                         | Member of
-------------+------------------------------------------------------------+-----------
 postgres    | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
 replication | Replication                                               +| {}
             | 1 connection                                               |

postgres=#
```

Next, we'll adjust the maximum number of connections permitted for the replication user. This is done by executing the following command within the psql client.

`ALTER ROLE replication CONNECTION LIMIT -1;`

```
postgres=# ALTER ROLE replication CONNECTION LIMIT -1;
ALTER ROLE
postgres=#
```

The configuration files for PostgreSQL are typically found in the following directory:

/etc/postgresql/12/main

We need to modify a file named postgresql.conf to configure this server as the master. While both nano and vim are suitable editors for this task, I will be using nano for this purpose

Note : Make sure to use a user account with sudo privileges or the root account for these steps, as administrative access is required to edit the postgresql.conf file.

`sudo nano /etc/postgresql/12/main/postgresql.conf`

```
\ahmed@postgresqlDB01:~$ sudo nano/etc/postgresql/12/main/postgresql.conf
[sudo] password for ahmed:
```

Edit the following parameter in the postgresql.conf file. If the line is currently commented out, uncomment it to activate the option

you can use Ctrl+w to search and go to desire line

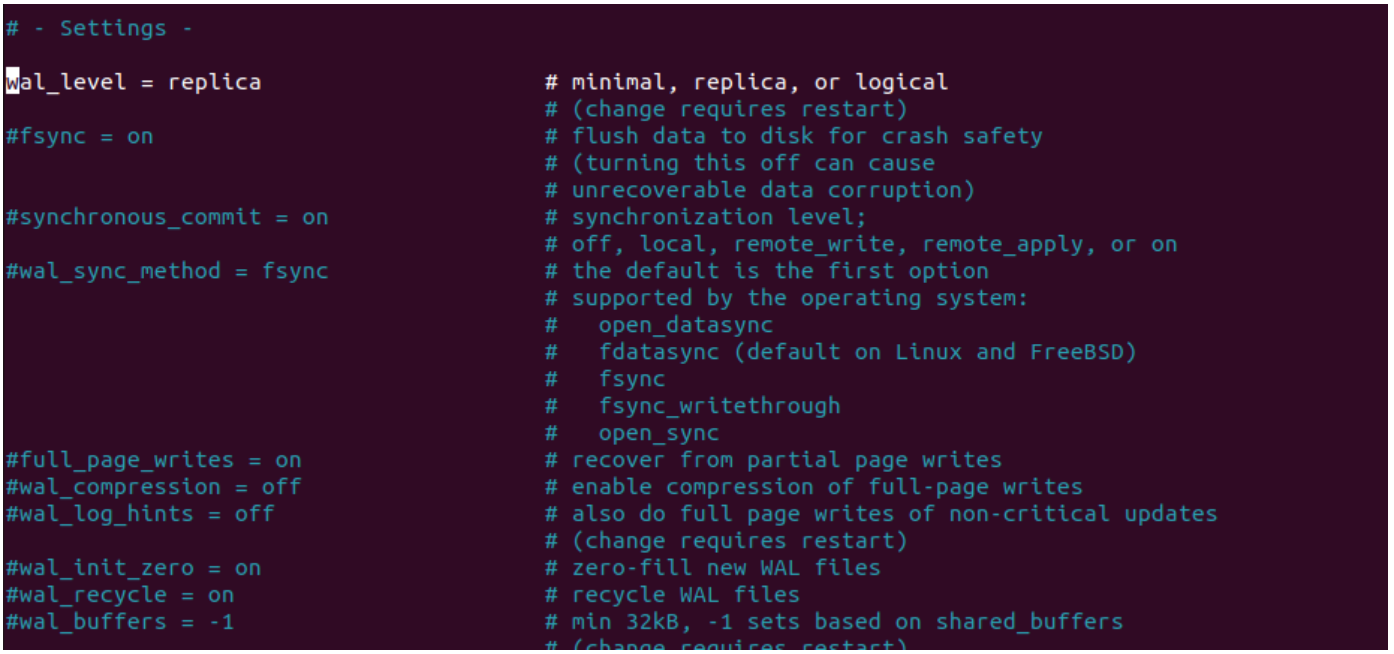
listen_addresses = '*'

wal_level = replica

max_wal_senders = 10

wal_keep_segments = 64

```
listen_addresses = '*'          # what IP address(es) to listen on;
                                        # comma-separated list of addresses;
                                        # defaults to 'localhost'; use '*' for all
                                        # (change requires restart)
port = 5432                             # (change requires restart)
max_connections = 100                   # (change requires restart)
#superuser_reserved_connections = 3     # (change requires restart)
unix_socket_directories = '/var/run/postgresql' # comma-separated list of directories
                                        # (change requires restart)
#unix_socket_group = ''                 # (change requires restart)
#unix_socket_permissions = 0777         # begin with 0 to use octal notation
                                        # (change requires restart)
#bonjour = off                          # advertise server via Bonjour
                                        # (change requires restart)
#bonjour_name = ''                      # defaults to the computer name
                                        # (change requires restart)

# - TCP settings -
# see "man 7 tcp" for details

#wal_level = replica                    # minimal, replica, or logical
                                        # (change requires restart)
#fsync = on                             # flush data to disk for crash safety
                                        # (turning this off can cause
                                        # unrecoverable data corruption)
#synchronous_commit = on                # synchronization level;
                                        # off, local, remote_write, remote_apply, or on
#wal_sync_method = fsync                # the default is the first option
                                        # supported by the operating system:
                                        #   open_datasync
                                        #   fdatasync (default on Linux and FreeBSD)
                                        #   fsync
                                        #   fsync_writethrough
                                        #   open_sync
#full_page_writes = on                  # recover from partial page writes
#wal_compression = off                  # enable compression of full-page writes
#wal_log_hints = off                    # also do full page writes of non-critical updates
                                        # (change requires restart)
#wal_init_zero = on                     # zero-fill new WAL files
#wal_recycle = on                       # recycle WAL files
#wal_buffers = -1                       # min 32kB, -1 sets based on shared_buffers
                                        # (change requires restart)
```



