PostgreSQL

DBA Infrastructure

Simon ELBAZ (selbaz@linagora.com)
Akrem JELASSI (ajelassi@linagora.com)

27 février 2023

Présentation

Histoire brève de PostgreSQL - POSTGRES

- PostgreSQL est dérivé du projet POSTGRES amorcé par l'Université de Californie à Berkeley
- POSTGRES est né en 1986 et a été financé par :
  - le DARPA (Defense Advanced Research Projects Agency)
  - l'ARO (Army Research Office)
  - le NSF (National Science Foundation)
  - ESL Inc.
- POSTGRES a été utilisé par différents projets universitaires.
- Illustra Information Technologies qui a fusionné avec Informix, rachetée par IBM commercialise le code dans les années 90

```
https://www.postgresql.org/docs/15/history.html
```

Postgres95

- En 1993, la communauté d'utilisateurs double de taille.
- La maintenance du projet devient très chronophage pour les équipes de recherche universitaires qui décident de mettre fin au projet avec la publication finale de la version 4.2
- En 1994, Andrew Yu et Jolly Chen ajoutent un interpréteur SQL à POSTGRES
- Le projet change de nom pour devenir Postgres95, l'héritier open source de POSTGRES
- Postgres95 est entièrement écrit en ANSI C et son code est réduit de 25%.
- Postgres95 v1.0 est 30-50% plus rapide que POSTGRES

PostgreSQL

- Le langage PostQUEL est définitivement abandonné au profit de SQL
- Le client psql est développé. Il utilise la librairie GNU readline
- Le support des grands objets (Large Objects) est mis en place
- En 1996, Postgres95 devient PostgreSQL car il semble important que l'année ne figure pas dans le nom
- Il est aussi possible de l'appeler Postgres (en référence à son ancêtre)

Exercice - Installer PostgreSQL 15 sur les serveurs de formation

- Installer le serveur PostgreSQL sur les serveurs **hqpg-0x** et **hqpg-0x-repl**
- Les serveurs ont pour OS Rocky Linux version 8

```
https://www.postgresql.org/download/linux/redhat/
https://rockylinux.org/
```

Ecosystème

Présentation de l'écosystème PostgreSQL

- La communauté PostgreSQL met à disposition un nombre assez important d'outils pour faciliter la vie des utilisateurs et des administrateurs
- Les outils suivants vont être présentées durant cette formation :
  - POWA
  - pgBadger
  - pgAdmin4

POWA

- PoWA signifie "PostgreSQL Workload Analyzer"
- Il supporte PostgreSQL 9.4+
- PoWA permet de collecter, agréger et purger des statistiques sur plusieurs instances PostgreSQL depuis plusieurs extensions statistiques
- En fonction des besoins de l'utilisateur, PoWA fonctionne avec 2 modes :
  - en utilisant le **background worker**. Ce fonctionnement est adapté aux environnements mono instances. Il nécessite un redémarrage de la base.
  - en utilisant le **PoWA collector**. Ne nécessite pas de redémarrage, collecte les informations depuis plusieurs bases, le standby inclus

```
https://powa.readthedocs.io/en/latest/
```

POWA - modules de statistiques

- Les modules de statistiques supportés par PoWA sont :
  - **pg_stat_statements** fournit des informations sur les requêtes en cours d'exécution
  - **pg_qualstats** fournit des informations sur les prédicats ou les clauses **where**
  - **pg_stat_kcache** fournit des informations sur le cache de l'OS
  - **pg_wait_sampling** fournit des informations sur les événements en attente
  - **pg_track_settings** suit et garde une trace des modifications du paramétrage du serveur PostgreSQL
- Il est également possible d'ajouter le support de HypoPG
- HypoPG permet de tester des index sans avoir à les déployer réellement en base

POWA - les composants logiciels

- **PoWA-archivist** : extension PostgreSQL de collecte des statistiques
- **PoWA-collector** : démon de collecte des informations des instances distantes de PostgreSQL
- **PoWA-web** : interface web de PoWA

Remarque : Il est préférable de ne pas déployer PoWA sur un environnement de production car les performances peuvent être négativement impactés

POWA - Exercice

- Déployer PoWA en suivant le lien :
  https ://powa.readthedocs.io/en/latest/quickstart.html

```
[root@localhost ~]# dnf install postgresql15-contrib
[root@localhost ~]# dnf install powa_15 pg_qualstats_15 pg_stat_kcache_15 hypopg_15
```

Modifier la ligne suivante dans **postgresql.conf** :

```
shared_preload_libraries = 'pg_stat_statements,powa,pg_stat_kcache,pg_qualstats,hypopg'
```

Puis redémarrer le service PostgreSQL :

```
[root@localhost ~]# systemctl restart postgresql-15.service
```

POWA - Exercice - Installation des extensions

```
[root@localhost ~]# su - postgres
Dernière connexion : mercredi 22 février 2023 à 13:16:44 EST sur pts/1
[postgres@localhost ~]$ psql
psql (15.2)
Saisissez « help » pour l'aide.

postgres=# CREATE DATABASE powa;
CREATE DATABASE
postgres=# \c powa
Vous êtes maintenant connecté à la base de données « powa » en tant qu'utilisateur « postgres ».
powa=# CREATE EXTENSION pg_stat_statements;
CREATE EXTENSION
powa=# CREATE EXTENSION btree_gist;
CREATE EXTENSION
powa=# CREATE EXTENSION powa;
CREATE EXTENSION
postgres=# CREATE EXTENSION hypopg;
CREATE EXTENSION
postgres=# CREATE ROLE powa SUPERUSER LOGIN PASSWORD 'astrongpassword' ;
CREATE ROLE
```

POWA - Exercice - Installation de l'interface web

Installer le paquet powa_15-web :

```
[root@localhost ~]# dnf install powa_15-web
```

Créer le fichier **/etc/powa-web.conf** :

```
servers={
  'main': {
    'host': 'localhost',
    'port': '5432',
    'database': 'powa'
  }
}
cookie_secret="linagora"
```

```
[root@localhost ~]# powa-web
[I 230222 18:01:27 powa-web:13] Starting powa-web on http://0.0.0.0:8888/
```

Pour accéder au serveur web depuis le PC, merci de créer le tunnel SSH équivalent avec PuttY :

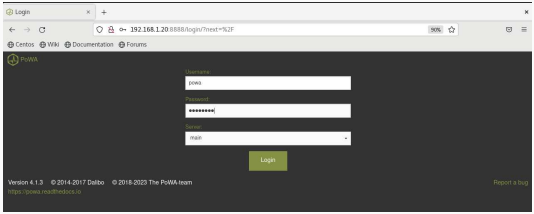
```
# ssh -L 8888:10.10.10.28:8888 hqpg-sandbox
```

Puis entrer l'URL :

```
http://localhost:8888/
```

POWA - Login

Fenêtre de login



POWA - Ajout d'une base de données

POWA - Visualisation des métriques

pgBadger

- pgBadger est un analyseur de logs rapide
- Il accepte un ou plusieurs fichiers de logs
- Il traite également les données fournies par l'entrée standard
- Il est capable de traiter un fichier de logs à distance avec un accès SSH sans mot de passe
- pgBadger s'appuie sur les protocoles http et ftp pour traiter les fichiers de log distants
- Il peut traiter les logs de pgBouncer
- Traitement incrémental des logs
- Génère des rapports au format HTML

```
https://pgbadger.darold.net/documentation.html
```

Métriques remontées par pgBadger

Les métriques suivantes sont remontées par pgBadger :

- Statistiques globales
- Requêtes les plus fréquemment en attente
- Requêtes ayant attendu le plus longtemps
- Requêtes ayant généré le plus fréquemment des fichiers temporaires
- Requêtes ayant généré les plus grands fichiers temporaires
- Requêtes les plus lentes
- Requêtes ayant duré le plus longtemps
- Requêtes les plus fréquentes

Métriques remontées par pgBadger

- Erreurs les plus fréquentes
- Histogramme du temps pris par les requêtes
- Histogramme du temps pris par les sessions
- Utilisateurs des requêtes les plus fréquentes
- Applications des requêtes les plus fréquentes
- Requêtes générant le plus d'annulation
- Requêtes les plus annulées
- Requêtes de type prepare/bind durant le plus longtemps

Exercice - Installer pgBadger

Installer pgBadger en appliquant les commandes suivantes en tant que **root** :

```
[root@localhost ~]# dnf install https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm https://dl.fedor
[root@localhost ~]# dnf install perl-Text-CSV_XS
[root@localhost ~]# dnf install pgbadger
```

Exercice - Paramétrage préalable de PostgreSQL

Vérifier que la ligne suivante est présente dans postgresql.conf :

```
log_min_duration_statement = 0
```

Les lignes du journal doivent avoir un minimum d'information :

```
log_line_prefix = '%t [%p]: '
log_checkpoints = on
log_connections = on
log_disconnections = on
log_lock_waits = on
log_temp_files = 0
log_autovacuum_min_duration = 0
log_error_verbosity = default
```

Remarques :

- Ne pas activer l'option log_statement car son format ne peut être traité par pgBadger
- pgBadger est restreint aux messages de log en anglais. Il ne peut traiter des logs en langue française.

Exercice - Utilisation de pgBadger

Pour tester l'installation, on peut lancer la commande suivante en tant que **postgres** :

```
# pgbadger --help
# pgbadger /var/lib/pgsql/15/data/log/postgresql-Wed.log -o /var/www/html/pgbadger.html
```

Ce rapport peut ensuite être visualisé avec un navigateur web.

pgAdmin4 - Présentation

- pgAdmin4 est un outil d'administration de base de données PostgreSQL
- Il est multi-plateforme (Microsoft Windows, Linux, MacOS)
- Il a une documentation fournie et détaillée
- Il possède 2 modes de déploiement :
  - le mode desktop
  - le mode serveur, multi-utilisateurs avec un accès web
- Il intègre un éditeur de requêtes SQL avec coloration syntaxique
- Les données sont affichées rapidement dans une grille interactive
- Le plan d'exécution de la requête est affichée de manière ergonomique

```
https://www.pgadmin.org/
```

pgAdmin4 - Présentation

- Il a un menu dédié à la gestion efficace des ACLs
- Débugger intégré du langage pl-pgsql
- Outil de diff des schémas
- Editeur de graphe ERD (Entity Relation Diagram) pour la conception et la documentation
- Outils de maintenance
  - Gestion de l'autovacuum
  - Tableau de bord de supervision
  - Sauvegarde, restauration, vacuum et analyze à la demande
  - Déploiement de job en SQL/shell/batch grâce un agent de programmation
- Un grand nombre de jeu de caractères supportés
- Gestion des objets PostgreSQL (table, types, vues matérialisées, ...)

