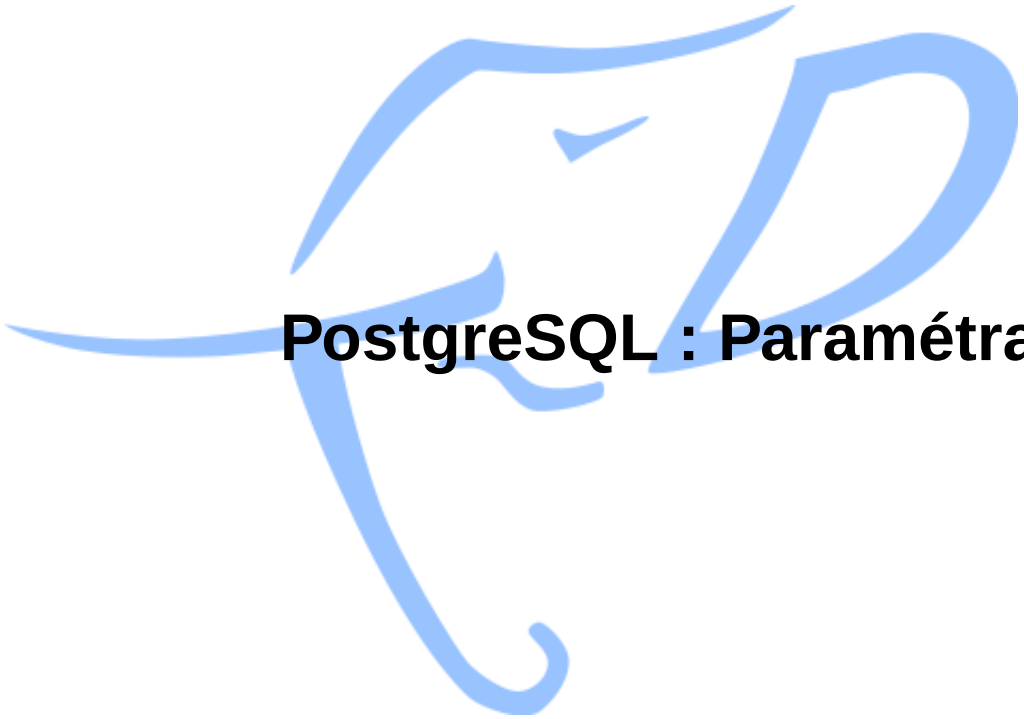
DALIBO

L'Expertise PostgreSQL



PostgreSQL : Paramétrage

Formateur : Jean-Paul Argudo
Contact : jean-paul.argudo@dalibo.com
Date : mars 2010

Table des Matières

Introduction

- Introduction au paramétrage
- Paramétrage du moteur

Les paramètres de configuration sont regroupés au sein de 6 grandes familles :

  - Connexion
  - Ressources
  - WAL
  - Planificateur
  - Statistiques
  - VACUUM

- Paramétrage des accès aux bases de données

PostgreSQL dispose d'un large éventail de méthodes d'authentification et de restriction des accès.

Licence Creative Commmons CC-BY-NC-SA

Fichiers de configuration pour PostgreSQL

- `postgresql.conf`
  paramétrage global de PostgreSQL
- `pg_hba.conf`
  authentification des clients
- `pg_ident.conf`
  mapping entre les utilisateurs IDENT et PostgreSQL

Introduction au paramétrage

- Fichier de configuration `postgresql.conf`
  - ⇒ un seul paramètre par ligne
  - ⇒ les commentaires commencent par `#`
  - ⇒ valeurs des paramètres entre guillements simples pour les chaînes

```
listen_addresses = '*'
```

Paramètrage spécifique

- Paramétrage spécifique pour une base de données, un utilisateur ou une fonction
  Il est possible de surcharger un ensemble de paramètre pour un utilisateur, pour une base de données, pour une fonction (ce dernier à partir de la 8.3) avec les commandes, respectivement, `ALTER DATABASE`, `ALTER USER` et `ALTER FUNCTION`. Les paramètres surchargent les paramètres fournis au démarrage de PostgreSQL, mais pas les paramètres de session. Ex :

  ```
  ALTER USER test SET random_page_cost = 3 ;
  ```

- Paramétrage à l'intérieur d'une session SQL
  Quelques paramètres peuvent être modifiés à l'intérieur de sessions SQL individuelles avec la commande SET. Ex :

  ```
  SET enable_seqcan TO off ;
  ```

- Inspection des paramètres
  La commande SHOW permet d'obtenir la valeur d'un paramètre. Ex :

  ```
  psql> SHOW random_page_cost ;
  ```

  De plus, tous les paramètres peuvent être trouvés dans la vue `pg_settings` :

  ```
  psql> SELECT setting FROM pg_settings WHERE name='random_page_cost' ;
  ```

  Cette vue contient aussi une courte description du paramètre dans la colonne `short_desc`.