```
# - Settings -

wal_level = replica                     # minimal, replica, or logical
                                        # (change requires restart)
#fsync = on                             # flush data to disk for crash safety
                                        # (turning this off can cause
                                        # unrecoverable data corruption)
#synchronous_commit = on                # synchronization level;
                                        # off, local, remote_write, remote_apply, or on
#wal_sync_method = fsync                # the default is the first option
                                        # supported by the operating system:
                                        #   open_datasync
                                        #   fdatasync (default on Linux and FreeBSD)
                                        #   fsync
                                        #   fsync_writethrough
                                        #   open_sync
#full_page_writes = on                  # recover from partial page writes
#wal_compression = off                  # enable compression of full-page writes
#wal_log_hints = off                    # also do full page writes of non-critical updates
                                        # (change requires restart)
#wal_init_zero = on                     # zero-fill new WAL files
#wal_recycle = on                       # recycle WAL files
#wal_buffers = -1                       # min 32kB, -1 sets based on shared_buffers
                                        # (change requires restart)
```

once done click Ctrl+x then y , then hit enter

now Slave server need authentication for replication. Now append following line to `/etc/postgresql/12/main/pg_hba.conf` file

`sudo nano /etc/postgresql/12/main/pg_hba.conf`

`# Replace 10.10.10.78 with slave server's private IP host replication replication 10.10.10.78/24 md5`

```
#
# Database administrative login by Unix domain socket
local   all             postgres                                peer

# TYPE  DATABASE        USER            ADDRESS                 METHOD

# "local" is for Unix domain socket connections only
local   all             all                                     peer
# IPv4 local connections:
host    all             all             127.0.0.1/32            md5
# IPv6 local connections:
host    all             all             ::1/128                 md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
local   replication     all                                     peer
host    replication     all             127.0.0.1/32            md5
host    replication     all             ::1/128                 md5


# DO NOT DISABLE!
# If you change this first entry you will need to make sure that the
# database superuser can access the database using some other method.
# Noninteractive access to all databases is required during automatic
# maintenance (custom daily cronjobs, replication, and similar tasks).
#
# Database administrative login by Unix domain socket
local   all             postgres                                peer

# TYPE  DATABASE        USER            ADDRESS                 METHOD

# "local" is for Unix domain socket connections only
local   all             all                                     peer
# IPv4 local connections:
host    all             all             127.0.0.1/32            md5
# IPv6 local connections:
host    all             all             ::1/128                 md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
local   replication     all                                     peer
host    replication     all             127.0.0.1/32            md5
host    replication     all             ::1/128                 md5
host    replication     replication     10.10.10.78/24          md5
```

next steps is to restart postgresql services

`systemctl restart postgresql systemctl status postgresql`

```
ahmed@postgresqlDB01:~$ systemctl restart postgresql
ahmed@postgresqlDB01:~$ systemctl status postgresql
● postgresql.service - PostgreSQL RDBMS
     Loaded: loaded (/lib/systemd/system/postgresql.service; enabled; vendor preset: enabled)
     Active: active (exited) since Fri 2023-12-01 17:15:00 +03; 8s ago
    Process: 3137 ExecStart=/bin/true (code=exited, status=0/SUCCESS)
   Main PID: 3137 (code=exited, status=0/SUCCESS)
        CPU: 2ms

17:15:00 01 ديسمبر postgresqlDB01 systemd[1]: Starting PostgreSQL RDBMS...
17:15:00 01 ديسمبر postgresqlDB01 systemd[1]: Finished PostgreSQL RDBMS.
ahmed@postgresqlDB01:~$ ^C
ahmed@postgresqlDB01:~$
```

We have finished all configuration in Master Server and our master server is ready for replication

Configuring Slave

for the slave its mandatory to first stop PostgreSQL services

`systemctl stop postgresql systemctl status postgresql`

```
ahmed@postgresqlDB01:~$ systemctl stop postgresql
ahmed@postgresqlDB01:~$ systemctl status postgresql
○ postgresql.service - PostgreSQL RDBMS
     Loaded: loaded (/lib/systemd/system/postgresql.service; enabled; vendor preset: enabled)
     Active: inactive (dead) since Fri 2023-12-01 17:17:44 +03; 9s ago
    Process: 1091 ExecStart=/bin/true (code=exited, status=0/SUCCESS)
   Main PID: 1091 (code=exited, status=0/SUCCESS)
        CPU: 2ms

16:47:09 01 ديسمبر postgresqlDB01 systemd[1]: Starting PostgreSQL RDBMS...
16:47:09 01 ديسمبر postgresqlDB01 systemd[1]: Finished PostgreSQL RDBMS.
17:17:44 01 ديسمبر postgresqlDB02 systemd[1]: postgresql.service: Deactivated successfully.
17:17:44 01 ديسمبر postgresqlDB02 systemd[1]: Stopped PostgreSQL RDBMS.
ahmed@postgresqlDB01:~$ s
```

next eddit the config file with the below parameter

`sudo nano /etc/postgresql/12/main/postgresql.conf`

`listen_addresses = 'localhost,158.245.203.119' wal_level = replica max_wal_senders = 10 wal_keep_segments = 64 hot_standby = on`

```
# These settings are ignored on a master server.