Echantillon d'objets gérés par pgAdmin4

- Contraintes d'exclusion
- Extensions
- Recherche pleine de texte - Full Text Search (FTS)
- Enveloppeurs de données externes - Foreign Data Wrappers
- Politiques de sécurité de lignes (RLS)

Exercice - Installation de pgAdmin4

- Le lien ci-dessous décrit le déploiement web d'un serveur pgAdmin4

```
[linagora@localhost ~]$ sudo -i
[root@localhost ~]# rpm --import https://www.pgadmin.org/static/packages_pgadmin_org.pub
[root@localhost ~]# rpm -i https://ftp.postgresql.org/pub/pgadmin/pgadmin4/yum/pgadmin4-redhat-repo-2-1.noarch.rp
[root@localhost ~]# dnf install -y policycoreutils-python-utils
```

https://www.howtoforge.com/how-to-install-pgadmin-4-on-rocky-linux/

Exercice - Activation du mode web

```
[root@localhost ~]# /usr/pgadmin4/bin/setup-web.sh
Setting up pgAdmin 4 in web mode on a Redhat based platform...
...
Email address: selbaz@linagora.com
Password:
Retype password:
You can now start using pgAdmin 4 in web mode at http://127.0.0.1/pgadmin4
```

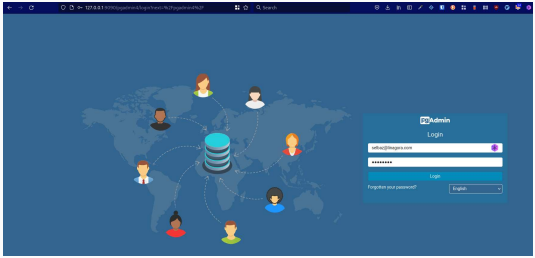
Pour accéder au serveur web depuis le PC, merci de créer le tunnel SSH équivalent avec PuttY :

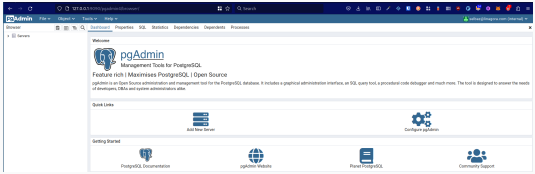
```
# ssh -L 9090:10.10.10.28:80 hqpg-sandbox
```

Puis entrer l'URL :

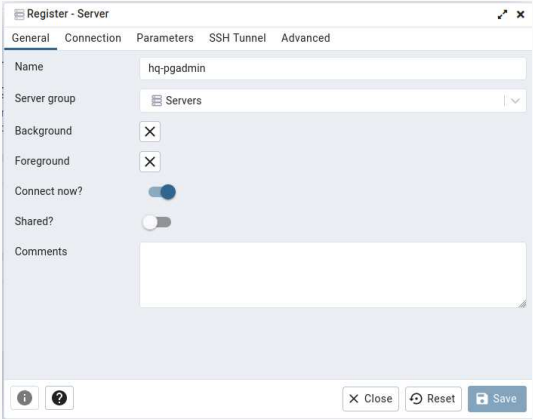
```
http://127.0.0.1:9090/pgadmin4
```

Exercice - pgadmin4 screenshots





Exercice - pgadmin4 screenshots



Le modèle template0

- Le modèle template0 est identique au modèle template1
- A la différence de template1, il ne doit jamais être modifié car il représente la base de données à l'état initial
- Ce modèle sert de base pour la création de base de données à partir d'un dump généré par **pg_dump**
- Une autre raison de partir du modèle template0 pour créer une nouvelle base de données, est le changement de jeu de caractères
- template0 ne contient pas de choix de jeu de caractères par défaut. Ce qui n'est pas le cas de template1.

Création d'une base de données à partir d'un modèle

Pour créer une base de données à partir du modèle template0, les 2 commandes suivantes sont utilisables au choix :

```
CREATE DATABASE dbname TEMPLATE template0;
```

ou depuis le shell :

```
createdb -T template0 dbname
```

Généralisation des modèles

- La commande **CREATE DATABASE** permet de crér une nouvelle base de données en utilisant une autre comme un modèle
- Cependant, cet usage est fortement déconseillé
- En effet, pendant la copie de la base source, aucune session ne peut se connecter sur la base source

La table pg_database

- La table pg_database inclut 2 colonnes intéressantes :
  - **datistemplate** : la base peut servir de modèle. Sinon seuls les super-utilisateurs et le propriétaire de la base peuvent s'en servir comme modèle
  - **datallowconn** : aucune nouvelle session ne peut être créée.

La base de données postgres

- Cette base est une simple copie de template1
- Elle sert de base de connexion par défaut pour les utilisateurs et les applications
- Elle peut être détruite et recopiée depuis template1

Haute disponibilité (HA)

Solutions de HA

- Il existe plusieurs solutions de HA.
- Certaines solutions intègrent nativement le partitionnement de données.
- Citus fait partie des solutions HA avec partitionnement de données
- L'objectif de cette formation est de permettre au client d'administrer ses bases de données avec le choix des outils qu'il a réalisé
- Le choix réalisé est **repmgr**

```
https://docs.citusdata.com/en/v11.2/get_started/what_is_citus.html
https://repmgr.org/
```

Présentation de repmgr

- repmgr est une solution de gestion de :
  - la réplication
  - le switchover
  - et le failover PostgreSQL
- Elle supporte les versions PostgreSQL de 9.4 à 15
- Elle est portée par l'entreprise EDB

Caractéristiques avancées de repmgr

- repmgr intègre les nouveautés introduites par PostgreSQL 9.3 :
  - la réplication en cascade
  - la commutation de la ligne de temps (timeline switching)
  - et la sauvegarde de base via la réplication
- Il est disponible sous la licence GPLv3
- La dernière version disponible pendant la rédaction de ce document est la v5.3.3

Notions de cluster repmgr

- Les notions suivantes sont présentes dans un cluster repmgr :
  - failover : Dans le cas de l'échec d'un noeud, le standby prend le relais et devient noeud primaire
  - switchover : Dans le cas d'une opération de maintenance, le DBA choisit de réaliser une basculer vers le noeud standby
  - fencing : Dans le cas d'une bascule failover vers le standby, il est important que l'ancien serveur primaire ne revienne pas en ligne et cause un split-brain. L'opération de fincing consiste à isoler l'ancien noeud primaire
  - witness server (serveur témoin) : Le serveur témoin sert à mettre en place un quorum pour éviter le split brain. Il ne fait pas partie du process de réplication.
  - En cas de coupure réseau entre les différents noeuds, ceux qui voient le serveur témoin entrent dans un process de vote pour élire un nouveau primary

Eléments de la solution repmgr

- Les principaux éléments de la solution repmgr sont :
  - le démon repmgrd : Il surveille l'état de la replication, réalise le failover en cas de défaillance du serveur primaire et envoie des notifications d'événements
  - la commande en ligne repmgr. Elle permet d'administrer le cluster PostgreSQL

Exercice - Déploiement de repmgr

- repmgr est inclus dans les dépôts Yum Repository de PostgreSQL
- Sur les serveurs hqpg-0x et hqpg-0x-repl, lancer les commandes suivantes :

```
# dnf list | grep repmgr
...
repmgr_15.x86_64                                    5.3.3-1.rhel8
repmgr_15-devel.x86_64                              5.3.3-1.rhel8
repmgr_15-llvmjit.x86_64                            5.3.3-1.rhel8
# dnf install repmgr_15
# dnf install rsync
```

- Paramétrer le noeud primaire en suivant le lien :
  https ://repmgr.org/docs/current/quickstart-postgresql-configuration.html
- **Remarques** : Certaines valeurs de paramètres ont changé entre les différentes versions de PostgreSQL 9.6+

Exercice - Création de la base de données repmgr

- Au départ, le serveur hqpg-0x est choisi comme primaire
- Sur le serveurs hqpg-0x, lancer les commandes suivantes :

```
# su - postgres
# createuser -s repmgr
# createdb repmgr -O repmgr
[postgres@localhost ~]$ psql
psql (15.2)
Saisissez « help » pour l'aide.

postgres=# ALTER USER repmgr SET search_path TO repmgr, "$user", public;
ALTER ROLE
```

Partitionnement des données

Présentation des indexes B-Tree

- Un index de type B-Tree est un arbre composé :
  - d'une racine
  - de noeuds intermédiaires
  - de feuilles
- Les feuilles correspondent aux données dans les tables
- Chaque noeud intermédiaire porte un nombre de **clefs** maximum et minimum

```
https://en.wikipedia.org/wiki/B-tree
https://www.postgresql.org/docs/15/btree.html
https://www.postgresql.org/docs/15/btree-implementation.html
```

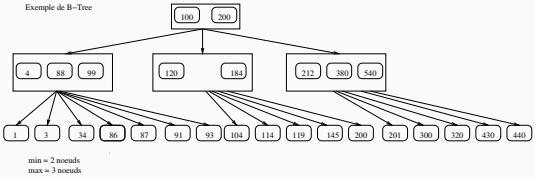
Présentation des indexes B-Tree

- Les **clés** définissent les bornes minimum et maximum des valeurs portées par les noeuds enfants
- Lorsqu'un noeud atteint son nombre maximum d'enfants, une scission ("split") a lieu
- Lorsqu'un noeud atteint son nombre minimum d'enfants, une fusion ("merge") a lieu

L'objectif des indexes B-Tree

- L'objectif de l'index B-Tree est de stocker en mémoire une partie d'un index très volumineux
- Cet objectif est atteint par le fait que les noeuds intermédiaires indexent des plages de données

Schéma explicatif des indexes B-Tree



Tables partitionnées

- PostgreSQL supporte le partitionnement de données.
- Le partitionnement PostgreSQL consiste à découper une table logique en plusieurs petits morceaux physiques

```
https://www.postgresql.org/docs/15/ddl-partitioning.html
```

