

**1-Introduction**

Base de données avec un moteur transactionnel, qui dispose d’un langage PL/PgSql et de nombreux outils autour :

* Client cli : psql (ligne de commande)
* Client GUI : pgAdmin (interface graphique)
* Postgis : volet géographique
* Nombreux système de réplications
* Système d’extensions

Nous avons un outil pour la bonne gestion des transactions le MVCC, client/serveur avec une gestion de connexion.

Il fait appel au moteur relationnel ACID : atomic, sonsitent, isolated, durable.

Atomic : une transaction se fait en entier ou pas du tout

Consistent : en cas d’erreur lors d’une transaction l’état redevient celui d’avant celle-ci

Isolated : une transaction en cours n’interfère pas avec les autres.

Durable : les données sont dispos en cas de redémarrage (système de recovery)

**Définitions**

+- Cluster Machine > Cluster PG > Database > Rôle / Schémas > Tables > Champs

Nous avons le cluster au sens machine (différentes machines qui communiquent les uns avec les autres [cloud, VM, conteneur …) et le cluster au sens postgre c’est une instance de postgre c’est une première enveloppe contenant un ensemble de database avec un moteur et une certaine version.

+- Cluster Machine : plusieurs machines communiquant entre elles avec de la réplication.

+- Cluster PG : instances avec allocation de ressources spécifiques et définition de configuration.

+- Database : ensemble de données structurée (une application par exemple)

+-Rôles/Users : utilisateurs avec login et mot de passe.

+-Schéma : Namespace ou espace de nom dans une base de données (un group d’éléments comme attribuer les droits sur certaines tables et pas toutes du namespace)

+-Tables : espace de stockage logique dans une base de données.

+-Champs : une colonne de tables

+-Lignes : une table est structurée en colonne/ligne

**2-Installation**

**Sur Debian**

apt-get install postgresql

apt-cache search postgresql

service postgresql status

service postgresql stop

service postgresql start

service postgresql reload (évite les coupure de service pour faire des modification de configuration)

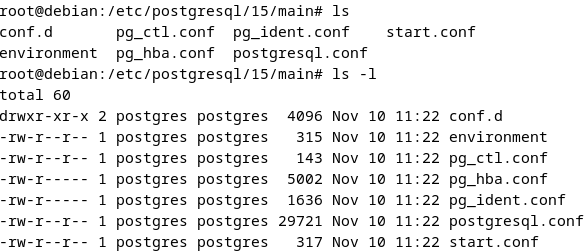
lancer une instance de postgre : ???

Supprimer postgresql

apt purge postgresql-15 postgresql-client-15

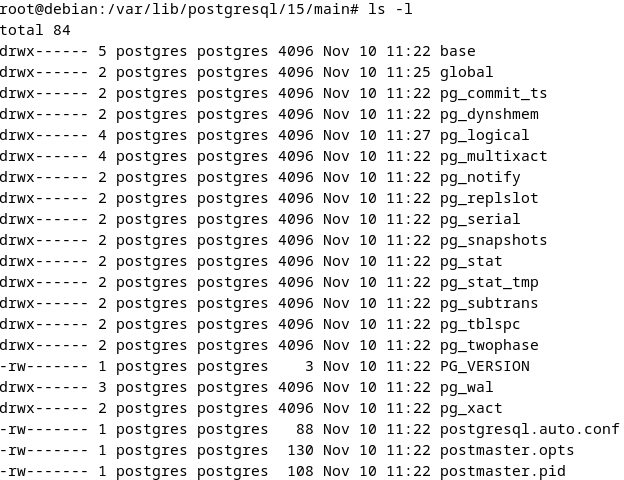
**Les principaux répertoires**

+- configuration : /etc/postgresql/<version>/<clusterpg (par defaut main)>



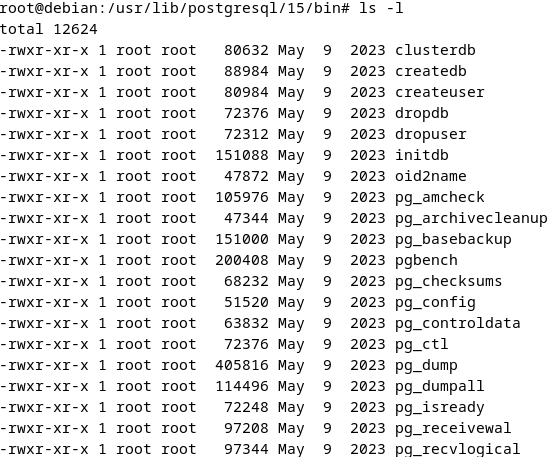
+- data : /var/lib/postgresql/<version>/<clusterpg (par defaut main)>

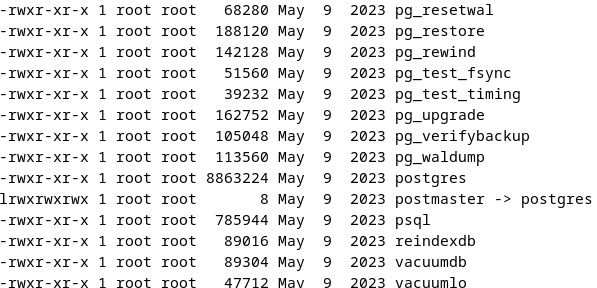
Eléments permettant le bon fonctionnement du moteur.



+- binaire principal : /usr/lib/postgresql/<version>/bin/postgresql

Nous avons les binaires comme le pg\_ctl pour lancer postgresql au tout début.





**3-binaires**

Les binaires à connaître et indispensable.

+- pg\_ctl :

+- gestion de l’instance/cluster

+-start/stop/kill

+-init : création autre espace de datas

+-promote : promotion du standby

+- psql : (un autre interface graphique avec pgadmin)

+- client de connexion à un cluster

+-précision utilisateur et/ou db

+-passage de sql en cli ou script sql

Permet d’avoir toutes l’aide et le faire de suite.

Binaires spécifiques à debian

+- pg\_createcluster :

+- création d’un cluster (une instance PG)

+- créatin des répetoires (/etc /var/lib/)

Permet de revenir vers une instance postgre (crée l’instance et les répertoires)

+- pg\_dropcluster :

+- suppression d’un cluster

+- cluster arrêté

Permet de supprimer un cluster ou l’arrêter

+- pg\_lscluster :

+- liste les clusters

Permet de lister les cluster présent sur la .

+- pg\_ctlcluster :

+- équivalent du pg\_ctl

+- contrôle du cluster (stop/start d’un cluster…)

Gérer le cluster spécifique au cluster et interagir individuellement

Spécifiques Sauvegarde

+- pg\_dump:

+- sauvegarde d’une instance

+-différent formats : plain text, binaire ….

+-différents niveaux d’objets (cluster/db/table/schéma)

Permet de sauvegarder avec finesse une instance ou certaine table avec des formats de sortie. (binaire utilisable que pour postgre) si c’est en plain-text on pourra le restaurer directement pas psql

+- pg\_dumpall: (intégrale du cluster)

+- sauvegarde intégrale en format binaire

+- pg\_restore:

+- restauration à partir d’une sauvegarde à partir de (pg\_dumpall)

Wrappers

+-équivalent de commandes sql

+-createdb :

+- création d’une base de données

Evite de se connecter en psql et lancer la commande create …

+-dropdb :

+- suppression d’une base de données

+-createuser :

+- création d’un utilisateur

(script shells)

+-dropuser :

+- supprime un utilisateur

Maintenance

+-reindexdb :

+- ré-indexation des index avec des paramètres pour restreindre le périmètre car il peut avoir des ralentissements.

+-vacuumdb :

+- tâche de maintenance (ménage)

+-vacuumlo :

+- suppression de large objects

**Spécifiques systèmes avancés**

+-pg\_controldata:

+- vérifie l’état du serveur et des infos critiques (control files vitaux pour le bon état de postgre à checked )

+-pg\_resetwal:

+- en cas de crash avec pb de WAL (Write Ahead Logging) = mode d’archivage permet une autre sauvegarde et manière de restaurer mais en cas de crash mais des pertes de data dernier recours)

+- attention : datas inconsistentes (dernier recours)

+-pg\_receive\_wal:

+- récupération des WAL d’une autre DB

Dans la réplication et permet de jouer les fichier d’archivages et récupérer à distance ou local .