#primary_conninfo = ''                  # connection string to sending server
                                        # (change requires restart)
#primary_slot_name = ''                 # replication slot on sending server
                                        # (change requires restart)
#promote_trigger_file = ''              # file name whose presence ends recovery
hot_standby = on                        # "off" disallows queries during recovery
                                        # (change requires restart)
#max_standby_archive_delay = 30s        # max delay before canceling queries
                                        # when reading WAL from archive;
                                        # -1 allows indefinite delay
#max_standby_streaming_delay = 30s      # max delay before canceling queries
                                        # when reading streaming WAL;
                                        # -1 allows indefinite delay
#wal_receiver_status_interval = 10s     # send replies at least this often
                                        # 0 disables
#hot_standby_feedback = off             # send info from standby to prevent
                                        # query conflicts
#wal_receiver_timeout = 60s             # time that receiver waits for
                                        # communication from master
                                        # in milliseconds; 0 disables
#wal_retrieve_retry_interval = 5s       # time to wait before retrying to
                                        # retrieve WAL after a failed attempt
#recovery_min_apply_delay = 0           # minimum delay for applying changes during recovery

# - Subscribers -
```

Now append following line to `/etc/postgresql/12/main/pg_hba.conf` file

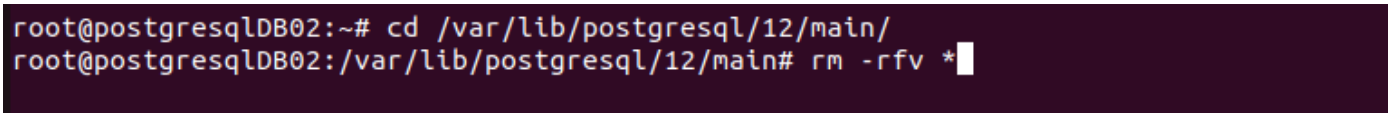
`sudo nano /etc/postgresql/12/main/pg_hba.conf`

```
# Replace 10.10.10.77 with slave master's private IP
host    replication     replication     10.10.10.77/24    md5
```

```
# "local" is for Unix domain socket connections only
local   all             all                                     peer
# IPv4 local connections:
host    all             all             127.0.0.1/32            md5
# IPv6 local connections:
host    all             all             ::1/128                 md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
local   replication     all                                     peer
host    replication     all             127.0.0.1/32            md5
host    replication     all             ::1/128                 md5
host    replication     replication     10.10.10.77/24          md5
```

For the next step, we need to remove the slave data directory. This task requires root privileges, so ensure you switch to the root user before proceeding

```
cd /var/lib/postgresql/12/main/ sudo rm -rfv *
```



```
removed 'global/1261_vm'
removed 'global/6114'
removed 'global/1213_vm'
removed 'global/6100'
removed 'global/1232'
removed 'global/4186'
removed 'global/2965'
removed 'global/4060'
removed 'global/1136'
removed 'global/2967'
removed 'global/pg_control'
removed 'global/2676'
removed directory 'global'
removed directory 'pg_commit_ts'
removed directory 'pg_dynshmem'
removed directory 'pg_logical/mappings'
removed directory 'pg_logical/snapshots'
removed 'pg_logical/replorigin_checkpoint'
removed directory 'pg_logical'
removed 'pg_multixact/members/0000'
removed directory 'pg_multixact/members'
removed 'pg_multixact/offsets/0000'
removed directory 'pg_multixact/offsets'
removed directory 'pg_multixact'
removed 'pg_notify/0000'
removed directory 'pg_notify'
removed directory 'pg_replslot'
removed directory 'pg_serial'
removed directory 'pg_snapshots'
removed 'pg_stat/global.stat'
removed 'pg_stat/db_1.stat'
removed 'pg_stat/db_0.stat'
removed directory 'pg_stat'
removed directory 'pg_stat_tmp'
removed 'pg_subtrans/0000'
removed directory 'pg_subtrans'
removed directory 'pg_tblspc'
removed directory 'pg_twophase'
removed 'PG_VERSION'
removed 'pg_wal/000000010000000000000001'
removed directory 'pg_wal/archive_status'
removed directory 'pg_wal'
removed 'pg_xact/0000'
removed directory 'pg_xact'
removed 'postgresql.auto.conf'
removed 'postmaster.opts'
root@postgresqlDB02:/var/lib/postgresql/12/main#
root@postgresqlDB02:/var/lib/postgresql/12/main# ls -l
total 0
root@postgresqlDB02:/var/lib/postgresql/12/main#
```

Now, we'll synchronize the slave database with the master database by executing the following command. This will transfer all the data from the master to the slave

`sudo su postgres cd /var/lib/postgresql/12/main/ pg_basebackup -h 10.10.10.77 -U replication -p 5432 -D /var/lib/postgresql/12/main/ -Fp -Xs -P -R`

```
root@postgresqlDB02:/var/lib/postgresql/12/main# su - postgres
postgres@postgresqlDB02:~$ cd /var/lib/postgresql/12/main/
postgres@postgresqlDB02:~/12/main$ pg_basebackup -h 10.10.10.77 -U replication -p 5432 -D /var/lib/postgresql/12/main/  -Fp -Xs -P -R
```

As mentioned earlier, remember the password set for the replication user we created. In the following step, you will be prompted to enter the password for this replication use

```
root@postgresqlDB02:/var/lib/postgresql/12/main# su - postgres
postgres@postgresqlDB02:~$ cd /var/lib/postgresql/12/main/
postgres@postgresqlDB02:~/12/main$ pg_basebackup -h 10.10.10.77 -U replication -p 5432 -D /var/lib/postgresql/12/main/  -Fp -Xs -P -R
Password:
Password:
24650/24650 kB (100%), 1/1 tablespace
postgres@postgresqlDB02:~/12/main$
```

Once the fetching process is complete, proceed to start the PostgreSQL service

`systemct start postgresql`