Tables partitionnées

- Dans certaines situations, le partitionnement a plusieurs avantages :
  - lorsque les données accédées sont localisées dans une partition ou un nombre réduit de partitions. A ce moment, la partie haute des indexes (B-Tree) qui correspond à ces données est entièrement en cache.
  - lorsque les données modifiées ou accédée couvrent la majorité de la partition, il est préférable d'utiliser un scan séquentiel au lieu d'utiliser un index
  - les chargements de données massifs ou suppressions massives de données correspondant à une partition sont réalisés en ajoutant une partition ou en supprimant une partition. En cas de suppression massive, un **VACUUM** est économisé
  - les données ou partitions les moins utilisées peuvent être migrées vers des disques moins onéreux et moins rapides

A quel moment faire le choix d'une table partitionnée

- Il peut être intéressant de partitionner une table lorsque sa taille dépasse celle de la quantité de RAM du serveur
- Il existe différents types de partitionnement :
  - Partitionnement par plage (Range partitioning)
  - Partitionnement par liste (List partitioning)
  - Partitionnement par hash (Hash partitioning)

Partitionnement par plage (Range partitioning)

- La plage de valeurs est portée par une ou plusieurs colonnes
- Les valeurs peuvent être de type date ou identifiants
- Les plages n'ont pas d'intersection : elles n'ont pas de valeurs en commun
- Le min de la plage est inclus, le max est exclus

Partitionnement par liste (List partitioning)

- La plage de valeurs est définie par une liste qui indique les valeurs autorisées dans la partition

Partitionnement par hash (Hash partitioning)

- La clé de partitionnement est définie par un modulo et un reste
- La valeur est autorisée dans la partition lorsque son modulo est égale au reste de la partition

Stockage des tables partitionnées

- Une table partitionnée est virtuelle. Elle ne possède pas de stockage.
- Ce sont ses partitions qui occupent du stockage
- Chaque partitiotn récupère une partie des données de la table
- Chaque ligne est oritentée vers la partition en fonction de la valeur de la clef de partition
- Lorsque la clef de partition est mise à jour, la ligne est déplacée vers la parition qui correspond à la nouvelle clef

Sous partitions

- Une partition peut elle même être partitionnée
- Les partitions ont les mêmes colonnes que leurs parents
- Cependant, elles peuvent définir leurs propres indexes, valeurs par défaut et contraintes

```
https://www.postgresql.org/docs/15/sql-createtable.html
```

Conversion des tables en tables partitionnées

- Il n'est pas possible de transformer une table en une table partitionnée et inversement
- Il est possible d'ajouter une partition à une table partionnée
- Il est possible d'ajouter une table partitionnée à une table partionnée en tant que partition
- Il est possible de transformer une partition d'une table en une table régulière
- Ces possibilités facilitent la maintenance des tables partitionnées
- Les tables étrangères (**foreign tables**) peuvent jouer aussi le rôle de partitions

```
https://www.postgresql.org/docs/15/sql-altertable.html
https://www.postgresql.org/docs/15/ddl-foreign-data.html
```

enable_partition_pruning

- Ce paramètre autorise le planificateur de requêtes d'élaguer certaines partitions de tables de son plan d'exécution
- Valeur par défaut **on**

```
https://www.postgresql.org/docs/15/runtime-config-query.html#GUC-ENABLE-PARTITION-PRUNING
```

Exercice - Manipuler des tables partitionnées

- Suivre l'exercice paragraphe Example dans https ://www.postgresql.org/docs/15/ddl-partitioning.html
- Créer une table partitionnée
- Ajouter une sous-partition
- Ajouter un index à une partition

Maintenance des partitions - Suppression d'une partition

- Il y a 2 possibilités pour supprimer une partition d'une table :

```
DROP TABLE measurement_y2006m02;
```

- L'autre possibilité est :

```
ALTER TABLE measurement DETACH PARTITION measurement_y2006m02;
-- ou
ALTER TABLE measurement DETACH PARTITION measurement_y2006m02 CONCURRENTLY;
```

- Elle permet de réaliser un dump ou une sauvegarde de la partition avant suppression définitive
- La clause **CONCURRENTLY** permet une modification de la table parente en parallèle

Maintenance des partitions - Ajout d'une partition - ATTACH

- Comme pour la suppression, il y a 2 possibilités pour ajouter une partition à une table :

```
CREATE TABLE measurement_y2008m02 PARTITION OF measurement
FOR VALUES FROM ('2008-02-01') TO ('2008-03-01')
TABLESPACE fasttablespace;
```

- L'autre possibilité est de créer une table en dehors de la table partitionnée puis de la rattacher :

```
CREATE TABLE measurement\_y2008m02
(LIKE measurement INCLUDING DEFAULTS INCLUDING CONSTRAINTS)
TABLESPACE fasttablespace;

ALTER TABLE measurement_y2008m02 ADD CONSTRAINT y2008m02
   CHECK ( logdate >= DATE '2008-02-01' AND logdate < DATE '2008-03-01' );

\copy measurement_y2008m02 from 'measurement_y2008m02'
-- possibly some other data preparation work

ALTER TABLE measurement ATTACH PARTITION measurement_y2008m02
    FOR VALUES FROM ('2008-02-01') TO ('2008-03-01' );
```

- Noter l'utilisation du **LIKE** qui permet de dupliquer la définition de la table
- **Remarque** : Il est important d'ajouter la vérification **CHECK** avant d'attacher la partition. Cela évite un scan verrouillé en **ACCESS LOCK** sur la partition.

Maintenance des partitions - Ajout d'une partition - ATTACH

- La clause de vérification avant l'**ATTACH** est importante en cas de présence d'une partition **DEFAULT**
- Sans la clause **CHECK**, la partition qui est rattachée va être scannée en étant verrouillé afin de vérifier qu'il n'y a pas d'enregistrement susceptible d'aller en **DEFAULT**

Indexes et tables partitionnées

- Comme vu précédemment, il est possible de créer des indexes sur des tables partitionnées
- Ces indexes seront appliqués aux futures partitions
- Il y a cependant une limitation : il n'est pas possible d'utiliser **CONCURRENTLY**. Il y a donc un verrou exclusif à l'ajout d'une partition
- Pour éviter cela, on utilise **CREATE INDEX ON ONLY** sur la table partitionnée
- L'index créé de cette manière est invalide
- Il est ensuite possible d'appliquer la clause **CONCURRENTLY** sur les partitions qui seront rattachées
- Une fois l'ensemble des partitions attachées avec l'index, celui-ci est validé
- Cette technique s'applique aussi aux clauses **UNIQUE** et **PRIMARY KEY**

Exercice - Utiliser la clause CREATE INDEX ON ONLY

- Merci de réaliser l'exercice de création d'index comme indiqué dans le lien https ://www.postgresql.org/docs/15/ddl-partitioning.html#DDL-PARTITION-PRUNING
- au paragraphe 5.11.2.2 en utilisant CREATE INDEX ON ONLY

Elagage (pruning) de partitions dans le planificateur de requêtes

- Comme vu précédemment l'élagage de partitions est activé par défaut pour le planificateur de requêtes

```
SET enable_partition_pruning = on;                 -- the default
SELECT count(*) FROM measurement WHERE logdate >= DATE '2008-01-01';
```

- Sans l'élagage de partitions, la requête précédente aurait scanné toute la table measurement
- Avec l'élagage, le planificateur détermine pour chaque partition que la donnée ne s'y trouve et ne scanne pas la partition

Vérification avec l'EXPLAIN plan

```
SET enable_partition_pruning = \textbfoff;
EXPLAIN SELECT count(*) FROM measurement WHERE logdate >= DATE '2008-01-01';
                                QUERY PLAN
-------------------------------------------------------------------------------
 Aggregate  (cost=188.76..188.77 rows=1 width=8)
   ->  Append  (cost=0.00..181.05 rows=3085 width=0)
         ->  Seq Scan on measurement_y2006m02  (cost=0.00..33.12 rows=617 width=0)
               Filter: (logdate >= '2008-01-01'::date)
         ->  Seq Scan on measurement_y2006m03  (cost=0.00..33.12 rows=617 width=0)
               Filter: (logdate >= '2008-01-01'::date)
...
         ->  Seq Scan on measurement_y2007m11  (cost=0.00..33.12 rows=617 width=0)
               Filter: (logdate >= '2008-01-01'::date)
         ->  Seq Scan on measurement_y2007m12  (cost=0.00..33.12 rows=617 width=0)
               Filter: (logdate >= '2008-01-01'::date)
         ->  Seq Scan on measurement_y2008m01  (cost=0.00..33.12 rows=617 width=0)
               Filter: (logdate >= '2008-01-01'::date)
```

- On peut voir que le planificateur scanne l'ensemble des partitions de la table lorsque l'élagage est désactivé

Vérification avec l'EXPLAIN plan

```
SET enable_partition_pruning = \textbfon;
EXPLAIN SELECT count(*) FROM measurement WHERE logdate >= DATE '2008-01-01';
                                QUERY PLAN
-------------------------------------------------------------------------------------
 Aggregate  (cost=37.75..37.76 rows=1 width=8)
   ->  Seq Scan on measurement_y2008m01  (cost=0.00..33.12 rows=617 width=0)
         Filter: (logdate >= '2008-01-01'::date)
```

- On peut voir que le planificateur scanne la bonne partitions lorsque l'élagage est activé
- Ce n'est pas la présence d'index qui va guider le planificateur mais bien les contraintes de partitionnement

Optimisation

Les indexes dans PostgreSQL

- B-Tree Index - très utile pour la recherche d'une valeur ou le scan d'une plage de valeur. Utilisé également pour les expressions régulières (pattern matching)
- Hash Index - très efficace pour recherche de valeurs égales
- Generalized Search Tree (GiST) - est une catégorie d'index offrant une architecture dans laquelle plusieurs stratégies d'indexes peuvent être implémentées. En fonction de ces stratégies, des opérateurs (*operator class*) sont définies.
- Par exemple, la distribution standard de PostgreSQL inclut des opérateurs de classe GisT pour les types de données à 2 dimensions
- Les indexes GiST sont aussi capables d'optimiser les recherches de type **"voisinage le plus proche"**

```
https://www.postgresql.org/docs/15/indexes-types.html
```

Les indexes dans PostgreSQL

- Space Partitioned GiST (SP-GiST) - similar au GiST, cette classe d'index supporte des structures de données non équilibrées
- Generalized Inverted Index (GIN) - utile pour indexer des valeurs de type multi-composants comme des tableaux. et tester la présence d'un élément
- Tout comme la famille GiST, la famille GIN supporte des stratégies d'indexation personnalisables par l'utilisateur. Les opérateurs de la cette classe d'index dépendent de la stratégie d'indexation.