+-pg\_controldata:

+- vérifie l’état du serveur et des infos critiques (control files vitaux pour le bon état de postgre à checked )

+-pg\_basebackup:

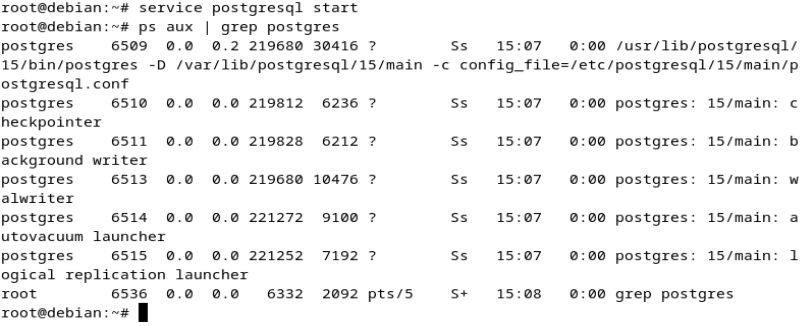
+- récupération de datas par une connexion à une autre BD (ex : init réplication)

Pour mettre une réplication en place.

**4-Procédures arrêt et démarrage**

> service postgresql status

>ps aux | grep postgres



Nous avons le binaire et la localisation des datas et éléments de configuration -c ….

>less /lib/systemd/system/postgresql.service



Nous avons un bin/true mais pour voir les commandes de lancement ! bin bash

>less /etc/init.d/postgresql

Postgresql : démarrage et arrêt (au même moment)

+-important pour un moteur de BDD surtout en relationnel

+- consistence des données

+- capacité à récupérer les données manquantes

Démarrage

1. Ressources ? : Affectation de la mémoire partagée

+- shared\_buffer (défaut 128 MB)

+- paramètre dans /etc/…/postgresql.conf

+- paramétré par clusterPG

1. Où on en est ? : Lecture des fichiers de contrôle (control files)

+- pg\_control dans $PGDATA/global (/var/lib/postgresql/15/main/global/pg\_control) où se situe les control files état arrêt/démarre en cas d’arrêt brutal.

+- état de la base en fonction de l’arrêt

1. Si récupération nécessaire ? : checkpoint

+- où sont disponibles les transactions manquantes ?

+- sur data file (disque) ou non ?

1. Si transactions disponibles sur WAL : (pour compléter les datas en jouant les transactions)

+- relancement des transactions

1. Si pas de de récupération possible ? erreur

On ne peut pas démarrer

1. Si tout est ok écoute 5432 et autorise les connexions

Arrêt

1. OS : Signal d’arrêt

+- Si Type SIGINT (intermédiaire)

1. Nouvelles connexions coupées
2. Stop les autres connexions
3. Rollback des transactions en cours (ne laisse pas terminer même en cours et revient en arrière)

+- Si Type SIGTERM (soft)

1. Coupe les nouvelles connexions
2. Fini les transactions en cours

Dans tous les cas :

Ecriture d’un checkpoint

Ecriture de la mémoire sur disque (data files)

Mise à jour des control files (pg\_control)

Rq : prend du temps (datas) et attente si un checkpoint est déjà en cours.

Si SIGQUIT (arrêt brutal)

+- pas de checkpoint

+- pas d’écriture des datas sur disque

+-pas de maj des control files

+- recovery

+- pertes de transaction (datas)

**5-Fonctionnement : les processus**

+- processus père : postgres -D <data\_dir> -c <conf\_dir>

+- principe de fork

+- 2 type de fils

+- les processus du moteur

+- les processus affectés aux connexions

Les processus ??

+- le checkpointeur : permet de poser des jalons mémoire/disque

+- écriture des dirty pages (mémoire) sur les data files (disque)

+- paramétrage de cette fréquence d’écriture

+- soit en durée et/ou en nombre de WAL switch.

(Write Ahead Logging) stock toutes les transactions pour rejouer en cas de mauvais arrêt très utile.

+- configuration des postgressql.conf

+- param : checkpoint\_segments et checkpoint timeout (position du jalon)

+- important pour le recovery notamment (coût en perf)

Pause des jalons pour savoir où on est arrivé en écriture disque et ensuite on continu et ainsi de suite.

+- background writter : écriture des dirty pages (mémoire) sur data files (disque)

+- wal writer : écriture de wal buffer sur disque

Wal buffer tous les logs transactions (insert …) sont dans un buffer mais pas sur disque et qui est charger sur disque avec des tailles définie (16Mo)

+- autovaccum launcher : en charge de l’autovaccum

Faire de la défragmentation (cleaner)

+- stats collector : calcule les statistiques

+- hyper important : calcule les statistiques

| (traitement efficace des requêtes)

+- écriture dans pg stats\_temp

Faire les requêtes plus rapidement comme un select contrôle le volume des données en statisqtique pour exécuter plus rapidement.

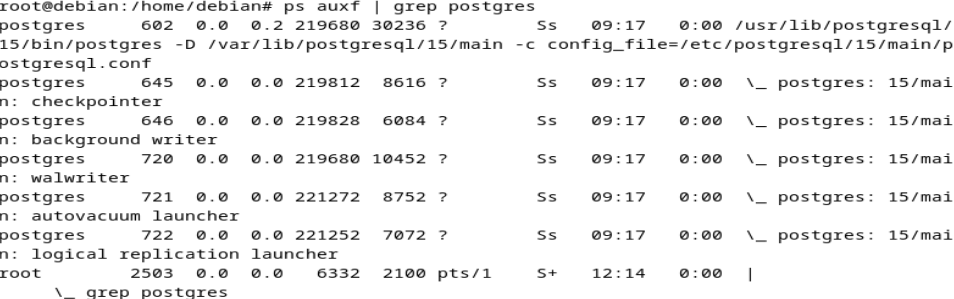
+- logical replication launcher :

+- en cas de réplication logique

+- lance un process pour les standbys qui se connectent

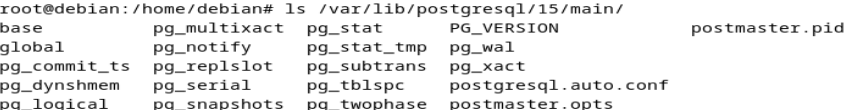
+postgres (idle) : pour chaque connexion

+- allocation de la ressource +- work memory



La première instance de postgres est main mais si on veut plusieurs instance et isolation c’est par cluster

Nous avons les fichiers de configuration de la data



Nous avons les fichiers d’instances ici une c’est le main



**6-Fonctionnement : la mémoire**

+- important pour un moteur de BDD

+- accessibilité rapide à la donnée

+- la mémoire n’est pas pérenne

+-cas d’arrêt : prévu > système d’écriture

+- incident : imprévu > perte de la mémoire / memory (rejoue à partir des wal)

+- postgres scinde en 4 la mémoire disponible :

+- shared buffer : mémoire partagée : (shared bufferwal buffer, clog buffer)

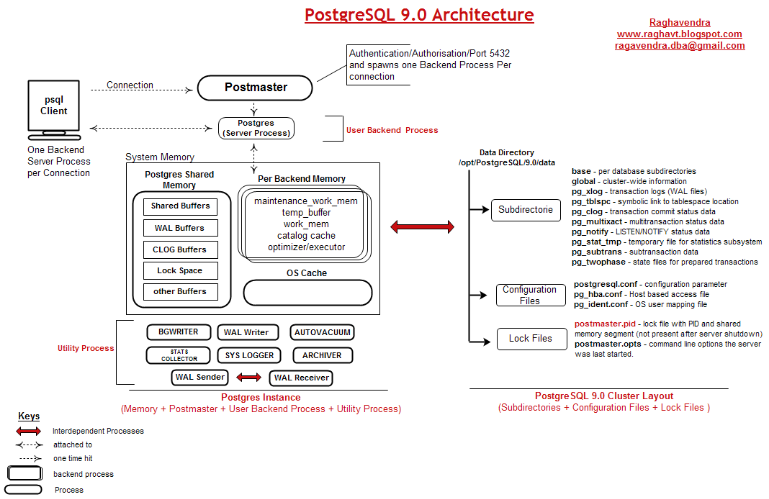
+- wal buffers :

+- work mem

+- maintenance work mem

+- on appelle shared memory = shared buffer + val buffer + c-log-buffer

+- aide pour configure : <https://pgtune.leopard.in.va/>



Shared buffer

+- par défaut 128 Mo

+- au mini 128ko + 16ko par max connexion

+-si plus 25% de celle du système

+- cette partie de la mémoire est ensuite écrite sur datafile

+- data plus rapide que sur datafile

+- background writer : process chargé de la copie

WAL buffer

+- en charge des archives des transactions

+- cette partie est ensuite écrite dans les WAL files puis archivé

+-wal write : en charge de la copie (wal buffer vers sur le disque)

+- archiver + process en charge de l’archivage

Work mem

+- mémoire allouée pour chaque utilisateur

+-permet les tri, classement, filtre, jointures… des requêtes clientes

+- affectation pour chaque connexion

+- cf postgresql.conf > reloead nécessaire

Maintenance work mem

+-mémoire dédiée aux opérations de maintenance

+-vaccum (défragmentation en cas de table supprimée), create index, reindex …

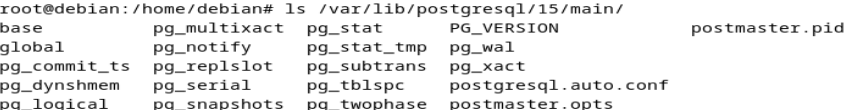
+-valeur qui peut jouer sur la vitesse de restauration

**7-Fonctionnement : les répertoires**

+- 2 essentiels

/var/lib/postgresql/15/main/ (instance et base sql pour restauration ou graphique)

/etc/postgresql/15/main/ (configuration)



+-base

+-localisation des bases

+-template 0 et template 1 : appelée lors de la création de BD (base modèle pur créer des databases)

+-postgresql

+-classement par oid (tables de correspondance entre oid et objects (oid id spécifique à

Postgres. /main/base/ls (requetes sur oid corrrespond une table ou autre équivalence en objet et oid.