```
postgres@postgresqlDB02:~/12/main$ systemctl sta
start   status
postgres@postgresqlDB02:~/12/main$ systemctl start postgresql
postgres@postgresqlDB02:~/12/main$ systemctl status  postgresql
● postgresql.service - PostgreSQL RDBMS
     Loaded: loaded (/lib/systemd/system/postgresql.service; enabled; vendor preset: enabled)
     Active: active (exited) since Fri 2023-12-01 17:32:58 +03; 8s ago
    Process: 3081 ExecStart=/bin/true (code=exited, status=0/SUCCESS)
   Main PID: 3081 (code=exited, status=0/SUCCESS)
        CPU: 3ms
postgres@postgresqlDB02:~/12/main$
```

Congratulations, you have successfully replicated your database! You can verify this by making any change in the master database and observing that it gets immediately replicated in the slave database

test master slave configuration

first will create database called Test DB;
then I will create table, insert random values then we will check the slave if the data is replication
To proceed with your plan:

1. **Create Database `TestDB`**:
   - First, create a database named `TestDB`. Use the following SQL command:

     ```
     CREATE DATABASE TestDB;
     ```

2. **Create a Table and Insert Random Values**:
   - After creating `TestDB`, switch to this database:

     ```
     \c TestDB
     ```

   - Then, create a table. Let's say you create a table named `example_table`:

```
CREATE TABLE example_table (
    id SERIAL PRIMARY KEY,
    data VARCHAR(100)
);
```

- Next, insert some random values into `example_table`:

```
INSERT INTO example_table (data) VALUES ('RandomValue1');
INSERT INTO example_table (data) VALUES ('RandomValue2');
INSERT INTO example_table (data) VALUES ('RandomValue3');
```

3. **Check Replication on the Slave**:

- Finally, on the slave server, check if the data has been replicated. You can do this by querying the same table on the slave server:

```
SELECT * FROM example_table;
```

```
postgres=# CREATE DATABASE TestDB;
ERROR:  database "testdb" already exists
postgres=# \l
                                 List of databases
   Name    |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-----------+----------+----------+-------------+-------------+-----------------------
 postgres  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
 testdb    | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
(4 rows)

postgres=# \cu testdb
invalid command \cu
Try \? for help.
postgres=# \c testdb
You are now connected to database "testdb" as user "postgres".
testdb=# CREATE TABLE city (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100),
    population INT
);
CREATE TABLE
testdb=# INSERT INTO city (name, population) VALUES ('CityName1', 500000);
INSERT 0 1
testdb=# INSERT INTO city (name, population) VALUES ('CityName1', 500000);
INSERT 0 1
testdb=# INSERT INTO city (name, population) VALUES ('CityName3', 300000);
INSERT 0 1
testdb=#
```

```
testdb=# select * from city;
 id |    name     | population
----+-------------+------------
  1 | CityName1 |     500000
  2 | CityName1 |     500000
  3 | CityName3 |     300000
(3 rows)

testdb=#
```

If the slave is properly replicating data from the master, you should see the same rows that you inserted on the master.

```
postgres=# \l
                                 List of databases
   Name    |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-----------+----------+----------+-------------+-------------+-----------------------
 postgres  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
 testdb    | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
(4 rows)

postgres=#
```

```
postgres=# \c testdb
You are now connected to database "testdb" as user "postgres".
testdb=# select * from city;
 id |    name     | population
----+-------------+------------
  1 | CityName1 |     500000
  2 | CityName1 |     500000
  3 | CityName3 |     300000
(3 rows)

testdb=#
```

monitoring replication

We can verify the replication status by using the following command. If the state displays 'streaming', it indicates that everything is functioning correctly

```
SELECT * FROM pg_stat_replication;
```

```
_port |         backend_start         | backend_xmin |   state   | sent_lsn  | write_lsn | flush_lsn | replay_lsn | write_lag | flush_lag | replay_lag | sync_priority | sync_state
------+-------------------------------+--------------+-----------+-----------+-----------+-----------+------------+-----------+-----------+------------+---------------+------------
45634 | 2023-12-01 17:32:56.069546+03 |              | streaming | 0/301B428 | 0/301B428 | 0/301B428 | 0/301B428  |           |           |            |             0 | async
```

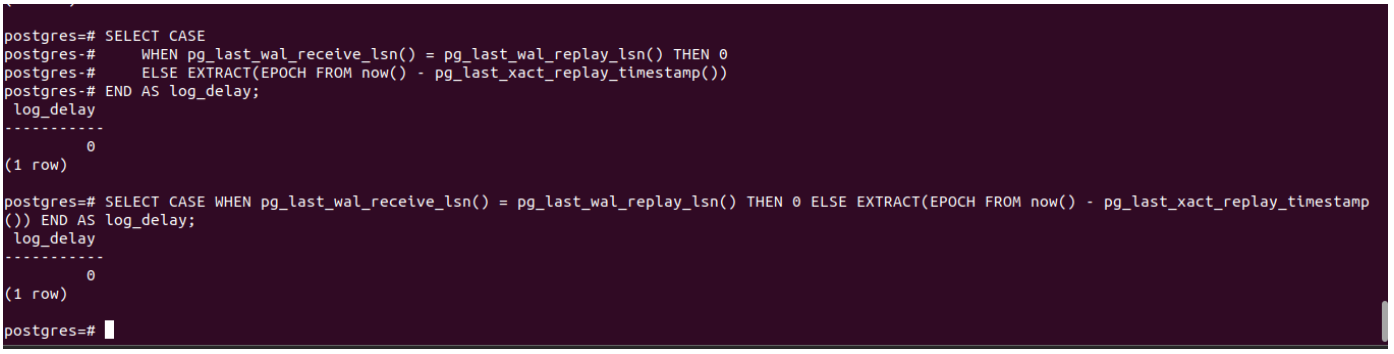
11.1 failover stream replication

verify the replication on slave server

before we start the failover , we need to verified the syn between the master and slave

in the slave run the below command and if its return 0 means no delay