Les indexes dans PostgreSQL

- Block Range Index (BRIN) - ce type d'index stocke des résumés d'information pour des blocs contigus et physiques de données. Ce type d'index est bien adapté aux données corrélées avec l'ordre physique de stockage
- Tout comme la famille GiST, la famille BRIN supporte des stratégies d'indexation personnalisables par l'utilisateur. Les opérateurs de la cette classe d'index dépendent de la stratégie d'indexation.
- L'index de type BRIN contient le min et le max de chaque bloc de données linéaire

Pistes d'optimisation

- La quantité de **shared_buffers** : elle peut occuper 1/4 de la RAM disponible
- **work_mem** est la quantité de RAM allouée à chaque process de traitement d'une session
- **effective_cache_size** est la quantité de RAM que le planificateur de requête a à sa disposition. En cas de valeur importante, il va privilégier l'utilisation des indexes. Pour une valeur plus faible, il privilégie les scans séquentiels.
- **effective_io_concurrency** est le nombre d'opération I/O que le serveur PostgreSQL peut exécuter en parallle sur un disque
- activation des Huge Pages

```
https://www.postgresql.org/docs/15/runtime-config-query.html
https://www.postgresql.org/docs/15/runtime-config-resource.html#RUNTIME-CONFIG-RESOURCE-ASYNC-BEHAVIOR
https://smartsla.08000linux.com/requests/1009
```

Activation des Huge Pages du kernel

- L'utilisation des Huge Pages Kernel permet d'allouer des pages de RAM plus imortantes au process PostgreSQL
- Il minimise les appels Kernel pour l'allocation de RAM

Pour vérifier que les Huge Pages sont activées, la commande suivante peut être utilisée :

```
# cat /proc/meminfo | grep Huge
AnonHugePages:  247808 kB
HugePages_Total:  19781
HugePages_Free:  16193
HugePages_Rsvd:   5193
HugePages_Surp:    0
Hugepagesize:    2048 kB
```

https://www.kernel.org/doc/html/latest/admin-guide/mm/hugetlbpage.html

EXPLAIN PLAN

TODO WIP

```
xxxx
```

Index only scans

indexonlyscan)

TODO WIP

```
https://www.postgresql.org/docs/15/indexes-index-only-scans.html
```

La commande CLUSTER

TODO WIP

```
https://www.postgresql.org/docs/15/sql-cluster.html
```

Utilisation des statistiques par le planificateur de requêtes

TODO WIP

https://www.postgresql.org/docs/15/planner-stats.html

Statistiques accumulées pendant le service - Run-time statistics

TODO WIP

```
https://www.postgresql.org/docs/15/runtime-config-statistics.html
```

Statistiques étendues

TODO WIP

```
https://www.postgresql.org/docs/current/planner-stats.html#PLANNER-STATS-EXTENDED
https://www.postgresql.org/docs/current/multivariate-statistics-examples.html
```

Requêtes parallèles

TODO WIP

https://www.postgresql.org/docs/15/parallel-query.html

Amélioration des performances

TODO WIP

https://www.postgresql.org/docs/15/performance-tips.html

Tests de performances

TODO WIP

```
https://www.postgresql.org/docs/15/pgbench.html
```

HOT - Heap-Only Tuples

TODO WIP

https://www.postgresql.org/docs/15/storage-hot.html

Sécurisation de la base PostgreSQL

Chiffrement dans PostgreSQL

- WIP

```
https://www.postgresql.org/docs/15/encryption-options.html
```

Authentification

- WIP

```
https://www.postgresql.org/docs/15/client-authentication.html
```

Politiques de sécurité des lignes - Row Security Policies

- WIP

```
https://www.postgresql.org/docs/current/ddl-rowsecurity.html
```

Maintenance

Tâches de maintenance de la base de données

La base de données nécessite des opérations de maintenance régulière pour assurer un service optimal. Les principales tâches de maintenance sont :

- Les sauvegardes régulières qui permettront de s'en sortir en cas d'accident grave
- Les opérations de VACUUM
- Les mises à jour des statistiques
- Les mises à jour des indexes
- La gestion des logs

`https://www.postgresql.org/docs/15/maintenance.html`

Les opérations de VACUUM

Les opérations de VACUUM sont lancées automatiquement par le démon autovacuum. Certains DBAs préfèrent gérer ce process eux-même à partir de tâches de type cron. Les raisons principales pour lancer un VACUUM régulier sont :

- récupérer l'espace disque occupé pour des lignes mises à jour ou supprimées
- mises à jour des statistiques utilisées par le planificateur de requêtes (query planner)
- mises à jour du flag de visibilité pour améliorer la performance des indexes only scan (chapitre Optimisation)
- protection contre la perte de données très anciennes causée par la rotation des identifiants de transaction ou de multi-transaction

```
https://www.postgresql.org/docs/15/routine-vacuuming.html
```

Différents type de VACUUM

Il existe 2 types de VACUUM :

- le VACUUM standard
- le VACUUM FULL

VACUUM standard

- Le VACUUM standard peut être lancé en parallèle des commandes SQL (**SELECT**, **UPDATE**, **INSERT** et **DELETE**)
- la commande **ALTER TABLE** ne peut être exécutée sur une table en cours de traitement par le process
- le VACUUM génère des I/O très importantes qui diminuent les performances des autres sessions actives
- Il est possible de tempérer la charge I/O causée par le VACUUM par l'intermédiaire des paramètres : **vacuum_cost_*** (cf. 120)

VACUUM FULL

- Ce VACUUM récupère plus d'espace disque que le standard.
- Il est cependant plus lent
- Le VACUUM FULL posse un verrou de type **ACCESS EXCLUSIVE** sur la table en cours de traitement. Il empêche tout autre traitement d'accéder à la table.
- Il est recommandé de privilégier l'utilisation du VACUUM standard

Récupération de l'espace disque

- Le **DELETE** ou **UPDATE** ne supprime pas immédiatement les anciennes versions d'une ligne afin que les process ayant accès à la table ait leur version courante de la ligne jusqu'à la fin de la transaction.
- Lorsqu'une version devient obsolète et qu'elle n'est plus utilisée par une session, le VACUUM marque l'espace occupé par la ligne comme disponible pour les prochaines lignes.
- Cela permet d'éviter l'explosion de l'espace disque occupé par une table.
- L'espace disque n'est donc pas libéré : il devient disponible.
- Dans le cas où l'espace disponible se trouve en fin de page et qu'il est facile d'acquérir un verrou de type exclusif, l'espace disque est rendu à l'OS

Récupération de l'espace disque

- A l'inverse, VACUUM FULL réécrit entièrement la table de manière compacte sans espace vide. Cela réduit la taille occupée par la table mais prend beaucoup de temps.
- Durant le VACUUM FULL, l'espace disque de la table double car l'ancienne version est gardée jusqu'à la fin de l'opération

Fréquence de passage du VACUUM

- La bonne pratique est de lancer un VACUUM standard suffisamment fréquemment pour éviter de lancer le FULL
- Le démon autovacuum respecte cette philosophie et n'utilise pas le FULL
- L'objectif n'est pas de garder une taille minimale pour les tables
- L'objectif est de garder une taille des tables **stable**
- Pour appliquer le VACUUM sur une base de données entière, il est possible d'utiliser vacuumdb

Cas des tables totalement supprimées

- Lorsqu'une table est entièrement supprimée, la commande **TRUNCATE** peut être appliquée
- TRUNCATE libère automatiquement l'espace disque
- **Limitation :** le TRUNCATE ne respecte les principes du MVCC (vue locale des données pour chaque session)

Cas des tables massivement mises à jour

- Lorsqu'une table est massivement mise à jour ou supprimée, il peut être intéressant de lancer un VACUUM FULL
- Une alternative au VACUUM FULL est la commande **CLUSTER**
- Cette commande réécrit la table en disposant les lignes en suivant l'ordre de l'index d'une colonne
- Elle est décrite plus en détail dans la partie optimisation

vacuumdb

- VACUUM s'applique sur une table.
- Pour l'appliquer sur une base de données, il est possible d'utiliser **vacuumdb**

```
https://www.postgresql.org/docs/15/app-vacuumdb.html
```

Mise à jour des statistiques du planificateur de requêtes

- Le planificateur de requêtes s'appuie sur les statistiques collectées sur les tables
- La commande **ANALYZE** exécute la génération de statistiques sur une table
- L'ANALYZE est également une option de la commande VACUUM. De cette manière, les 2 opérations sont lancées en parallèle sur la table.
- L'ANALYZE est également une option du démon **autovacuum**
- Dans le cas où l'administrateur système sait que les mises à jour de la table n'affecte pas les statistiques, il est possible de gérer l'ANALYZE manuellement.
- **Les mises à jour dans les partitions de table ou les tables enfants ne provoquent d'ANALYZE automatique des tables parentes.**
- Pour cela, il est nécessaire de lancer l'ANALYZE manuellement sur les tables parentes.