+-SELECT relname , oid, refilemode FROM pg\_class wher relname=’paul’;

+-ou SELECT pg\_relation\_filepath(‘paul’)

+-free Space Map (FSM) et visibility Map (VM) : espace dispo, lien avec Vacuum

+-global

+- control files : pg\_control appelé au démarrage

+- connaitre l’état du moteur

Dans le main/global/

/pg\_control

Gère les control file et permet à postgres de savoir où il est et couplé au checkpoint.

Pour reprendre si nécessaire

/pg\_wal

Contient les fichiers d’archivage les écritures de log de transaction contiennent à chaque fois que postgres reçoit une requête.

/pg\_stat

Statistique permet l’élaboration des plans de requêtes (select retranscrit sur plusieurs requêtes pour optimiser la requête et pour gagner du temps)

+- pg\_tblspc :

+- pointeur vers les tablespaces

+- pg\_repslot :

+- informations sur la réplication par slot



Le conf.d principale fichier de postgres personnalise des configurations ou faire des includes.

Le pg\_hba permet la connexion à la base de données (certaines ip ou range\_ip pour toutes les bases ou certaines ou sur des users)

Pg\_ctl.conf passer des options supplémentaires au lancement du service postgresql

Le start.conf mode auto pour la machine qui redémarre ou manuel.

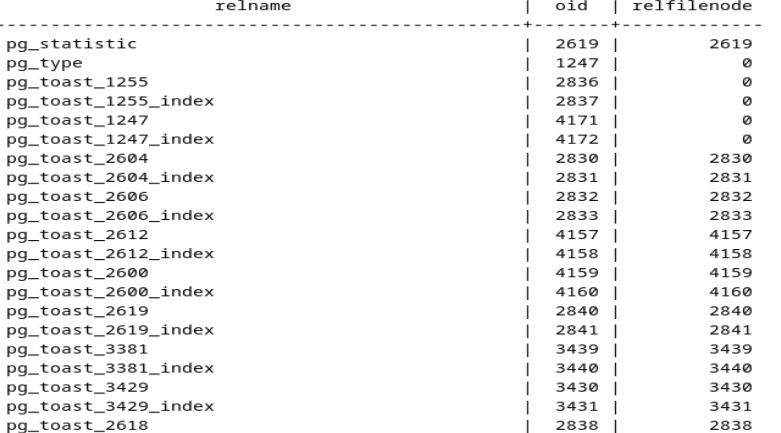
Le pg\_ident.conf fait du mapping user système et user postgres (user system pour psql)

Entrer dans psql

>su – postgres

>psql

Postgres# SELECT relname, oid, relfilenode FROM pg\_class;



**8-Autorisation d’accès: pg\_hba**

/etc/postgresql/15/main

Netstat -ntaup

Le postgres sql écoute 5432 et écoute le 127.0.0.1

+- ip d’écoute : listen adresse

+- ou ip

+- plusieurs séparés par virgule

+- changement restart

/etc/postgres/15/main/postgresql.conf

Listen\_addresses = ‘\*’

service postgresql restart

si on a 0.0.0.0 :5432 on écoute sur toutes les ip

+- pg\_hba.conf (host base authentification)

+-Host Based Authntification

+- /etc/postgresql/…

+- lu de haut en bas (pour le déroulement de la configuration)

+- reload pour prise en compte

+- filtre multifacteurs :

+- types

+- database

+-user

+-ip/net

+-méthode d’authentification md5, ldap)

Les facteurs

+-type : local ou host (distant)

+-database : spécifique ou global (all)

+-user : spécifique ou global (all)

+-adresse :

+-une ip, un masque de sous réseau (CIDR)

+-ipv4 ou ipv6

+-méthode :

+trust : sans login

+peer : user de l’OS (exemple postgres)

+password : user/password simple

+md5 : avec chiffrement

+reject : refus (utile en cas de maintenance)

+ldap…

Exemple

Entrer dans psql

>su – postgres

>psql

>\q

postgres@postgres: -$ logout

on va créer un usr avec psql

>create user paul encrypted password ‘password’

Puis dans pg\_hba: host all all 127.0.0.1/32 md5

psql -h 127.0.01 -U paul postgres

Password for user paul : …

Connexion à la base

Modif pg\_hba: host all all 127.0.0.1/32 reject

Nous n’avons plus accès

**Si nous avons des problèmes il faut avoir le reflexe :**

**less /var/log/postgresql/postgresql-11-main.log**

/---création de database---/

>su – postgres

>psql

>create database pilpaul

>\q

pg\_hba> host pipaul paul 127.0.0.1/32 md5 #paul peut se connecter à la table pipaul

pg\_hba> host postgres criss 127.0.0.1/32 md5 #le user criss peut uniquement se connecter à la base postgres

connexions : memory

+- max\_connexion : 400 bytes par connexion

Rq : si elles ne sont pas établies

+- connexions établies : paramètres work\_mem par connexion établie (128mo)

**9- psql : le client indispensable**

PSQL : les débuts

2 modes de connexions en local ou passe par la socket

Remarque :

+- cluster : main

+- port : 5432

+- serveur : localhost

+-user : postgres

Créer un utilisateur postgresql

Depuis root >su postgres (l’utilisateur par défaut n’a pas de mot de passe et il est accessible que en local)

postgres@debian: createuser - -interactive criss (option -P pour un mot de passe sinon le faire via psql)

shall the new role be a superuser? (y/n) y

connexion psql: postgres@debian: psql

postgres=# \password criss

Enter new password for user criss: criss

Enter it again: criss

+- accès à psql avec des options et en passant par la socket (pas local)

psql -h 127.0.0.1 -p 5432 -U criss -d postgres

Rq : --cluster dans le cas de plusieurs cluster ou nom différent

Il faut aussi le déclarer dans /etc/postgres/15/main/pg\_hba.conf

# IPV4 local connections

host all all 127.0.0.1/32 scram-sha-256

host postgres criss 127.0.0.1/32 scram-sha-256

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*UTILISATION DECOULE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

postgresql:commande psql :

\l = liste des bases

\d = liste des tables

\du = liste des utlisateurs

\dn+ = lister les schémas et les droits

\q = quitter

\h = aide

USE labase = pour se connecter sur la base

\c labase = pour se connecter sur la base

SELECT version() ; = version PostgreSQL

SELECT current date ; = date actuelle

\ ? aide mémoire psql

\c nom\_base nom\_utilisateur #connecte à la base nom\_base avec e rôle nom\_utilisateur

Créer un utilisateur postgresql non sécurisé avec les commandes SQL

postgres=# CREATE ROLE <nom\_utilisateur> LOGIN;

postgres=# ALTER ROLE <nom\_utilisateur> CREATEDB;

postgres=# CREATE DATABASE <nom\_base\_de\_donnee> OWNER <nom\_utilisateur>;

postgres=# ALTER ROLE <nom\_utilisateur> WITH ENCRYPTED PASSWORD 'mon\_mot\_de\_passe';

--------------------Linux----------------------------------------------------------------------------------------------------------

>cat /etc/passwd | cut -d : -f 1 |sort



Créer un utilisateur :

root>adduser criss

Ajouter l’utilisateur au groupe wheel :

root>usermod -a -G wheel criss

root>groups criss

supprime l’utilisateur :

root>userdel -r criss

l’utilisateur perds la commande sudo su :

User is not in the sudoers file

l'utilisateur debian n'a pas accès à sudo donc

> su root

password:???