Activer le paramétrage

- Rechargement du fichier de configuration *online*

```
pg_ctl reload
/etc/init.d/postgresql-8.2 reload
```

  Le processus `postmaster` relit le fichier de configuration à chaque fois qu'il reçoit le signal SIGHUP. Utiliser l'outil `pg_ctl` avec l'option `reload` permet d'envoyer le signal SIGHUP au processus `postmaster`.
  Sous `debian`, utiliser `/etc/init.d/postgresql-8.2 reload` pour cela (pour une version 8.2).
  Sous `windows`, `pause service` permet de recharger la configuration.
  Attention, quelques paramètres de configuration nécessitent un redémarrage complet du système de gestion de bases de données, elles sont alors clairement indiquées comme telles dans la documentation.
- Passage de paramètre au démarrage

```
postmaster -c log_connections=yes -c log_destination='syslog'
```

  Il est possible de passer des paramètres de configuration au démarrage de PostgreSQL sur la ligne de commande avec l'option -c. Ces options surchargent alors le paramétrage du fichier `postgresql.conf`.
- Paramétrage par variable d'environnement

```
env PGOPTIONS='-c geqo=off' psql
```

  En positionnant la variable d'environnement PGOPTIONS, il est possible de modifier le comportement de PostgreSQL pour une session particulière (valable pour toutes les applications clientes basées sur la libpq).
  Exemple :

```
env PGOPTIONS='-c geqo=off' psql
```

  Sous `windows`, passer par les `propriétés de l'ordinateur`.

Le fichier postgresql.conf

- Emplacement des fichiers
- Connexion / sécurité
- Gestion des ressources et VACUUM
- WAL et checkpoints
- Planificateur et optimiseur génétique
- Logs, traces et stats
- Locales

Emplacement des fichiers (1/2)

- `data_directory`
  Répertoire à utiliser pour le stockage des données.
  Cette option peut seulement être spécifiée au lancement du serveur.
- `config_file`
  Fichier de configuration (`postgresql.conf`) principal.
  Cette option peut seulement être spécifiée au lancement du serveur.

Emplacement des fichiers (2/2)

- `hba_file` (pg_hba.conf)
  Fichier de configuration pour l'authentification.
  Cette option peut seulement être initialisée au lancement du serveur.
- `ident_file` (pg_ident.conf)
  Fichier de configuration pour l'authentification par ident.
  Cette option peut seulement être initialisée au lancement du serveur.
  Il est possible de conserver les fichiers de configuration dans un répertoire différent de celui préconisé lors de l'installation en utilisant l'option -D lors du lancement du postmaster ou en positionnant la variable d'environnement `PGDATA`.
- `external_pid_file` (postmaster.pid)
  Fichier contenant le PID du processus père.
  Cette option peut seulement être initialisée au lancement du serveur.

Configuration des connexions (1/3)

- `listen_addresses`

  Adresses TCP/IP sur lesquelles le serveur écoute.

  Paramètre initialisé au lancement du serveur.

  Liste de noms d'hôtes ou d'adresses IP séparés par des virgules. Le caractère '*' correspond à toutes les interfaces IP disponibles. Si la liste est vide, le serveur n'écoute aucune interface IP et la connexion ne peut alors se faire que par socket Unix.

  Ce paramètre a remplacé `tcpip_socket` en 7.4, qui activait ou non l'écoute sur les interface réseau et était un booléen.

- `port`

  Port TCP sur lequel le serveur écoute.

  Paramètre initialisé au lancement du serveur.

Configuration des connexions (2/3)

- `unix_socket_directory`

  Répertoire où est créé le fichier socket.
  Paramètre initialisé au lancement du serveur.

- `unix_socket_group`

  Groupe utilisateur de la socket.
  Paramètre initialisé au lancement du serveur.

- `unix_socket_permissions`

  Nombre octal précisant la configuration des droits sur la socket.
  Paramètre initialisé au lancement du serveur.

Configuration des connexions (3/3)

- `max_connections`

  Nombre maximum de connexions concurrentes au serveur.