`https://www.postgresql.org/docs/15/sql-vacuum.html`

Cas des tables fréquemment mises à jour

- En fonction de la répartition des valeurs de données dans une colonne, il peut être intéressant ou non de lancer la commande ANALYZE
- Pour des données ayant une plage de valeurs importante, l'ANALYZE est intéressant
- Dans le cas inverse, il n'est pas nécessaire dans le lancer
- L'ANALYZE peut être être restreint à une **colonne**. Particulièrement, celles impliquées dans les clauses WHERE avec une répartition de valeurs extrêmement irrégulière.
- En pratique, il s'avère plus efficace d'appliquer l'ANALYZE à la base de données

Qualité des statistiques

Pour augmenter l'échantillonage de l'ANALYZE, il est possible d'utiliser 2 paramètres :

- **default_statistic_target**. Par défaut, 100. La base de données sera impactée
- **ALTER TABLE SET STATISTICS**. Seule la table sera impactée.

```
https://www.postgresql.org/docs/15/runtime-config-query.html#GUC-DEFAULT-STATISTICS-TARGET
```

Mise à jour du tableau de visibilité

- La table de visibilité de chaque page indique si une page de données est visible à toutes ltransactions actives.
- Si elle a été modifiée par une session et est en attente de flush vers le disque (dirty page), elle sera traitée par le VACUUM.
- Sinon elle ne sera pas traitée par le VACUUM.
- Le module **pg_visibility** permet de visualiser les informations stockées dans la table de visibilité.
- La table de visibilité est utilisée pour les index-only scans pour éviter à l'index de rafraîchir les données qu'il embarque.

```
https://www.postgresql.org/docs/15/storage-vm.html
https://www.postgresql.org/docs/15/pgvisibility.html
```

Prévention des erreurs causées par la rotation des identifiants de transaction

- Chaque transaction (XID) a un identifiant stocké sur un entier de 32 bits
- Cela autorise 2^{32} transactions
- Au bout de ce nombre de transaction, le compteur XID repasse à 0
- Les transactions qui étaient dans le passé se retrouvent dans le futur
- Pour prévenir ce type de situation, PostgreSQL utilise un identifiant spécial **FrozenTransactionId**
- Cet identifiant est antérieur à toute valeur de XID
- C'est le rôle du VACUUM de positionner cet identifiant de transaction sur la ligne

Rayon de 2^{31} pour chaque XID

- D'après ce qui a été dit précédemment chaque XID voit :
  - au maximum 2^{31} transactions plus anciennes que lui
  - et au maximum 2^{31} transactions plus récente que lui

VACUUM agressif

- le VACUUM s'appuie sur la table de visibilité pour savoir s'il est nécessaire de traiter la table et supprimer les versions obsolètes des lignes
- Il est cependant parfois nécessaire de geler les XID et MXID d'un certain âge
- Cette opération s'appelle le VACUUM **agressif**

Age d'une ligne

- Ma compréhension de la documentation m'incite à dire que l'âge d'une ligne a pour formule :
  - si $XIDcourant > XIDligne$, $age = XIDcourant - XIDligne$
  - sinon, $age = XIDcourant - XIDligne + 2^{31}$
- Une ligne ne sera jamais plus *âgée* que $2^{31} = 2$ millions transactions
- A partir d'un certain âge paramétrable, son XID est gelé en FrozenTransactionId

vacuum_freeze_min_age

- **vacuum_freeze_min_age** indique le nombre de transactions **"vues par le XID de la ligne"** au-delà duquel VACUUM envisage de traiter une **ligne** de table pour geler son XID
- Il correspond à l'âge minimum d'une ligne pour être éligible au VACUUM agressif
- Une valeur trop basse de ce paramètre déclenche le gel potentiellement inutile d'une ligne avec le risque qu'elle soit dégelée en cas de modification
- Une valeur trop haute de ce paramètre augmente le nombre de transactions nécessaires avant que cette **ligne** de table puisse être traitée à nouveau par le VACUUM
- Valeur comprise entre 0 et 10^9. Par défaut, $50 * 10^6$ transactions.
- Valeur automatiquement cappée à **autovacuum_freeze_max_age**/2

https://www.postgresql.org/docs/15/runtime-config-client.html#GUC-VACUUM-FREEZE-MIN-AGE

Age d'une table - vacuum_freeze_table_age

- Le serveur applique un VACUUM **agressif** à la table lorsque **vacuum_freeze_table_age** - **pg_class.relfrozenxid** est positif
- Lorsque **vacuum_freeze_table_age** = 0, un VACUUM agressif est systématiquement appliqué
- **pg_class.relfrozenxid** est le niveau du XID en dessous duquel tous les XID de la table ont été gelés
- En résumé, il correspond au XID le plus vieux de la table et indique en somme "l'âge" de la table
- Valeur par défaut : $150 * 10^6$ transactions

`https://www.postgresql.org/docs/15/catalog-pg-class.html`

autovacuum et vacuum_freeze_min_age

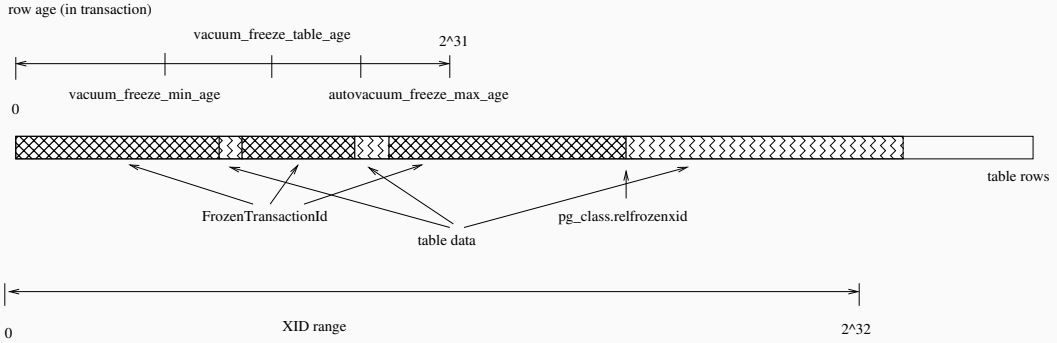
- la durée maximale qu'une table ne soit pas traitée par un VACUUM agressif est : 2^{31} − **vacuum_freeze_min_age**
- La valeur de **vacuum_freeze_min_age** est stockée au passage du dernier VACUUM agressif
- La fréquence du passage de l'**autovacuum** est environ toutes les **autovacuum_freeze_max_age** − **vacuum_freeze_min_age** transactions.
- Pour empêcher la perte de données, il est déclenché même si **autovacuum_freeze_max_age** n'est pas positionné
- La limite maximale posée par PostgreSQL est : **vacuum_freeze_table_age** $= 0.95 *$ **autovacuum_freeze_max_age**

Paramètre autovacuum_freeze_max_age

- Une valeur plus élevée de **vacuum_freeze_table_age** est inutile (car le VACUUM agressif sera déclenché par **autovacuum_freeze_max_age**)
- Une valeur moins élevée de **vacuum_freeze_table_age** déclenchera des VACUUM agressifs plus fréquents
- Le seul inconvénient d'augmenter **autovacuum_freeze_max_age** entraîne un espace disque plus important occupé par les répertoires :
  - **pg_xact** : Statut des commits
  - **pg_commit_ts** : Timestamp des commits si cette fonctionnalité est activée
- Valeur par défaut : $200 * 10^6$

https://www.postgresql.org/docs/15/runtime-config-autovacuum.html#GUC-AUTOVACUUM-FREEZE-MAX-AGE

Schéma récapitulatif



Tables de suivi des XID

- Les données sur les XIDs sont stockées dans les table pg_class et pg_database
- La colonne pg_class.relfrozenxid indique le plus ancien XID non gelé de la table
- Cette colonne est mise à jour par la dernière opération de VACUUM agressif réussie
- Au niveau base de données, la colonne pg_database.datfrozenxid indique le min de tous les pg_class.relfrozenxid

Requêtes SQL pour le suivi des XID

```
SELECT c.oid::regclass as table_name,
       greatest(age(c.relfrozenxid),age(t.relfrozenxid)) as age
FROM pg_class c
LEFT JOIN pg_class t ON c.reltoastrelid = t.oid
WHERE c.relkind IN ('r', 'm');

SELECT datname, age(datfrozenxid) FROM pg_database;
```

- La colonne age indique le nombre de transactions entre le XID courant et celui du cutoff du VACUUM agressif
- relkind = 'r' : table ordinaire
- relkind = 'm' : vue matérialisée

https://www.postgresql.org/docs/15/catalog-pg-class.html

Logs de suivi des VACUUM

- Lorsque l'option **VERBOSE** de VACUUM est activée, certaines statistiques de la table sont affichées.
- L'évolution des champs **relfrozenxid** et **relminmxid** est mentionné
- Le même niveau de détails de log est activé lorsque l'option **log_autovacuum_min_duration** a une valeur différente de -1
- Valeur par défaut de l'option **log_autovacuum_min_duration** = 10 minutes
- Cette option est pratique car elle permet de surveiller l'activité de l'autovacuum

Le champ relminmxid

- To translate
- All multixact IDs before this one have been replaced by a transaction ID in this table. This is used to track whether the table needs to be vacuumed in order to prevent multixact ID wraparound or to allow pg_multixact to be shrunk. Zero (InvalidMultiXactId) if the relation is not a table.

```
https://www.postgresql.org/docs/15/catalog-pg-class.html
```

En cas d'échec de l'autovacuum

- En cas d'échec de l'autovacuum, le serveur émet le warning suivant avant la limite $2^{32} - 40$ millions de transactions :

```
WARNING:  database "mydb" must be vacuumed within 39985967 transactions
HINT:  To avoid a database shutdown, execute a database-wide VACUUM in that database.
```

- Un VACUUM lancé par un **super utilisateur** devrait résoudre le problème
- Les droits **super utilisateur** sont nécessaires pour modifier le champ **datfrozenxid**
- Si le DBA ignore les avertissements, le serveur s'arrête et refuse toute nouvelle transaction avant la limite $2^{32} - 3$ millions de transactions :

```
ERROR:  database is not accepting commands to avoid wraparound data loss in database "mydb"
HINT:  Stop the postmaster and vacuum that database in single-user mode.
```

- Dans ce cas, il devient nécessaire de lancer le serveur en mode single-user

Démarrage du serveur PostgreSQL en mode single-user

- Pour démarrer le serveur en mode single-user, lancer la commande suivante :

```
postgres --single -D /usr/local/pgsql/data other-options my_database
```

- Dans ce mode, le serveur n'applique pas les mesures de sécurité pour éteindre le process en cas de "danger"
- A ce moment, il devient possible de lancer le VACUUM FREEZE

https://www.postgresql.org/docs/15/app-postgres.html

Multi-transactions et repli des identifiants de transactions

- Il est possible qu'une ligne de table soit verrouillée par plusieurs transactions
- Les identifiants de transactions multiples sont stockés dans le répertoire **pg_multixact**
- Comme pour les transactions classiques, ils sont stockées sur 32 bits et varient entre 0 et 2^{32}
- Les seuils de traitement d'une ligne et d'une table sont respectivement **vacuum_multixact_freeze_min_age** et **vacuum_multixact_freeze_table_age**
- L'indicateur d'âge d'une table pour les multi-transactions est : **pg_class.relminmxid**
- De manière similaire, toute table ayant l'âge **autovacuum_multixact_freeze_max_age** pour ses multitransactions, sera traitée par un VACUUM agressif