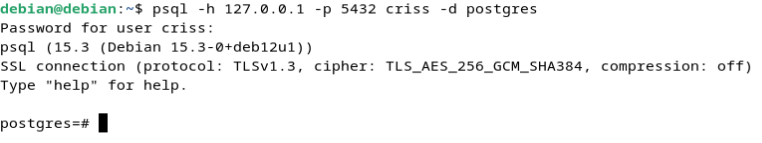
root@debian:/home/debian# nano /etc/sudoers

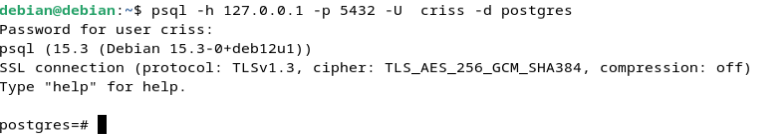
#User privilege specification

root ALL=(ALL:ALL) ALL

debian ALL=(ALL:ALL) ALL #ajouter debian pour utiliser sudo

--------------------------------------------------------------------------------------------------------------------------------





Pour se connecter à une autre instance

psql –cluster 15/instance1 -p 5432 -h 127.0.0.1 -U criss -d maBase

Nous pouvons passer des commandes psql en ligne de commande directement sans être dans psql :

postgres@debian: psql -c "\l+" / psql -c "\dt" #liste toute les tables

**pour la doc** : postgres@debian: psql - -help

Nous pouvons faire directement du sql :

postgres@debian: psql -c "SELECT \* FROM table"

postgres@debian: psql -f monficsql.sql

**10- Databases : template, création**

Databases:

Serveur > Cluster > Databases > Tables

+-database = un contenant partageant ses ressources (au sein du cluster)

+- l’objet élémentaire est la table

+- les tables sont composée de colonnes et de lignes

+-les colonnes sont différentes types : int, varchar, date ….

/etc/postgresql/15/main/pg\_hba.conf



Local all all 127.0.0.1/32 peer

Postgres=# \l

Tables : postgres

template0 #ne pas toucher c’est un template de référence

template1 #templace créer depuis template0 donc mieux utiliser template1

Création d’une database

CREATE DATABASE name

[ [ WITH ] [ OWNER ] [=] user\_name ]

[TEMPLATE [=] template ]

[ENCODING [=] encoding ] >> encodeage (utf8 ….)

[LC\_COLLATE [=] lc\_collate ] >> tri des strings

[LC\_CTYPE [=] lc\_type ] >> type de caractère (maj, min …

[TABLESPACE [=] tablespace ] >>endroit où sont stockées la database autre

[CONNECTION LIMIT [=] connlimit ] >>limite la connexion

Postgres=# CREATE DATABASE crissbas ;

CREATE DATABASE

Postgres=#\l #visualise les databases



Postgres=#drop database crissbase ; #supprime la base

+-créer une base : CREATE DATABASE mydb WITH ENCODING ‘UTF-8’ TEMPLATE TEMPLATE1;

Templare1 > template0 > creation DB

+- avec le wrapper

created -T template1 mydn

Rqm: on peut utiliser n’importe quelle base à l’arrêt comme template (modèle)

Postgres=#alter database mydb

ALLOW LIMIT OWNER TO RENAME TO SET

CONNECTION LIMIT OWNER TO

Postgres=#alter database owner to criss; #renomme le propriétaire de la database

Postgres=#create database mytemplate1,

CREATE DATABASE

Postgres=#create table matable (id int, champs1 varchar(255));

Postgres=#\c #postgres template0 template1 mytemplate1

Postgres=#\c mytemplate1 #you are now connected to database mytable as user postgres

Postgres=#\dt #did not find any relations

Ici nous avons créé la table dans la base de données postgres donc non !!!!

Postgres=#drop table matable ; #supprime matable sur la base de données postgres on la veux dans mytemplate.

Postgres=#\c mytemplate1#you are now connected to database mytempplate1 as user postgres

mytemplate1=# create table matable (id int, champs1 varchar(255));

mytemplate1=#\dt

mytemplate1=#\q

Postgres=#create database mydatabase #sans template

mydatabase =#\c mydatabase

mydatabase =#\dt #did not find any relations on efface la base de donnée on va créer avec un template

Postgres=#create database mydatabase template mytemplate;

Postgres=#\c mydatabase

Postgres=#\dt et nous avons les autres tables qui ont été créée sur l’autre template.

Postgres=#

Postgres=#

Postgres=#

Postgres=#

**11- POSTGRESQL: Tables**

Serveur > Cluster > Databases > Tables

+- database = un contenant partageant ses ressources (au sein du cluster)

+- l’objet élémentaire est la table

+- les tables sont composées de colonnes et de lignes

+- les colonnes de différents types int, char, varchar, date, float, double, timstamp …

+- différents types de tables :

+- classiques (create table standard)

+-temporaires : seulement en mémoire (perdue en cas de crash ou arrêt mais très rapide) pour job

+-avec héritage : récupère les colonnes de la table mère

+-partitionnement : découpage en plusieurs (sous table) par élément d’une colonne

+-non loguée : sans écriture des logs (traces des transactions) > plus rapide mais perte datas si crash.

Création d’une table

CREATE TABLE matable (id int, champs1 varchar(255))

Supprimer une table

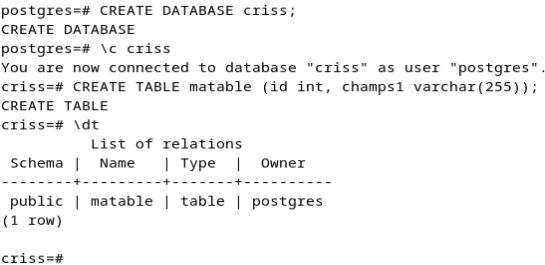
DROP matable ;

Vider la table :

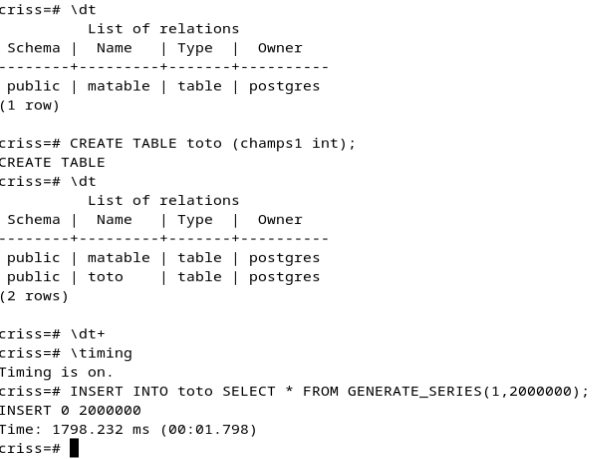
TRUNCATE TABLE matable ;

Modifier

ALTER TABLE (ALTER COLUMN ou ADD COLUMN) …



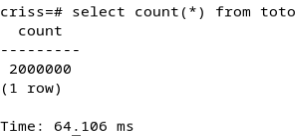
Inserer 2 millions de valeurs dans la table toto



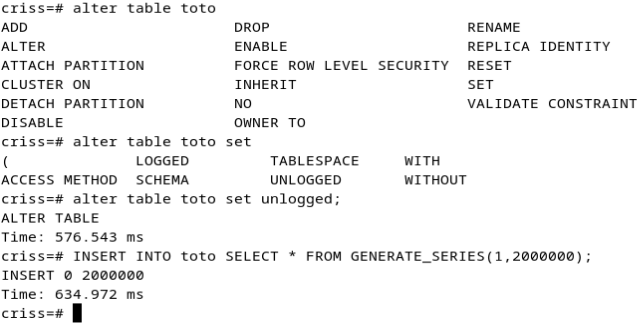
\dt+



Compte les lignes :



Nous passons en mode non logué (sans les logs nous avons divisé par deux le temps d’insertion mais on ne peut rejouer la transaction ))

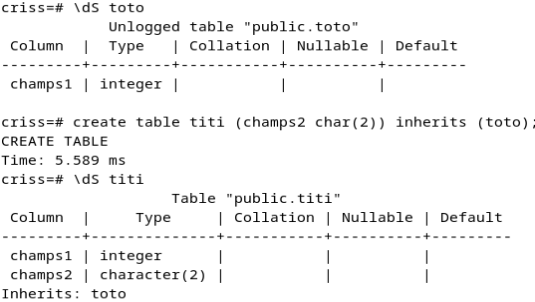


Temporaires (en mémoire)

CREATE TEMPORARY TABLE (champs1 int);

Héritage

CREATE TABLE titi (champs2 char(2)) INHERITES (toto);



Partitionnement

CREATE TABLE temperature ( measure\_timestamp TIMESTAMPZ,

Sensor\_id INTEGER,

Measure\_value DOUBLE PRECISION,

Measure\_unite custom\_enum\_unit)

PARTITION BY RANGE (measure\_timestamp);

CREATE TABLE temperature\_201709

PARTITION OF temperature

FOR VALUES FROM (‘2017-09-01’) TO (‘2017-10-01’);