  Paramètre initialisé au lancement du serveur.

  Attention : augmenter ce paramètre provoque une augmentation de la mémoire partagée utilisée ce que ne permet pas forcément la configuration par défaut du système d'exploitation.

- `superuser_reserved_connections`

  Nombre de connexions réservées aux super-utilisateurs

  Le nombre maximum de connexions concurrents pour les utilisateurs standards est donc de `max_connections` - `superuser_reserved_connections`.

Sécurité et authentification (1/2)

- `authentication_timeout`

  Temps maximum pour terminer l'authentification du client, en secondes

- `ssl`

  Utilisation de connexions ssl pour chiffrer les communications client/serveur
  Paramètre initialisé au lancement du serveur.

- `ssl_ciphers`

  Liste des algorithmes de chiffrement pour une connexion SSL.
  Paramètre initialisé au lancement du serveur.

Sécurité et authentification (2/2)

- `password_encryption`

  Détermine si le mot de passe doit être chiffré par défaut lors d'un `CREATE USER` ou `ALTER USER`.

- `db_user_namespace`

  Active les noms d'utilisateur par base de données. Les utilisateurs sont créés sous la forme nomutilisateur@nombase. Lorsqu'un client se connecte, le caractère @ et le nom de la base de connexion sont ajoutés au nom de l'utilisateur.

- `krb_server_keyfile`, `krb_srvname`, `krb_server_hostname`, `krb_caseins_users`, `krb_realm`

  Paramétrage du service Kerberos si utilisé pour l'authentification.

  Paramètre initialisé au lancement du serveur.

Consommation de ressources (1/4)

- `shared_buffers`

  Nombre de tampons en mémoire partagée

  Paramètre initialisé au lancement du serveur.

  Chaque tampon fait 8192 octets. Ce paramètre doit valoir au minimum 16 ou au minimum 2 fois la valeur de `max_connections`. Des valeurs de quelques milliers sont recommandées pour des installations en production. Sous Windows, les valeurs importantes sont moins efficace. Il est possible d'obtenir de meilleurs résultats avec des valeurs relativement basses (au maximum 50000 tampons, soit environ 400 Mo).

  Attention : augmenter ce paramètre provoque une augmentation de la mémoire partagée utilisée ce que ne permet pas forcément la configuration par défaut du système d'exploitation.

- `temp_buffers`

  Nombre maximum de tampons temporaires utilisés par chaque session.

  Chaque tampon fait 8 Ko en mémoire. À partir de la 8.2, il est possible d'indiquer cette valeur avec une autre unité.

  Ce sont des tampons locaux au niveau de la session utilisés seulement pour accéder aux tables temporaires.

- `work_mem`

  Mémoire à utiliser pour les opérations de tri interne et pour les tables de hachage avant de basculer sur des fichiers temporaires sur disque.

  La valeur est spécifiée en Ko et vaut par défaut 1024 (soit 1 Mo).

  Les opérations de tri sont utilisées pour `ORDER BY`, `DISTINCT` et les jointures d'assemblage. Les tables de hachage sont utilisées dans les jointures par hachage, les agrégations par hachage et le traitement des sous-requêtes `IN`.

  Si tous les processus l'utilisent, le serveur peut se voir contraint à utiliser le swap et les performances s'en ressentiront. Mais si cette valeur est trop basse, les processus postgres doivent stocker les résultats intermédiaires sur disque dès que la mémoire de tri est remplie.

  Il est possible de tracer l'utilisation de fichiers temporaires en configurant le paramètre `log_temp_files` à une valeur très basse (0 permet d'être sûr de tout tracer).

  Attention : la valeur `work_mem` peut être allouée dans le pire des cas plusieurs fois pour chacune des sessions. Il faut tenir compte de ce facteur lors du choix de cette valeur.

Consommation de ressources (2/4)

- `maintenance_work_mem`

  Mémoire maximum utilisée dans les opérations de maintenance telles que `VACUUM`, `CREATE INDEX` et `ALTER TABLE ADD FOREIGN KEY`.

  La valeur est spécifiée en Ko et vaut par défaut 16384 (soit 16 Mo). Des valeurs importantes (par exemple entre 256 et 512 Mo) pourraient améliorer les performances sur les opérations VACUUM et pour la restauration des sauvegardes de bases de données.

- `max_fsm_pages`

  Nombre maximum de pages disque pour lesquels les espaces libres seront tracés dans la *carte partagée des espaces libres* (*free space map*).

  Paramètre initialisé au lancement du serveur.