Le démon autovacuum

- L'autovacuum lance automatiquement le VACUUM et l'NALAYZE (collecte de statistiques) sur chaque table
- Par défaut, le paramètre **track_counts** = on. Il est utilisé par l'autovacuum pour suivre l'activité de la base de données et se déclencher au moment opportun
- Le démon autovacuum se décline en plusieurs process :
  - l'**autovacuum launcher** en charge de démarrer l'**autovacuum worker** pour chaque base de données
  - le launcher démarre un work tous les **autovacuum_naptime** secondes pour chaque base de données
- un work sera donc lancé tous les autovacuum_naptime/N secondes s'il y a N bases de données
- La limite est de **autovacuum_max_workers** worker par base de données

Le paramètre autovacuum_max_workers

- Si le nombre de bases de données est supérieur à autovacuum_max_workers, la prochaine base de données sera traité sitôt que le worker aura fini.
- Les worker ne sont pas inclus dans les limites **max_connections** or **superuser_reserved_connections**.

```
https://www.postgresql.org/docs/15/runtime-config-autovacuum.html
```

A quel moment autovacuum se déclenche - modifications/suppressions

- Lorsqu'une table a une valeur **relfrozenxid** plus âgée que **autovacuum_freeze_max_age**
- Lorsque le nombre de relations de la table (modifiées ou supprimées depuis le dernier VACUUM) atteint le seuil suivant :

```
vacuum threshold = vacuum base threshold + vacuum scale factor * number of tuples
```

- avec vacuum base threshold = autovacuum_vacuum_threshold et vacuum scale factor = autovacuum_vacuum_scale_factor
- le nombre de tuples est fourni par pg_class.reltuples

A quel moment autovacuum se déclenche - insertions

- Lorsque le nombre de relations de la table (inserées depuis le dernier VACUUM) atteint le seuil suivant :

```
vacuum insert threshold = vacuum base insert threshold + vacuum insert scale factor * number of tuples
```

- avec vacuum base insert threshold = autovacuum_vacuum_insert_threshold et vacuum scale factor = autovacuum_vacuum_insert_scale_factor
- le nombre de tuples est fourni par pg_class.reltuples
- Pour une table modifiée principalement par **INSERT** et peu par **UPDATE** ou **DELETE**, il est intéressant de baisser le seuil **autovacuum_freeze_min_age** pour alléger la charge des prochains VACUUM (*avec un risque de dégel peu élevé*)

A quel moment autovacuum réalise un ANALYZE

```
analyze threshold = analyze base threshold + analyze scale factor * number of tuples
```

- Le seuil d'ANALYZE est comparé avec le nombre de lignes en INSERT, DELETE ou UPDATE depuis le dernier ANALYZE
- avec analyze base threshold = **autovacuum_analyze_threshold** et analyze scale factor = **autovacuum_analyze_scale_factor**

Limites de l'autovacuum

- Les tables partitionnées ne sont pas traitées par l'autovacuum
- Les statistiques sont collectés manuellement en lançant un ANALYZE manuel à l'initialisation de la table et après chaque changement significatif
- Les tables temporaires ne sont pas accessibles par l'autovacuum
- Elles nécessitent un traitement manuel de l'ANALYZE et VACUUM en passant par des commandes SQL
- Les paramètres de l'autovacuum sont définis dans **postgresql.conf**. Cependant, il est possible de les surcharger directement à la définition de la table

```
https://www.postgresql.org/docs/15/sql-createtable.html#SQL-CREATETABLE-STORAGE-PARAMETERS
```

Répartition de la charge du VACUUM par les facteurs de coûts

- Durant les opérations de VACUUM ou d'ANALYZE, les opérations I/O peuvent être pénalisantes pour les autres opérations du serveur
- Lors de l'exécution des commandes VACUUM et ANALYZE, le serveur garde une trace des coûts I/O engendrés par ces opérations et réalise une accumulation de ces coûts.
- Il est possible pour l'administrateur de définir un seuil d'accumulation d'I/O à partir duquel le VACUUM rentre en sommeil et les coûts sont remis à 0.
- Par défaut, cette fonctionnalité est **désactivée** pour les opérations manuelles. Pour l'activer, il est nécessaire de positionner le paramètre **vacuum_cost_delay** à une valeur différente de zéro.
- Le seuil est défini par le paramètre **vacuum_cost_limit**. Valeur par défaut 200.
- La durée de sommeil du process VACUUM en millisecondes est : **vacuum_cost_delay**. Valeur par défaut 0.

Comment est calculé le coût du VACUUM

- Les différents critères utilisés pour calculer le coût d'un VACUUM sont :
  - **vacuum_cost_page_hit** : Coût de verrouillage du buffer, recherche et scan de la page associée. Valeur par défaut : 1
  - **vacuum_cost_page_miss** : Coût de verrouillage, récupération de la page depuis le disque et scan. Valeur par défaut : 2
  - **vacuum_cost_page_dirty** : Coût de modification d'une page par le VACUUM (gel, table de visibilité, ...). I/O nécessaire pour écrire la page en disque. Valeur par défaut : 20
- Ces paramètres de coût s'appliquent sur l'ensemble des workers et sont répartis sur cet ensemble.
- Cependant, lorsqu'un worker traite une table ayant positionné **autovacuum_vacuum_cost_delay** ou **autovacuum_vacuum_cost_limit**, les coûts induits par ce traitement ne sont pas inclus dans les coûts globaux.

https://www.postgresql.org/docs/15/runtime-config-resource.html#RUNTIME-CONFIG-RESOURCE-VACUUM-COST

Traitement des verrous par l'autovacuum

- Les workers autovacuum ne bloquent pas les autres process en général
- Si un process essaie d'acquérir un verrou de type **SHARE UPDATE EXCLUSIVE** qui entre en collision avec l'autovacuum, l'autovacuum est interrompu
- Dans le cas où l'autovacuum démarre pour prévenir le pli des identifiants de transaction (XID wraparound), celui-ci n'est pas interruptible
- La requête lancée par l'autovacuum dans ce cas se termine par la chaîne "(to prevent wraparound)" dans la vue pg_stat_activity
- Le lancement fréquent de commandes qui réclame des verrous de type **SHARE UPDATE EXCLUSIVE** empêche l'autovacuum de se terminer systématiquement

La réindexation

- Lorsque des clés de l'index sont regulièrement supprimées, il est intéressant de lancer la commande **REINDEX**
- Elle permet de récupérer l'espace non utilisé et améliore légèrement l'accès aux index
- REINDEX réorganise les clés de l'index de manière adjacente pour permettre un accès optimal
- L'occupation de l'espace disque par les indexes non B-tree n'est pas très bien maîtrisée.
- Il est important de surveiller l'espace disque occupé par les index
- REINDEX pose par défaut un verrou de type **ACCESS EXCLUSIVE** sur la table en cours d'indexation.
- Il est impossible de modifier la table pendant ce type de réindexation
- Le process de réindexation renseigne la table **pg_stat_progress_create_index** pendant son exécution

La réindexation concurrente

- La réindexation concurrente est activée avec l'option **CONCURRENTLY**
- Elle permet la modification de la table en cours de réindexation
- Elle est plus gourmande en CPU et I/O car l'index est généré en 2 passes :
  - Une première passe pour scanner la table et regénérer un index temporaire
  - Cet index devient disponible pour l'ajout de clés
  - Une deuxième passe pour récupérer les clés générées pendant la première phase

```
https://www.postgresql.org/docs/15/sql-reindex.html
```

Troubleshooting de la réindexation concurrente

- En cas d'erreur de rebuild de l'index, la commande \d appliquée à la table permet de vérifier l'état des index de la table :

```
   postgres=# \d tab
       Table "public.tab"
 Column |  Type   | Modifiers
--------+---------+-----------
 col    | integer |
Indexes:
    "idx" btree (col)
    "idx_ccnew" btree (col) INVALID
```

- Dans le cas présent, l'index est à l'état invalide
- Il est suffixé par **ccnew**, il correspond à l'index temporaire créé à la première passe
- Il suffit de le supprimer avec la commande `DROP INDEX` et de le recréer avec `REINDEX CONCURRENTLY`
- S'il est suffixé par **ccold**, il correspond à l'index original qui est maintenant obsolète
- Il suffit de le supprimer avec la commande `DROP INDEX`

Limites de la réindexation concurrente

- Il est possible d'appliquer plusieurs **REINDEX** sur d'autres index de la table en parallèle
- Ce n'est pas possible avec **REINDEX CONCURRENTLY**
- **REINDEX CONCURRENTLY** n'est pas utilisable dans une transaction alors que **REINDEX** est utilisable dans une transaction
- **REINDEX SYSTEM** n'est pas utilisable avec l'option **CONCURRENTLY**
- L'option **CONCURRENTLY** ne peut pas être utilisable sur un index de type contrainte d'exclusion. Ce type d'index peut être réindexé sans l'option **CONCURRENTLY**

https://www.postgresql.org/docs/current/ddl-partitioning.html#DDL-PARTITIONING-CONSTRAINT-EXCLUSION

Exercices

- VACUUM
- VACUUM FULL après DELETE pour vérifier récuperation espace disque
- Modifier freeze_min_age et freeze_table_age
- VACUUM FREEZE
- relfrozenxid
- reindexation

L'importance des sauvegardes

- Il est important de posséder 2 copies supplémentaires à la copie des données de productions.
- Une des 2 copies se situe hors site.
- Chacune des 2 copies est stockée sur un média différent.

```
https://www.it-connect.fr/sauvegarde-quest-ce-que-la-regle-du-3-2-1/
```

Différents types de sauvegardes

Il existe différents types de sauvegarde :

- logique avec les commandes `pg_dump` et `pg_dumpall`
- au niveau du système de fichiers avec la commande `tar`.[1]
- binaire avec la commande `pg_basebackup`