CREATE TABLE temperature\_201710

PARTITION OF temperature

FOR VALUES FROM (‘2017-10-01’) TO (‘2017-11-01’);

INSERT INTO temperature SELECT

Tval, sensorid, (100.0\*random())-50.0, ‘°C’

FROM generate\_series(‘2017-10-01 00:00:00’, CURRENT\_TIMSTAMP, ‘1 minute’::interval) tval

CROSS JOIN generate\_series(1,1000,1) sensorid

<https://www.loxodata.com/post/cantwaitpg11_part/>

partitioner une table pour travailler avec des tables plus rapidement sans trop de données

**12- POSTGRESQL: Index : B-Tree**

+-Objectif : augmente la vitesse des requêtes

+- index = table structurée pour un besoin

+-index = pointeur

+-attention : demande de la ressource et demande de la performance

+- index = entretien (réindex)

+-index = baisse performance en écriture

+-postgresql dispose de plusieurs types d’index :

+- b-tree (r-tree spatial postgis)

+- hash

+- Gist \*

+-Gin \*

+- Brin \*

+-création d’un index :

CREATE INDEX idx\_criss ON crisstable USING BTREE (t\_champs1);

+-B-Tree : cas d’utilisateion : < <= >= > (Ne s’applique pas : recherche text ou plainText recherche de mot)

+-table longue

+-champs de jointures

+-peu de valeurs rapatriées

+-éviter sur les tables mise à jour régulièrement

+-complément : <https://fr.wikipedia.org/wiki/Arbre_B>

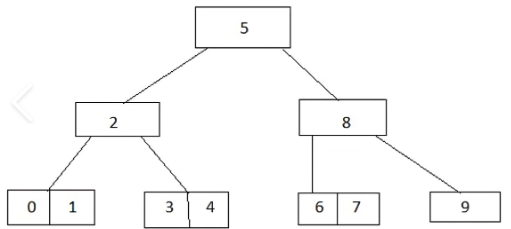
B-Tree

+- Balancing tree : arbre équilibré (bonne répartition des données arbre inverse)

+- créé par Rudolf (boeing)

+- Principe :: ex : 0 1 2 3 4 5 6 7 8 9

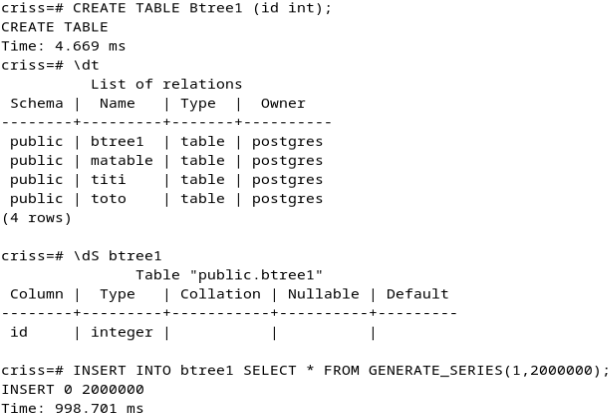
Clés = n branche-1



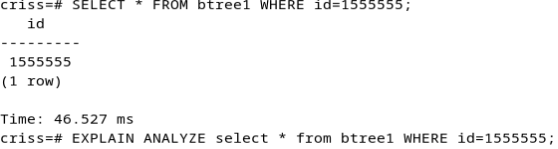
valeur

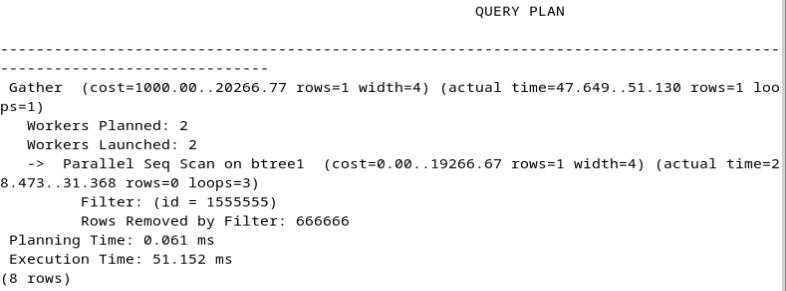
Recherche 6 = 6>5 => 8>6 = 6

Création d’une table B-tree

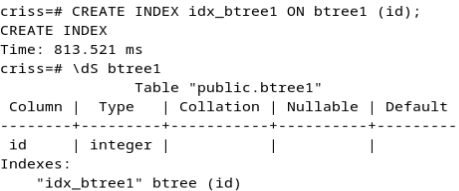


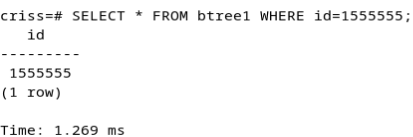
Recherche d’une valeur :





Nous allons créer un index





Nous avons réduit le temps de la requête

**13- POSTGRESQL: Index : hash**

+-Objectif : augmente la vitesse des requêtes

+- index = table structurée pour un besoin

+-index = pointeur

+-attention : demande de la ressource et demande de la performance

+- index = entretien (réindex)

+-index = baisse performance en écriture

+-postgresql dispose de plusieurs types d’index :

+- b-tree (r-tree spatial postgis)

+- hash

+- Gist \*

+-Gin \*

+- Brin \*

+-création d’un index :

CREATE INDEX idx\_criss ON crisstable USING HASH (t\_champs1);

+-B-Tree : cas d’utilisateion : = ou <> (performant s’il trouve cette valeur ou qui n’est pas cette valeur)

+-table longue

+-champs de jointures

+-peu de valeurs rapatriées

+-éviter sur les tables mise à jour régulièrement

+-complément : <https://fr.wikipedia.org/wiki/Arbre_B>

Hash

+- une fonction de hashage => hash la clé (clé/valeur)

+- le résultat est un entier qui détermine le positionnement du pointeur

+- le pointeur pointe vers la ligne correspondante

Exemple :

1. La table

|  |  |
| --- | --- |
| Nom | Données |
| Paul | Rouge |
| Jean | Vert |
| René | bleu |

1. Le hachage

hash(Paul) = 2 (prend Paul et hash Paul a une valeur entière 2)

hash(Jean) = 3

hash(René) = 1

1. L’index

La ligne généré par le hash comme hash(Paul) = 2 correspond à la ligne du pointeur ligne 2 et on lui affecte un pointeur (B)

|  |  |
| --- | --- |
| Index | Pointeur |
| René  Paul  Jean | A  B  C |

1. Recherche

1- je cherche Paul

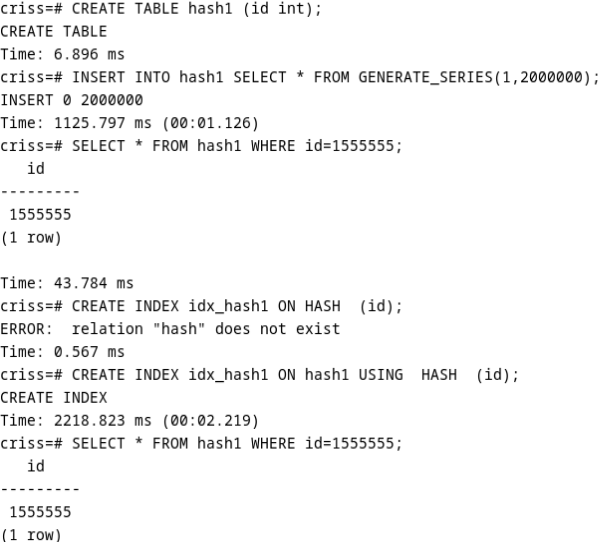
2- hash(Paul)=2 #ligne 2 et on trouve le pointeur B et je suis renvoyer à la ligne Paul Rouge

3- recherche du pointeur dans l’index 2

4- correspondance du pointeur dans la table

5-rouge

Select Time : 0.518 ms au lieu de 43.784 ms : avec le hash

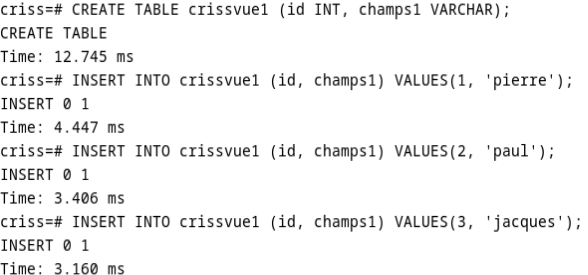


**14- POSTGRESQL: Vues classique et matérialisée**

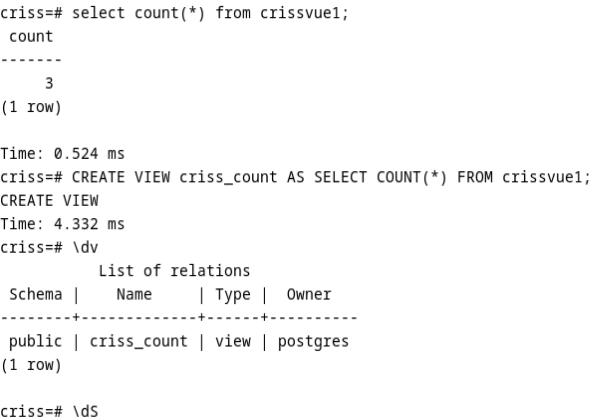
+-vue classique : requête utilisable sous forme de table

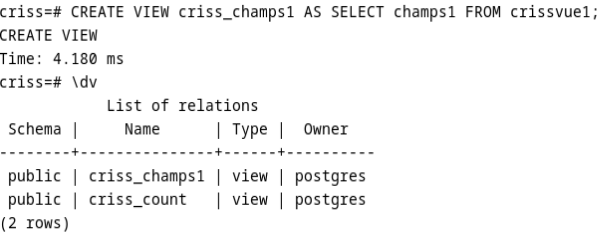
Permet aussi de fournir des droits sur une vue et non sur la table comme un ex tract de données.