```
SELECT CASE WHEN pg_last_wal_receive_lsn() = pg_last_wal_replay_lsn() THEN 0
ELSE EXTRACT(EPOCH FROM now() - pg_last_xact_replay_timestamp()) END AS
log_delay;
```



stop the primary server

once you confirmed everything is fine , you need to stop the services either by shutdown the server or stop PostgreSQL services
for this purpose i will stop the PostgreSQL services .
`systemctl stop postgresql`

```
postgres@ahmed-virtual-machine:~$ systemctl stop postgresql
postgres@ahmed-virtual-machine:~$ systemctl status postgresql
postgresql@12-main.service  postgresql.service
postgres@ahmed-virtual-machine:~$ systemctl status postgresql
○ postgresql.service - PostgreSQL RDBMS
     Loaded: loaded (/lib/systemd/system/postgresql.service; enabled; vendor preset: enabled)
     Active: inactive (dead) since Tue 2023-12-05 12:15:48 +03; 33s ago
    Process: 36730 ExecStart=/bin/true (code=exited, status=0/SUCCESS)
   Main PID: 36730 (code=exited, status=0/SUCCESS)
        CPU:
Show Applications
postgres@ahmed-virtual-machine:~$
```

promote the standby to read-write

next step is to promote the standby server to be able to read and write
using the following command

```
psql -c "SELECT pg_promote();"
#This command promotes the standby to a read-write primary.
```

```
postgres@ahmed-virtual-machine:~$ psql -c "SELECT pg_promote();"
 pg_promote
------------
 t
(1 row)

postgres@ahmed-virtual-machine:~$
```

no if you try to create database it will allow since we enable read-write

```
postgres=# create database kamaraj;
CREATE DATABASE
postgres=# \l
                                 List of databases
   Name    |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-----------+----------+----------+-------------+-------------+-----------------------
 ahmed     | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 cars      | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 kamaraj   | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 postgres  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 salah     | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
 test2     | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 testdb    | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
(9 rows)

postgres=#
```

edit the pg_hba.conf

edit the pg_hba.conf in slave server and add the ip of the master server to act as slave

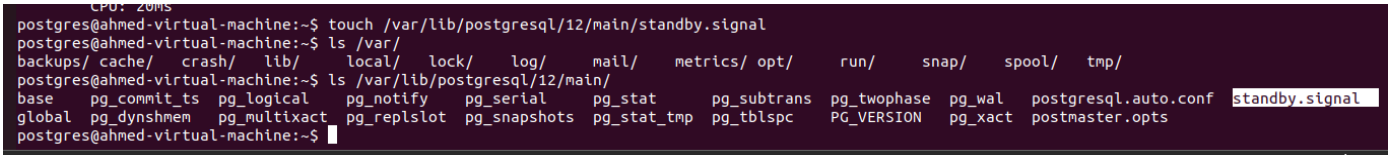
`cd /etc/postgresql/12/main echo "host replication replication 10.10.10.80/24 md5" >> pg_hba.conf`

```
postgres@ahmed-virtual-machine:~$ cd /var/lib/postgresql/12/main
postgres@ahmed-virtual-machine:~/12/main$ echo "host replication replication  10.10.10.80/24 md5" >> pg_hba.conf
postgres@ahmed-virtual-machine:~/12/main$
```

Create standby.signal on Old Primary

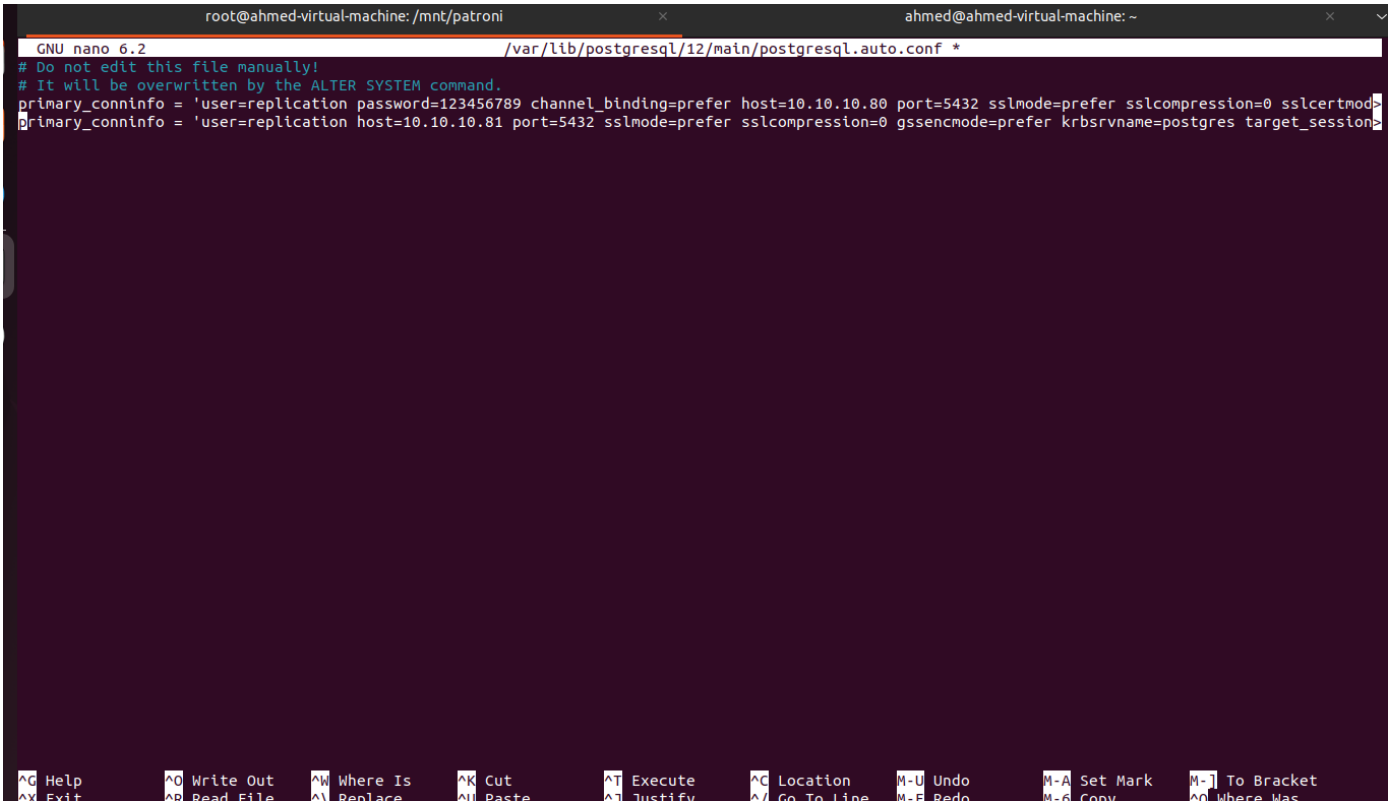
Create the standby.signal file on the old primary to ensure it starts as a standby when brought back online:

```
touch /etc/postgresql/12/main/standby.signal
```



also edit the below postgresql.auto.conf on master server
with the below paramter

```
#Update the postgres.auto.conf file with NEW MASTER SERVER DETAILS
vi /etc/postgresql/12/main/postgresql.auto.conf

#Modify the primary_conninfo parameter to reflect the new standby IP:

primary_conninfo = 'user=replication password=123456789 host=10.10.10.81
port=5432 sslmode=prefer sslcompression=0 gssencmode=prefer
krbsrvname=postgres target_session_attrs=any'
```



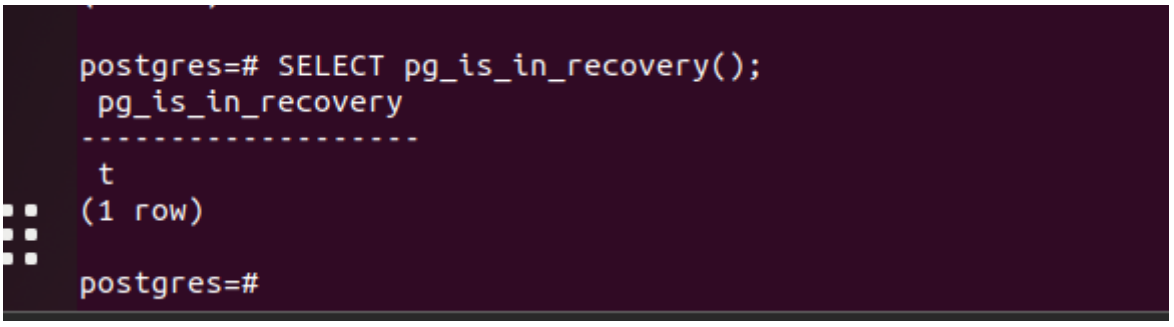
after that start the PostgreSQL services on master server

```
systemctl start postgresql
```

check if old master id read only

run the below command if its return T means the database is in read only if F means the database in read write

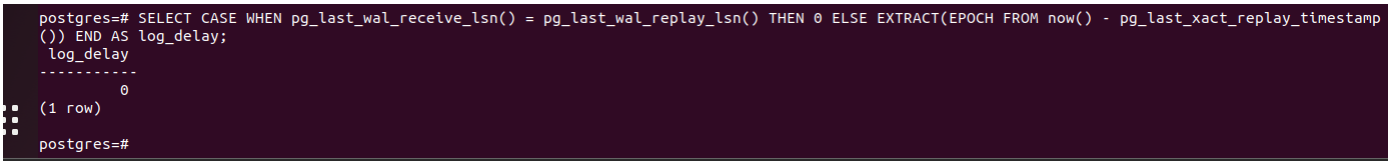
```
SELECT pg_is_in_recovery();
This query will return a single boolean value:
true if the server is a standby and false if it's the primary
```



failback

fist verify that database is syncing by running the below command on slave server and should be 0

```
SELECT CASE WHEN pg_last_wal_receive_lsn() = pg_last_wal_replay_lsn() THEN 0
ELSE EXTRACT(EPOCH FROM now() - pg_last_xact_replay_timestamp()) END AS
log_delay;
```



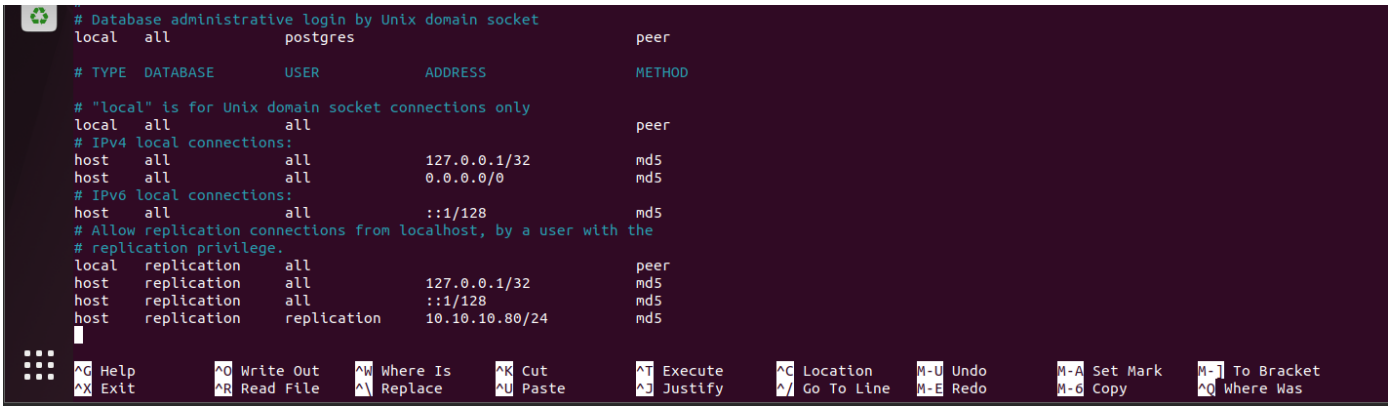
master failback config part

start by stopping the services on master server