`https://www.postgresql.org/docs/15/backup.html`

1. La sauvegarde du système de fichiers n'est pas recommandée par la documentation officielle de PostgreSQL

Supervision du serveur de base de données

Supervision de la base de données

TODO WIP

```
https://www.postgresql.org/docs/15/monitoring.html
```

Tests de vérification de la bonne santé de la base

TODO WIP

```
https://bucardo.org/check_postgres/
```

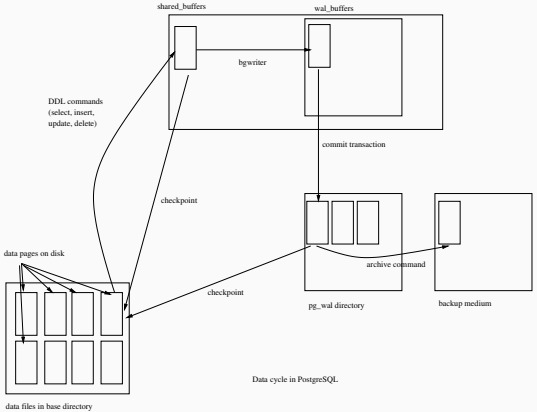
Supervision de l'activité des disques

- TODO

```
https://www.postgresql.org/docs/15/diskusage.html
```

Point In Time Recovery

Cycle des données dans PostgreSQL



https://www.postgresql.org/docs/15/wal-configuration.html

Multiversion Concurrency Control - MVCC

- PostgreSQL fournit un grand nombre d'outils pour gérer l'accès concurrentiel aux données
- De manière interne, la cohérence des données est mise en place en utilisant un modèle multi-versions (MVCC)
- Cela signifie que chaque requête SQL a la vision d'un instantané des données (snapshot)
- Cet instantané correspond à une version de la base de données il y a quelques instants, quelques soient les modifications actuellement réalisées sur les données
- Cela empêche les requêtes de voir des données incohérentes modifiées par des requêtes parallèles
- Cette méthode fournit l'isolation transactionnelle pour chaque session de la base des données

https://www.postgresql.org/docs/15/mvcc-intro.html

Multiversion Concurrency Control - MVCC

- Le MVCC en évitant les méthodes de verrouillage des bases de données classiques minimise la contention et favorise la performance dans un environnement multi-utilisateurs
- En cas de nécessité de verrouillage, il existe différents type de verrous :
  - le verrouillage en lecture
  - le verrouillage en écriture
  - ces 2 types de verrous ne se gênent pas mutuellement : un verrou en lecture n'empêche pas l'écriture et vice-versa

Mise en place de la génération des WAL

- Pour activer la génération des WAL, merci de modifier les paramètres suivants dans *postgresql.conf*

```
wal_level = replica # or higher
archive_mode = on
[...]
```

https://www.postgresql.org/docs/15/continuous-archiving.html

archive_command et archive_library

- La copie des WAL vers le système de sauvegarde se paramètre depuis les 2 valeurs suivantes :
  - **archive_command** : Commandes shell d'archivage des WAL
  - **archive_library** : Binaire d'archivage des WAL
- Chacune des 2 valeurs s'utilise au choix. Elles ne peuvent utilisées en même temps.
- Pour les exercices, le paramètre **archive_command** sera utilisé.

```
archive_command = 'test ! -f /mnt/server/archivedir/%f && cp %p /mnt/server/archivedir/%f' # Unix
   [...]
```

Fonctionnement de l'archivage des WAL

- La commande d'archivage des WALs est lancée par le même utilisateur du process *postmaster*
- Elle est censée renvoyer un code différent de 0 en cas d'erreur
- Il est important de vérifier que le WAL n'existe pas dans le répertoire cible sous peine de l'écraser
- En cas d'erreur d'archivage, PostgreSQL retente autant de fois que nécessaire
- Tant que le fichier WAL n'est pas archivé, celui-ci n'est pas supprimé du répertoire pg_wal, ce qui peut amener à une saturation du système de fichiers
- En cas de saturation du répertoire pg_wal, le serveur s'arrête avec une erreur de type PANIC. Il pourra redémarrer une fois qu'il aura de l'espace disque disponible

Caractéristiques des WAL

- Le nom des fichiers WAL inclut jusqu'à 64 caractères
- Il est composé de lettres ASCII, de chiffres et de points
- Il est important de garder le nom de l'archive WAL
- La sauvegarde des WALs ne permet pas de restaurer *postgresql.conf*, *pg_hba.conf* ou *pg_ident.conf*
- Il est important de sauvegarder ces 3 fichiers indépendamment
- Ces fichiers peuvent être stockés dans un répertoire paramétrable

Timelines (lignes de temps)

- Après chaque opération de recovery, PostgreSQL créée une nouvelle ligne de temps (timeline)
- Au passage, le serveur crée un fichier "timeline history" qui lui indique la série de WAL générés après le recovery
- Les WALs créés après l'opération de recovery appartiennent à cette nouvelle ligne de temps
- Par défaut, l'opération de recovery rétablit la timeline la plus récente
- Il est possible de rétablir une timeline donnée dans le passé

```
https://www.postgresql.org/docs/15/continuous-archiving.html
```

Format du nom du WAL

Dans le répertoire /pg_wal, on peut voir le WAL suivant :

- **00000001**000000000000001B
- La timeline est représentée par les 8 premiers chiffres du nom du WAL

https://www.postgresql.org/docs/15/continuous-archiving.html

La commande pg_basebackup

- La commande `pg_basebackup` sauvegarde un cluster PostgreSQL
- Elle ne s'applique pas sur une base de données en particulier
- L'utilisateur lançant cette commande doit posséder le droit **REPLICATION** ou être un super-utilisateur
- La commande peut être lancée sur un serveur standby
- La table *pg_stat_progress_basebackup* donne une idée de la progression de la commande

```
https://www.postgresql.org/docs/15/app-pgbasebackup.html
https://www.postgresql.org/docs/15/progress-reporting.html#BASEBACKUP-PROGRESS-REPORTING
```

Options de la commande `pg_basebackup`

Les options qui semblent les plus intéressantes sont :

- `-F format`
- *format* a les valeurs suivantes :
  - *p* ou *plain*. Format par défaut.
  - *t* ou *tar*. Génération d'une archive du répertoire des données *base.tar*

`https://www.postgresql.org/docs/15/app-pgbasebackup.html`

Options de la commande `pg_basebackup`

- `-R` or `--write-recovery-conf`
  - la commande génère automatiquement le fichier **standby.signal**
- `-T` *`olddir=newdir`* ou `--tablespace-mapping=`*`olddir=newdir`*
  - cette option permet de mapper les répertoires du serveur liés aux tablespaces à d'autres répertoires locaux. Option valide uniquement si le format **plain** est utilisé.
- `-X` *`method`* ou `--wal-method=`*`method`*
  - *method* a les valeurs suivantes :
    - *n* ou *none*. Le backup n'inclut pas les WAL.
    - *f* ou *fetch*. Les WAL générés durant le backup sont récupérés une fois le backup généré. Dans le cas de l'utilisation de cette option, le paramètre `wal_keep_size` nécessite d'être ajusté correctement afin que le serveur puisse garder l'ensemble des WAL (et ne supprime ni n'en recycle durant le backup).
    - *s* ou *stream*. Mode par défaut. Une $2^{ème}$ connexion est établie vers le serveur pour streamer les WAL générés durant le backup.

Options de la commande `pg_basebackup`

- `--no-estimate-size`
  - le serveur n'estime plus la taille du backup avant de débuter le process. Cela permet de raccourcir la durée de la génération du backup.
- **Environment**
  - Cet utilitaire, tout comme la majorité des utilitaires PostgreSQL, utilisent les mêmes variables d'environnement que ceux de la libpq.
  - La variable d'environment PG_COLOR indique si la couleur est utilisée dans les messages de diagnostique. Les valeurs possibles sont *always*, *auto* et *never*.

Checkpoints

- A chaque checkpoint, l'ensemble des pages chargées et modifiées (dirty pages) dans les shared_buffers sont flushées en disque.
- Les données présentes dans les WAL sont également écrites en disque.
- En cas de rejeu des WAL (**REDO**), le serveur part de la dernière transaction liée au checkpoint pour rejouer les WALs.
- Cette opération engendre une forte activité I/O
- La fréquence des checkpoint est liée à 2 paramètres :
  - **checkpoint_timeout**
  - **max_wal_size**

https://www.postgresql.org/docs/current/wal-configuration.html

checkpoint_timeout et max_wal_size

checkpoint_timeout

- Intervalle en secondes entre 2 checkpoints automatiques

max_wal_size

- Taille disque maximale occupée par les WAL. Lorsque cette taille est atteinte, l'opération de checkpoint est déclenchée ainsi que archive_command.

min_wal_size

- Taille disque minimale occupée par les WAL. Lorsque ce seuil est atteint, les WAL ne sont plus supprimés mais recyclés (renommés).

```
https://www.postgresql.org/docs/current/runtime-config-wal.html#GUC-CHECKPOINT-TIMEOUT
https://www.postgresql.org/docs/current/runtime-config-wal.html#GUC-MAX-WAL-SIZE
```

La commande pg_verifybackup

- La commande `pg_verifybackup` vérifie l'intégrité d'une sauvegarde générée par la commande `pg_basebackup`
- Elle s'applique sur un backup généré au format plain. Dans le cas de la vérification d'une archive tar, il est nécessaire de désarchiver auparavant.

https://www.postgresql.org/docs/15/app-pgverifybackup.html

La commande pg_amcheck

- La commande `pg_amcheck` vérifie l'intégrité d'une base de données.
- Fait appel à l'utilitaire amcheck
- S'applique aux tables ordinaires, celles de type TOAST, les vues matérialisées, les séquences, les indexes btree.
- Possède des options de filtrages de base, de schéma et de table

```
https://www.postgresql.org/docs/current/app-pgamcheck.html
https://www.postgresql.org/docs/current/amcheck.html
```

Principales options de la commande `pg_amcheck`

Les principales options de cette commande sont :

- `--no-dependent-indexes` exclut de la vérification les indexes associés à la table
- `--no-dependent-toast` exclut de la vérification les tables TOAST associées à la table
- Options applicables aux indexes
  - `--heapallindexed` crée un nouvel index temporaire en mémoire pour vérifier que l'index actuellement stocké est valide. Cette option utilise de la RAM limitée par le paramètre `maintenance_work_mem`

Options de la commande `pg_amcheck`

- `--parent-check` vérifie les indexes de la table parent [2]

2. https ://www.postgresql.org/docs/current/ddl-inherit.html

Exercice

Réalisation d'un backup full de la base de données

La procédure de déroulement du PITR s'appuie sur :

- la génération d'un backup full
- l'archivage des WALs

Il existe 2 méthodes pour réaliser une sauvegarde full de la base données :

- l'outil pg_basebackup
- l'API bas niveau avec des appels aux fonctions PostgreSQL

La méthode étudiée pendant les exercices est l'outil pg_basebackup.