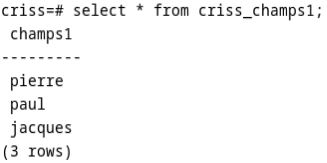
Création de la table et insertions



+- création de la vue

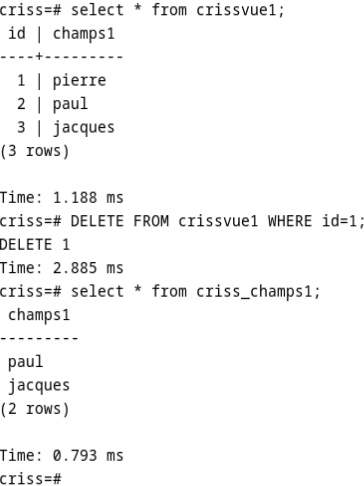




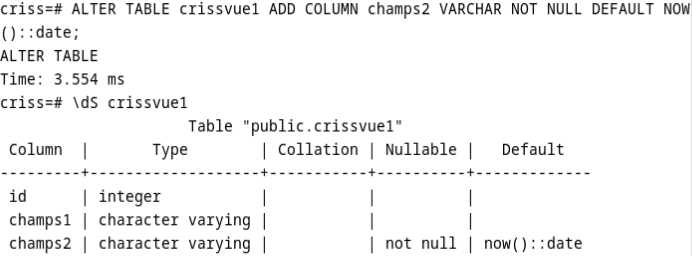


Permet de restituée la vue

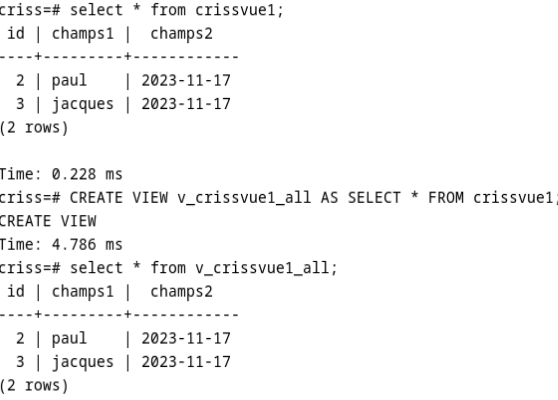
Si nous supprimons une ligne de la table d’origine la conséquence sur la vue et prend automatiquement le changement Mais attention si on fait évoluer la structure de la table il faut recréer la vue !!!



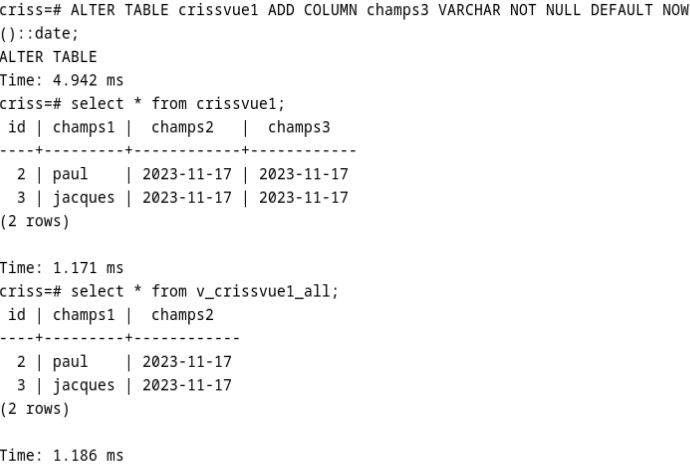
+- Nous faisons évoluer note table ajouter un champs2



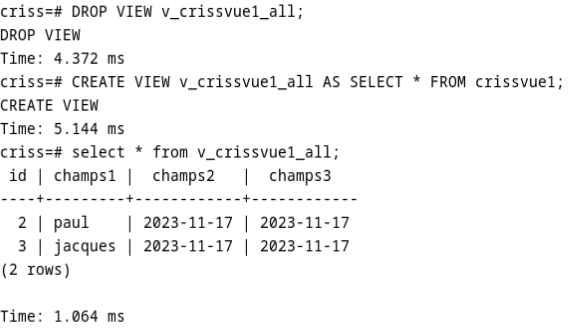
Nous recréons une vue constituer avec un select \* c’est important quand on va refaire évoluer ajouter un champs3



Ajouter un champs3 et utiliser la dernière vue et ici elle n’est pas à jour.



Donc obliger de supprimer la vue et la recréer



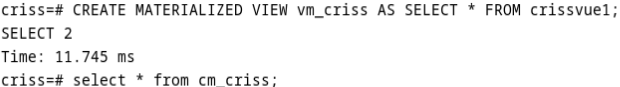
+- La vue matérialisée : (cela permet de faire des backups à un instant)

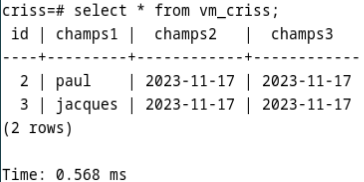
+- vue matérialisée = copie de la table (data) à un instant donné

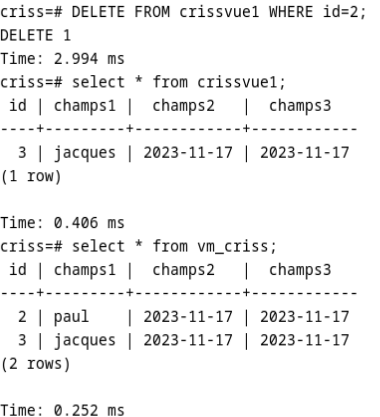
+- création de la vue matérialisée

C’est une copie d’une table en parlant des datas et structure à un instant donné.

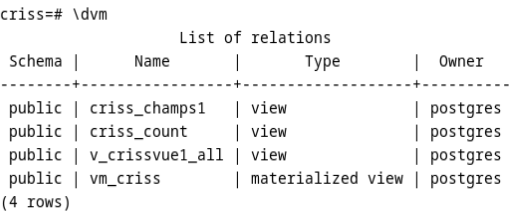
Donc si on supprime une donnée la vue matérialisée ne pourra pas se mettre à jour.







\dmv ou \dm



**15- POSTGRESQL: Droits : Rôles et Users**

+- gestion des droits : databases, schémas, tables, colonnes …

Il se gère principalement par le pg\_hba ou au sein de l’instance …

+- en postgresql : users = rôles (pas avec oracle) users Paul , rôle développeur

+- un user/role existe pour un cluster PG (instance)

+- lien avec le pg\_hba pour les autorisation de connexions à la socket



+- possibilité d’héritage = principe de roles (similaire à Oracle)

+- user = login + mdp + droits

Listes les users/roles : \du

Créer un ustilisateur : create user

Options :

+- superutilisateur : fait ce qu’il veut

+- createdb : autorise la création de database (nocreatedb)

+- createrole : autorise la création d’autres rôles/users (nocreaterol) …

+- inherit role : héritage d’un autre rôle en mentionnant son nom

+- login : autorise la connexion à l’instance

+- replication : spécifique

+- connection limit : nombre maximum de connexions concurrentes

+- password / encrypted password : définition du mot de passe entre simples quotes

+- valid until : définir une date de péremption/validité

+- admin : niveau admin (DBA oracle)

Exemple d’utilisateurs de niveaux différents

DROP USER toto ; #supprime l’utilisateur toto

ALTER USER tutu CREATEDB ; #modifie les accès

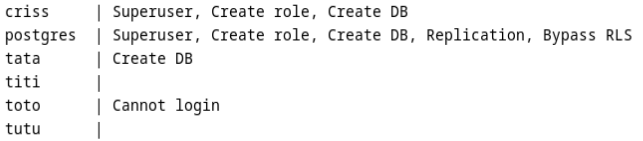
CREATE USER toto NOLOGIN NOCREATEDB;

CREATE USER tutu WITH ENCRYPTED PASSWORD ‘password’ LOGIN NOCREATEDB;

CREATE ROLE titi LOGIN NOCREATEDB;

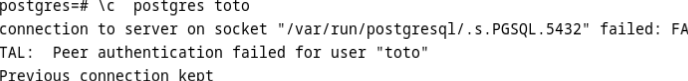
CREATE USER tata LOGIN CREATDB;

\du



Connexion par port: il faut compléter le pg\_hba.config

postgres@debian:/home/debian$ psql -h 127.0.0.1 -U tutu -d postgres



**16- POSTGRESQL: Droits : grants et databases**

La gestion des droits spécifiques des databases.