```
systemctl stop postgresql
```

make sure the below parameter is added in master pg_hba.conf
host replication replicator 10.10.10.80/24 md5"

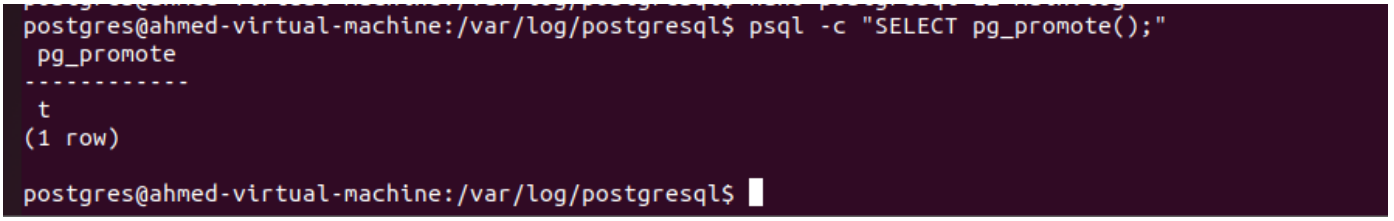
```
cd /etc/postgresql/12/main/
nano pg_hba.conf
```



promote the slave (old-master)

go to the slave server (old master) and make it read-write ,
using the below command

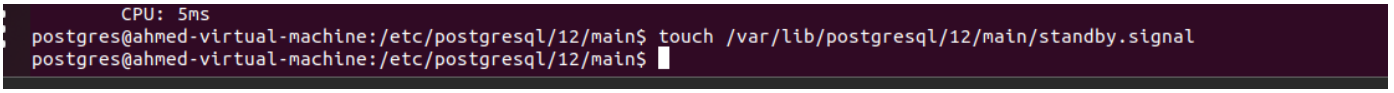
```
psql -c "SELECT pg_promote();"
```



create single file on master server (old slave)

create the singe file on salve (old master)

```
touch /var/lib/postgresql/12/main/standby.signal
```



Edit postgresql.auto.conf on Old Primary

```
cd /etc/postgresql/12/main
nano postgresql.auto.conf
```

add the below line ensure to mention the slave (old-master) ip

```
primary_conninfo = 'user=replication password=123456789 host=10.10.10.80
port=5432 sslmode=prefer sslcompression=0 gssencmode=prefer
krbsrvname=postgres target_session_attrs=any'
```

start the master (old slave) services

start the the services using the below command

```
systemctl start postgresql
```

check the log and observer if the server started as standby

`

cd /var/log/postgresql/

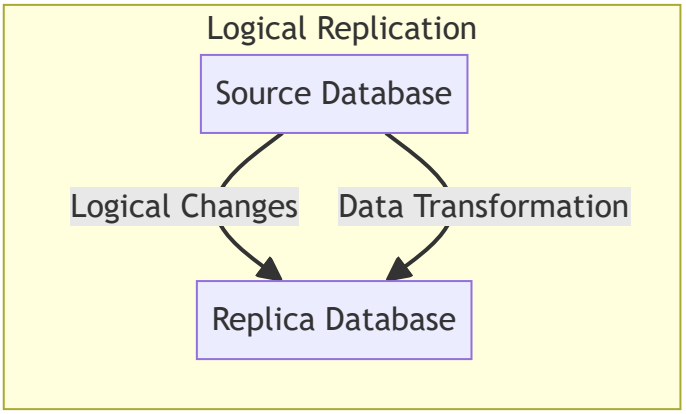
nano postgresql-12-main.log

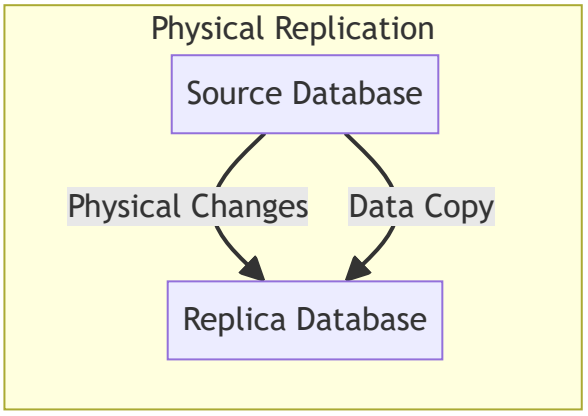
`