Mise en place de l'archivage des WALs

- Vérifier que PostgreSQL est déployé sur les serveurs hqpg-0x et hqpg-0x-repl
- Mettre en place la copie des WAL vers le répertoire `/opt/wal_backup` avec le niveau **replica**
- La commande archive_command s'appuie sur rsync pour transférer les WALs depuis le primaire. Pour éviter d'avoir à indiquer le mot de l'utilisateur postgres, réaliser la génération et l'échange mutuel de clefs SSH sur les 2 serveurs comme décrit en page 164
- Paramétrer la commande archive_command dans postgresql.conf

```
archive_command = 'rsync %p hqpg-0x-repl:/opt/wal_backup/%f'
```

Résumé du paramétrage de la sauvegarde des WALs

Sur le serveur principal, les lignes suivantes sont inclues dans postgresql.conf :

```
# Add settings for extensions here
listen_addresses = '*'
wal_level = replica
archive_mode = on
archive_command = 'rsync %p 10.10.10.29:/opt/archivedir/%f'
```

- Sur le serveur de sauvegarde, créer le répertoire `/opt/wal_backup/` de sauvegarde des WALs avec l'utilisateur postgres

Remarque importante sur la procédure de PITR

Remarque Le point de restauration est obligatoirement après la date de fin du base backup, c'est à dire, le temps de fin de l'appel à pg_basebackup_stop. Il n'est possible de restaurer des données à une date durant laquelle le backup est en cours. Pour restaurer à une telle date, il est nécessaire de restaurer le précédent backup et de rejouer les sauvegardes WALs jusqu'à ce point.

Utilisation de pg_basebackup pour sauvegarder la base de données

Sur le serveur principal, ajouter la ligne suivante dans pg_hba.conf :

```
host  replication  postgres  10.10.10.0/24  trust
```

Recharger le paramétrage avec la commande suivante :

```
postgres=# SELECT pg_reload_conf();
```

Génération d'un backup full

Sur le serveur de sauvegarde des WAL, appliquer les commandes suivantes :

- Comme indiqué dans [3], la commande pg_basebackup réalise un checkpoint (p. 145) sur le serveur
- `pg_basebackup -h hqpg-0x -Ft -z -P -D /opt/backup/20230213`

3. https ://www.postgresql.org/docs/current/app-pgbasebackup.html

Inspection de l'archive générée

```
[postgres@localhost 20230202_1]$ ls -lrt
total 4420
-rw------- 1 postgres postgres  181880  2 févr. 07:58 backup_manifest
-rw------- 1 postgres postgres 4321056  2 févr. 07:58 base.tar.gz
-rw------- 1 postgres postgres   17075  2 févr. 07:58 pg_wal.tar.gz
```

- backup_manifest inclut le listing avec les checkum de l'archive base.tar.gz
- pg_wal.tar.gz inclut les WAL générés durant l'opération du pg_basebackup

Création du jeu de données

Sur le serveur principal, lancer les commandes suivantes :

```
su - postgres
creatuser hq -d
createdb hqdb -O hq
psql hqdb
-- 02/02/2023 16h27
create table test_pitr1 (col1 text);
create table test_pitr2 (col1 text);
-- quelques minutes plus tard
-- 16h30
create table test_pitr3 (col1 text);
```

https://www.postgresql.org/docs/current/app-createuser.html
https://www.postgresql.org/docs/current/app-createdb.html

Restauration de la base de données à une date donnée

Sur le serveur PostgreSQL principal,

```
[root@localhost 15]# systemctl stop postgresql-15.service
[root@localhost 15]# su - postgres
Dernière connexion : samedi  4 février 2023 à 20:01:48 EST sur pts/0
[postgres@localhost ~]$ cd 15
# mv data data.20230213
# scp 10.10.10.29:/opt/backup/20230205_1/* .
# rm -rf data && mkdir data && chmod 0700 data
# tar xvzf ../base.tar.gz
# cd pg_wal/
# tar xvzf ../../pg_wal.tar.gz
#  cd ..
#  touch recovery.signal
```

https://www.postgresql.org/docs/15/runtime-config-wal.html#RUNTIME-CONFIG-WAL-RECOVERY-TARGET

Restauration de la base de données à une date donnée

Ajouter les lignes suivantes dans postgresql.conf

```
restore_command = 'rsync 10.10.10.29:/opt/archivedir/%f %p'
recovery_target_time = '2023-02-05 02:20:00+01'
```

Démarrer le serveur avec la commande.

Logs de la restauration

```
rsync: link_stat "/opt/archivedir/00000002.history" failed: No such file or directory (2)
rsync error: some files/attrs were not transferred (see previous errors) (code 23) at main.c(1670) [Receiver=3.1.3]
2023-02-04 20:28:02.203 EST [22385] LOG:  début de la restauration de l'archive à 2023-02-04 20:20:00-05
2023-02-04 20:28:03.278 EST [22385] LOG:  la ré-exécution commence à 0/1C000028
2023-02-04 20:28:03.648 EST [22385] LOG:  restauration du journal de transactions « 00000001000000000000001D » à partir
2023-02-04 20:28:05.011 EST [22385] LOG:  état de restauration cohérent atteint à 0/1C000100
2023-02-04 20:28:05.011 EST [22381] LOG:  le système de bases de données est prêt pour accepter les connexions en lectu
rsync: link_stat "/opt/archivedir/00000001000000000000001F" failed: No such file or directory (2)
2023-02-04 20:28:05.172 EST [22385] LOG:  arrêt de la restauration avant validation de la transaction 740, 2023-02-04 2
2023-02-04 20:28:05.172 EST [22385] LOG:  pause à la fin de la restauration
2023-02-04 20:28:05.172 EST [22385] ASTUCE :  Exécuter pg_wal_replay_resume() pour promouvoir.
2023-02-04 20:33:01.702 EST [22383] LOG:  début du restartpoint : time
2023-02-04 20:33:05.828 EST [22383] LOG:  restartpoint terminé : a écrit 26 tampons (0.2%); 0 fichiers WAL ajoutés, 2 s
2023-02-04 20:33:05.829 EST [22383] LOG:  la ré-exécution en restauration commence à 0/1E000028
```

Basculement de la base de données en mode écriture

```
hqdb=# create table test_pitr_20230206144000 (col1 text);
ERREUR:  ne peut pas exécuter CREATE TABLE dans une transaction en lecture seule
hqdb=# quit
[postgres@localhost pg_wal]$ psql
psql (15.1)
Saisissez « help » pour l'aide.

postgres=# select pg_wal_replay_resume();
 pg_wal_replay_resume
----------------------

(1 ligne)

postgres=# create table test_pitr_20230206144000 (col1 text);
CREATE TABLE
```

Apparition d'une nouvelle ligne de temps

```
[postgres@localhost pg_wal]$ ls -lrt
total 32772
-rw------- 1 postgres postgres 16777216  6 févr. 08:41 000000010000000000000001E
-rw------- 1 postgres postgres       51  6 févr. 08:41 00000002.history
drwx------ 2 postgres postgres       72  6 févr. 08:41 archive_status
-rw------- 1 postgres postgres 16777216  6 févr. 08:46 000000020000000000000001E
```

Génération et échange mutuel des clés SSH

```
[linagora@localhost ~]$ sudo -i
[root@localhost ~]# su - postgres
[postgres@localhost ~]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/var/lib/pgsql/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
```

- Copier la clef publique dans le répertoire $\sim$ /*postgres*/*.ssh*/*authorized_keys*
- Vérifier avec la commande ssh postgres@hqpg-0x-repl et inversement que la session SSH est établie sans avoir à fournir de mot de passe

En cas de corruption des WALs

- Dans le cas où l'opération de recovery détecte un WAL corrompu, l'opération s'interompt et le serveur ne démarre pas. Il est possible de relancer l'opération de recovery avec une cible de recovery située avant le point de corruption pour permettre à l'opération de se terminer.
- Dans le cas où l'opération de recovery échoue pour une raison externe (crash système ou archive WAL inaccessible), le recovery peut être relancera. Il reprendra proche du point où il s'est interrompu.
- Le redémarrage du mode recovery fonctionne de manière similaire au checkpoint en mode normal : le serveur enregistre son état de manière périodique en disque, puis met à jour le fichier pg_control pour indiquer les WAL déjà traités et qui ne nécessitent pas d'être scannées à nouveau.

https://www.postgresql.org/docs/15/continuous-archiving.html

Modèles de base de données - Template databases

- La commande **CREATE DATABASE** fonctionne en copiant une base de données.
- Par défaut, elle copie la base de données standard **template1**.
- Si des objets sont créés dans template1, ils seront présents dans les bases créées ultérieurement
- Par exemple, si le langage PL/Perl est déployé sur template1, il sera présent sur l'ensemble des bases dérivées

```
https://www.postgresql.org/docs/15/manage-ag-templatedbs.html
```

Identifier les points de contention

Les différents niveaux de verrouillage dans PostgreSQL

Il existe différents niveaux de verrouillage dans PostgreSQL. Ces niveaux sont :

- table
- ligne
- page

```
https://www.postgresql.org/docs/current/explicit-locking.html
```

Les deadlocks - Verrouillages mortels

- PostgreSQL est capable de détecter les deadlocks
- Dans cette situation, le serveur arrête l'une des sessions bloquantes

Les verrous de niveau table

- WIP

Le schéma *information*

- yyyyy

```
https://www.postgresql.org/docs/15/information-schema.html
```

Les vues système

- yyyyy

```
https://www.postgresql.org/docs/15/views.html
```

Le catalogue système

- yyyyy

```
https://www.postgresql.org/docs/15/catalogs.html
```

Le module pg_stat_statements

- yyyyy

```
https://www.postgresql.org/docs/15/pgstatstatements.html
```

Le module pg_buffercache

- yyyyy

```
https://www.postgresql.org/docs/current/pgbuffercache.html
https://easyteam.fr/postgresql-tout-savoir-sur-le-shared_buffer/
```

La vue pg_lock

- yyyyy

```
https://www.postgresql.org/docs/15/view-pg-locks.html
```

L'isolation des transactions

- yyyyy

https://www.postgresql.org/docs/15/transaction-iso.html

Bibliographie

Webographie

Sommaire

Sommaire

Conclusion