+- important pour du mutualisé : sécurité et isolation

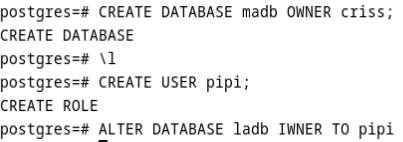
+- instance peu de limite en nb de DB

<https://www.endpoint.com/blog/2008/11/10/10000-databases-on-postgresql-cluster>

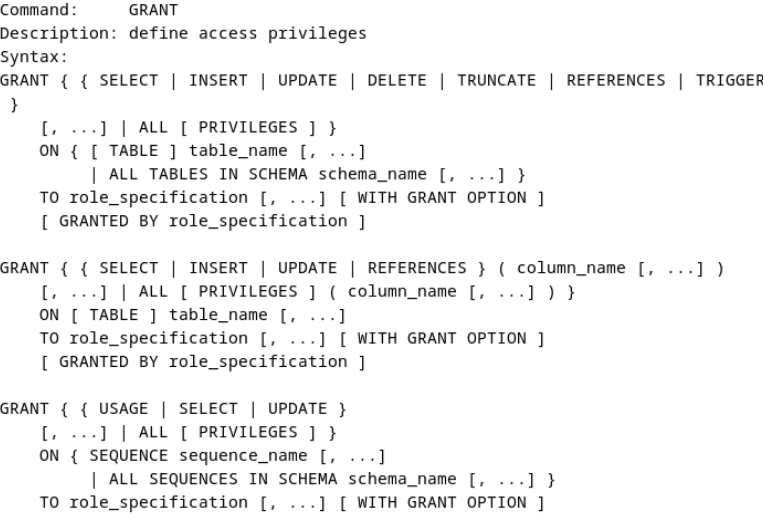
Rq : limite du filesystem

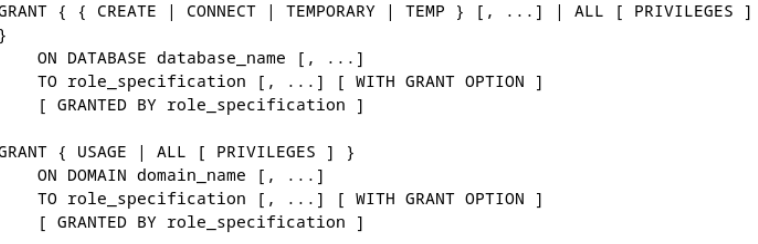
+- gestion des droits via la clause GRANT

Crée une database madb avec le propriétaire criss puis nous allons changer le propriétaire par celui que l’on a créée



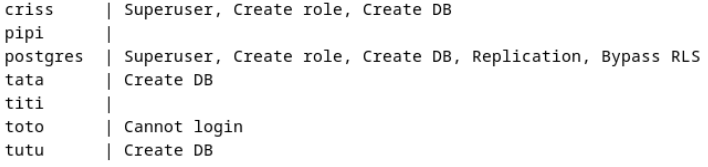
Affectation des droits GRANT (sur database, table , séquence, …)





Le CREATE permet uniquement de créer des schémas dans database

On se sert que sur le GRANT DATABASE



Crée la database





Modifie les droits de propriétés sur cette database

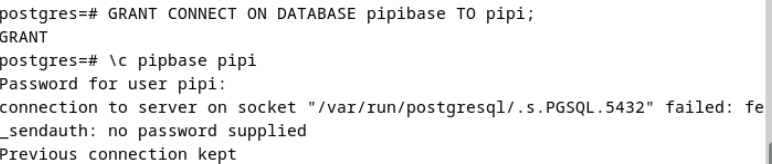




+-révocation des droits pour tous les users sauf le owner (droit juste pour utilisateur et postgres c’est tout)



+- GRANT ouvrir les droits de connexion à la base à toto



+-création utilisateur

Create user didi login createdb;

Create user dodo login created;

+- creation d’une database en tant que user postgres

Create database mondb

* Peer : signifie qu'il fera confiance à l'identité (authenticité) de l'utilisateur UNIX. Donc ne demande pas de mot de passe.
* md5 : signifie qu'il demandera toujours un mot de passe et le validera après le hachage avec MD5.
* Trust : signifie qu'il ne demandera jamais de mot de passe et fera toujours confiance à n'importe quelle connexion.

Bon nous avons un problème entre peer et md5

RESOLUTION CREER DES UTILISATEUR AVEC PASSWORD :

CREATE USER tutu WITH ENCRYPTED PASSWORD ‘tutu’ LOGIN CREATEDB;

+- ne peut pas créer de schéma

CREATE SCHEMA myschema ;

+-autorisation de création des schémas :

GRANT CREATE ON DATABASE maDB TO toto;

Attention si un schema existe il faut gérer les droits sur le schéma (REVOKE ….)

On crée ce qu’on veut en public puis on va cloisonner on va créer un schéma et créer des zone sécuriser et ensuite on crée des tables à l’intérieur.

**17- POSTGRESQL: les Schémas**

+- schémas = vues des objects de la base de données. (les droits à travers les Schémas)

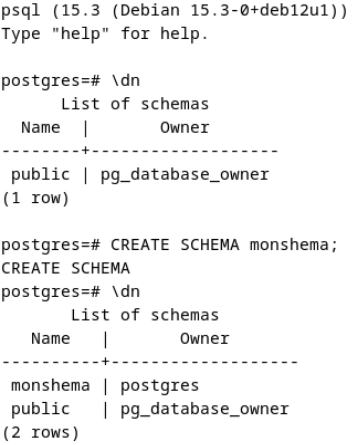
+- objectif : faciliter la gestion des droits au sein d’une base de données.

+- schéma par défaut = public

C’est une sorte de conteneur qui va faciliter les droits au niveau au sein d’une base de données

Par défaut nous avons le Schéma : Public

Création d’un schéma



Autoriser la création de schéma pour un utilisateur :

GRANT CREATE DATABASE maBase TO titi;

Supprimer les droits de création de Schéma d’un utilisateur :

REVOKE CREATE DATABASE maBase TO titi;

Tables et Schéma:

+-Comment créer une table dans un schéma ?

+- deux méthodes :

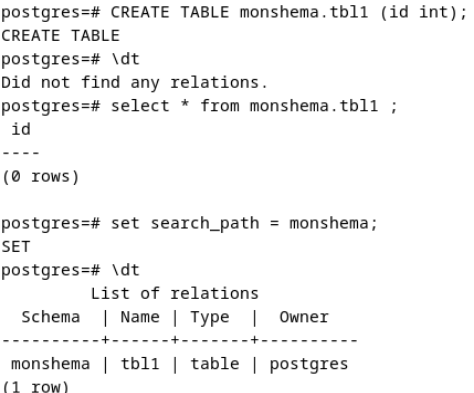
+- search\_path

+- préfix <nom\_schéma>.<nom\_table>

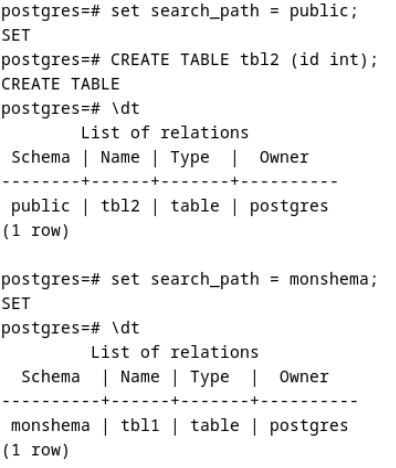
1. > CREATE TABLE monschema.tbl1 (id int); #création d’une table

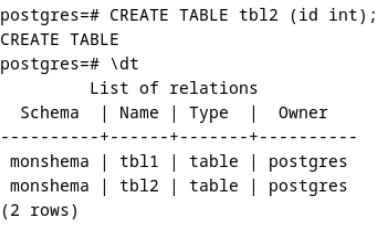
\dt pour visualiser la table ne fonctionne pas il faut faire un search\_path = monshema

Mais pour revenir il faut se mette dans son schéma public set search\_path = public ;



Pour créer une table mais sans préfixage on se remet dans le schéma public





Nous pouvons remarquer le cloisonnement des tables avec les schémas.