```
2023-12-05 16:38:53.188 +03 [34017] LOG:  background worker "logical replication launcher" (PID 34024) exited with exit code 1
2023-12-05 16:38:53.214 +03 [34019] LOG:  shutting down
2023-12-05 16:38:53.351 +03 [34017] LOG:  database system is shut down
2023-12-05 16:51:44.637 +03 [34633] LOG:  starting PostgreSQL 12.17 (Ubuntu 12.17-1.pgdg22.04+1) on x86_64-pc-linux-gnu, compiled by gcc (Ubuntu
2023-12-05 16:51:44.638 +03 [34633] LOG:  listening on IPv4 address "0.0.0.0", port 5432
2023-12-05 16:51:44.638 +03 [34633] LOG:  listening on IPv6 address "::", port 5432
2023-12-05 16:51:44.652 +03 [34633] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
2023-12-05 16:51:44.809 +03 [34634] LOG:  database system was shut down at 2023-12-05 16:38:53 +03
2023-12-05 16:51:44.810 +03 [34634] LOG:  entering standby mode
2023-12-05 16:51:44.828 +03 [34634] LOG:  consistent recovery state reached at 0/3039E08
2023-12-05 16:51:44.828 +03 [34634] LOG:  invalid record length at 0/3039E08: wanted 24, got 0
2023-12-05 16:51:44.841 +03 [34633] LOG:  database system is ready to accept read only connections
2023-12-05 16:51:44.931 +03 [34638] LOG:  fetching timeline history file for timeline 3 from primary server
2023-12-05 16:51:44.955 +03 [34638] LOG:  started streaming WAL from primary at 0/3000000 on timeline 2
2023-12-05 16:51:44.981 +03 [34638] LOG:  replication terminated by primary server
2023-12-05 16:51:44.981 +03 [34638] DETAIL:  End of WAL reached on timeline 2 at 0/3039E08.
2023-12-05 16:51:44.983 +03 [34634] LOG:  new target timeline is 3
2023-12-05 16:51:44.993 +03 [34638] LOG:  restarted WAL streaming at 0/3000000 on timeline 3
2023-12-05 16:51:45.247 +03 [34634] LOG:  redo starts at 0/3039E08
```

12. logical replication

deterrent than stream replication , where the master will send wall log , there we send the actual command such `(insert into t1 values (1,'''value'' ))`





physical replication

in physical replication , the replica is force to copy the whole database schema tables and so on from master server .
physical replication in this case can not support in replicating singe table

logical replication

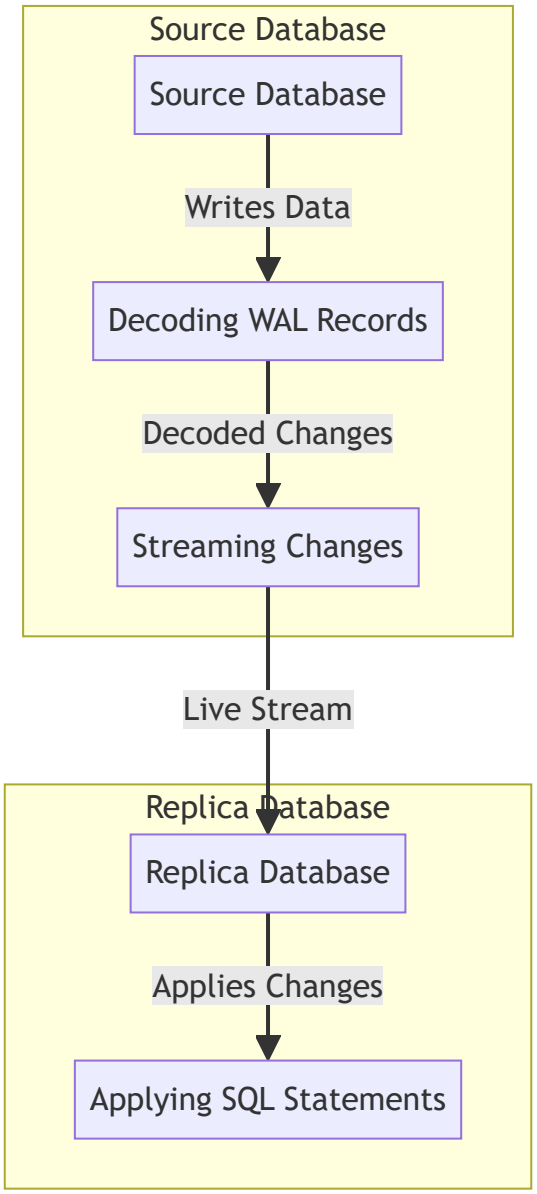
as mentioned only replicated the changes , which give the advantage of replicating singe table .
here the primary database will send the DML command to be replayed on standby server
here's the scoop:

1. **Decoding WAL Records**: In PostgreSQL, Write-Ahead Logging (WAL) records all changes made to the database. When using logical replication, the first step is to decode these WAL records. Think of it like decrypting a secret code; this process extracts the actual changes that were made to the data.
2. **Streaming to Replica Server**: Once those changes are decoded, they're streamed over to the replica server. This is like sending a live feed of the changes happening in the source database to

the replica, ensuring it stays up to date with the latest data modifications.

3. **Applying Statements on Replica**: On the replica server, these decoded changes are then applied as SQL statements. It's like having a copycat follow along with the source database's actions, executing the same SQL commands to mimic the changes.

So, in a nutshell, PostgreSQL goes through this process of decoding, streaming, and applying changes to keep the replica database in sync with the source. It's like a well-choreographed dance of data replication!



this whole setup the primary server is called publisher server , and replica is called subscriber server . similar to [MS SQL replicationudemy]

physical replication

1. must have the both server must have identical configuration
2. the data has to be placed on file system to be similar to both master and slave

3. these type of replication wont work to migrate from older version to newer version .
   sudo update-alternatives --install /usr/bin/initdb initdb /usr/lib/postgresql/12/bin/initdb 1

logical replication limitations

1- doesn't support DDL command corresponded to creating index

logical replication setup

sequence of steps:

1- instantiate 2 PostgreSQL database cluster
2. configure the publisher with "wal_level =logical"
3. start the instances
4. create a database and the tables