Savoir où l’on est > SHOW SEARCH\_PATH;

Comment avec un \dt savoir les tables même si l’on n’est pas dans le bon schéma :

ALTER USER criss SET SEARCH\_PATH = monshema;

À la connexion par défaut on sera connecté à monshema

+-2 permissions

+- USAGE : lecture dans un schéma

+- CREATE : création de tables dans un schéma

+- grant

REVOKE ALL ON SCHEMA monshema FROM PUBLIC;

\c mabase toto

CREATE TABLE monshema.matable (id int);

GRANT CREATE ON SCHEMA monshema TO toto;

CREATE TABLE monshema.matable (id int);

Ce qu’on souhaite c’est que le user toto ne puisse pas se connecter à monshema ?

Pour tout le monde:

REVOKE ALL ON SCHEMA monshema FROM PUBLIC;

\c mabase toto #on se connecte à mabase avec toto

Set search\_path = monshema ;

SET

Show search\_path ; #on est bien connecté à monshema

Select \* from monshema.tbl1;

Permission refused

GRANT USAGE SCHEMA monshema TO toto; #permet la lecture des éléments dans shema.

Set search\_path = monshema ;

SET

\dt #liste les relations

GRANT CREATE SCHEMA monshema TO toto;

CREATE TABLE monshema.tbl3 (id int);

\*\*\*\*ON PEUT CREER DES DROITS ASSEZ FIN AVEC LES SCHEMA SANS GERER LES DROITS UN A UN\*\*\*\*

**18- POSTGRESQL: Droits : Tables et Colonnes**

+-Limitation pour les tables :

+- SELECT : sélection

+-INSERT : insertion de lignes

+-UPDATE : mise à jour de lignes

+-DELETE : suppression de lignes

+-TRUNCATE : vider la table

+-REFERENCES : utilisation de la table comme clef (clé étrangère …)

+-TRIGGER : mise en place de déclencheurs

+-ALL

+- Syntax

GRANT ALL ON TABLE tbl1 TO <role\_ou\_public>

REVOKE ALL ON TABLE tbl1 FROM <role\_ou\_public>

--------------------------------------------

CREATE TABLE tbl1 (id int, champs1 varchar);

INSERT INTO tbl1 VALUES (1, ‘hello’);

INSERT INTO tbl1 VALUES (2, ‘world’);

REVOKE ALL ON TABLE tbl1 FROM toto;

\c user

--------------------------------------------

>\du #utilisateur

>\l #database

>\c criss mabase

CREATE TABLE tbl1 (id int, champs1 varchar);

\dt #on voit la table créée

INSERT INTO tbl1 VALUES (1, ‘hello’);

INSERT INTO tbl1 VALUES (2, ‘world’);

REVOKE ALL ON TABLE tbl1 FROM toto;

\c mabase toto #

GRANT INSERT ON TABLE tbl1 TO toto ;

!!Mais pas le droit d’un select

+-Limitation pour les colonnes/champs de tables

+- SELECT

+-INSERT

+-UPDATE

+-REFERENCES

\c mabase toto

SELECT \* FROM tbl1 ;

\c criss mabase

REVOKE ALL ON TABLE tbl1 FROM toto;

GRANT SELECT(champs1) ON TABLE tbl1 to toto;

\c mabase toto

SELECT \* FROM tbl1; #la requete veux retourber l’intégralité des champs

SELECT champs1 FROM tbl1;#plus pécis

\*\*\*Remq: partir d’un revoke all et donner les droits petit à petit et de créer les table depuis l’utilisateur et non avec postgres \*\*\*\*

**19- POSTGRESQL:Logs : DDL, formats**

+-Ne pas confondre avec les logfiles (logs de transaction – WAL)

+-répertoire : $PG\_DATA/pg\_log/ (VAR/LOG/PG\_SQL

+- postgresql = decouplage entre la collecte du log et son écriture

+- processus = logging collector

+-paramétrage dans le /etc/postgresql/../../postgresql.conf

+-paramètres log\_destination : type de logs

+- stderr

+- csvlog

+- syslog

+- eventlog

+-log\_directory = localisation

+- log\_filename = format des noms des fichiers (standard strftime)

+- attention : log\_truncate écrase les fichiers si ON

Less /var/log/postgresql/postgresql-15-main.log

Nano /etc/postgresql/15/maim/postgresql.conf

Log\_destination = csvlog (logging collector to be on

Logging\_collector = pn

Lod\_directory = /tmp

Log\_filename = criss

Log\_file\_mode = 0660

…

Quesqu’on va loger

+-information contenue dans les logs = logs\_statement

+- attention aux performances

+- none : erreur uniquement

+- ddl : non + erreur

+- mod : ddl + changements de datas (insert, update, delete)

+- all : mod + select

+- log\_line\_prefix = format des lignes de logs #mise en page

Log\_statement= all ;

Psql> select \* from mabase

Et voici les logs de la requête

Ls -larth /tmp

Less /tmp/dernier\_fichier.csv

+- logs concernant d’autres éléments : log des connexions

+- log\_checkpoints = on

+- log\_connectionx = on

+- log\_disconnections = on

+- log\_duration = on

Logs – Debug

+- logguer peu pour la performance

+- ponctuelle pour debugger augmenter les logs

+-ajout en fonction du besoin

+-pour une database :

ALTER DATABASE madb SET log\_statement = ‘none’ #madb plus de log sur une database.

Modification pour debugger mettre à all

ALTER DATABASE madb SET log\_statement = ‘all’

+-pour un user:

ALTER USER madb IN DATANASE criss SET log\_statement = ‘all

+- logger requêtes lentes:

Log\_min\_duration = -1

Rq: -1 = rien sinon en millisecondes / indépendant de log\_statement

Ce qui permet de par exemple 1000ms donc 1sc toutes les requêtes qui dépassent seront logguer.

**20- POSTGRESQL: COPY : export et import personnalisés**

COPY – réaliser des exports et se passe de INSERT INTO

+- /h COPY #aide

+- création d’une table

CREATE TABLE tbl1 (id int, champsl varchar);

INSERT INTO tbl1 SELECT \*, ‘l’ FROM generate\_series(1, 200000);

+- export vers un fichier texte

COPY tbl1 TO ‘/tmp/export\_tbl.txt’

+- export en csv

COPY tbl1 TO ‘ /tmp/export\_tbl.txt’ CSV HEADER;

Requête personnalisé

+-exporter une requête précise

COPY (SELECT COUNT(\*) FROM tbl1) TO ‘/tmp/export\_tbl1.txt’ ;

txt avec le nombre de ligne.

+- export vers une command shell

COPY tbl1 TO PROGRAM ‘wc -l > /tmp/shell\_count.txt’;

+- import de données – ex : curl ‘interroger une api pour collecter des datas’

COPY tbl1 FROM PROGRAM ‘curl <http://exemple.com>’

**21- POSTGRESQL: Sauvegarde : pg\_dump**

+- important mais cela reste une image (on ne peut pas revenir dans le temps)

+- complémentaire à l’archive log

+- pad de méthode unique => à adapter à sa situation

+- ne copie pas les users et autres spécifiques à l’instance

Ne copie pas les users donc juste les tables champs séquences.

+- deux outils (complémentaire) :

+- pg\_dump : personnalisation au maximum

+- pg\_dumpall : backup intégral de l’instance (y compris users …)

Beaucoup d’options (une partie identique à psql) :

+- -h : hostname (eh oui c’est pas l’aide)

+- -d : database

+- -p : port

+- -U : user

+- -f : fichier de sortie

+- -F : précision du format de sortie

+- c : custom (binaire)

+- d : directory (permet de parallélisation)

+- t : tar

+- p : plain text

+- -j : parallélisation

Liens

<https://xavki.blog/postgresql-tutoriaux-francais/>

support postgre (créations d’outils comme psql)

<https://www.dalibo.com/>

<https://www.google.com/imgres?imgurl=https://blog.anayrat.info/img/2014/PGArchOverview.png&imgrefurl=https://blog.anayrat.info/2014/12/22/introduction-%25C3%25A0-postgres-part1/&h=756&w=1100&tbnid=34cln6ja2t-JHM&tbnh=186&tbnw=271&usg=AI4_-kTUaXiRhyoIDpqit1vRSaJEE_STFA&vet=1&docid=kkfKAnx7yPHtzM>

<https://www.postgresql.org/docs/current/reference.html>

<https://www.postgresql.org/docs/15/app-dropuser.html>