  6 octets de mémoire partagée sont consommés pour chaque emplacement de page. Ce paramétrage doit être supérieur à 16 * max_fsm_relations.

  Ce paramètre disparaît de la configuration de PostgreSQL à partir de la version `8.4`.

- `max_fsm_relations`

  Nombre maximum de relations (tables et index) pour lesquelles les espaces libres seront tracés dans la *carte partagée des espaces libres* (*free space map*).

  Paramètre initialisé au lancement du serveur.

  70 octets de mémoire partagée sont consommés par emplacement.

  Ce paramètre disparaît de la configuration de PostgreSQL à partir de la version `8.4`.

Consommation de ressources (3/4)

- `max_stack_depth`

  Profondeur maximum de la pile d'exécution du serveur.

  Le paramétrage par défaut est de 2048 Ko (soit 2 Mo), ce qui est très petit et comporte peu de risques. Néanmoins, cela pourrait être trop petit pour autoriser l'exécution de fonctions hautement récursives.

Consommation de ressources (4/4)

- `max_prepared_transactions`

  Nombre maximum de transactions pouvant être dans l'état « préparées » simultanément.

  Configurer ce paramètre à zéro désactive la fonctionnalité des transactions préparées, appelée « Two Phase Commit » et utilisée principalement par XA de Java.

- `max_files_per_process`

  Nombre maximum de fichiers ouverts simultanément permis pour chaque sous-processus serveur.

- `preload_libraries`

  Spécifie une ou plusieurs bibliothèques préchargées au lancement du serveur.

  En préchargeant une bibliothèque partagée (et en l'initialisant dans les cas applicables), le temps de lancement de la bibliothèque est évité à la première utilisation de la bibliothèque. Cela permet aussi de réaliser des allocations de mémoire partagée, impossibles à faire une fois le serveur démarré.

Paramètres de délais de VACUUM (1/2)

Afin de réduire l'impact des I/O des commandes VACUUM et ANALYZE, il est possible d'activer des compteurs qui, une fois un seuil atteint, provoquent un arrêt momentané des processus.

- `vacuum_cost_delay`

  Temps de repos du processus (en millisecondes) quand la limite de coût a été atteinte (0 par défaut, donc désactivé).

  La valeur de ce paramètre va généralement de 10 à 100. Une valeur supérieure peut souvent être considérée comme contre-productive.

- `vacuum_cost_page_hit`

  Coût estimé pour nettoyer via `vacuum` un tampon trouvé dans le cache des tampons partagés.

- `vacuum_cost_page_miss`

  Coût estimé pour nettoyer via `vacuum` un tampon qui doit être lu sur le disque.

Paramètres de délais de VACUUM (2/2)

- `vacuum_cost_page_dirty`

  Coût estimé chargé quand `vacuum` modifie un bloc qui était précédemment propre.

- `vacuum_cost_limit`

  Coût accumulé qui causera l'endormissement du processus de `vacuum`.

Note : Certaines opérations contenant des verrous critiques devraient se terminer aussi rapidement que possible.

Les délais de `vacuum` basés sur le coût ne surviennent pas pendant ces opérations. Du coup, il est possible que le coût accumulé soit bien plus important que la limite spécifiée. Pour éviter des délais inutilement longs dans de tels cas, le délai réel est calculé de la façon suivante

```
vacuum_cost_delay * accumulated_balance / vacuum_cost_limit
```

avec un maximum de vacuum_cost_delay * 4.

Note : Il est conseillé de faire varier vacuum_cost_delay et vacuum_cost_limit. Les trois autres paramètres ne servent qu'à estimer le coût de chaque opération et n'ont la plupart du temps pas à être modifiés.

Écriture en tâche de fond (1/2)

À partir de PostgreSQL 8.0, il existe un processus serveur séparé pour l'écriture en tâche de fond, dont la seule fonction est de demander au système d'exploitation les écritures des tampons partagés modifiés. Le but est que les processus serveur gérant les requêtes des utilisateurs déléguent les écritures des tampons à ce processus, ce qui réduira l'attente des utilisateurs.

Cette fonctionalité réduit aussi la pénalité de performance associée aux points de vérification, puisqu'une bonne partie des tampons à synchroniser à son déclenchement l'auront déjà été par ce processus.

- `bgwriter_delay`

  Délai, en millisecondes, entre les tours d'activité du processus d'écriture en tâche de fond.

- `bgwriter_lru_percent`

  Pourcentage des tampons les plus proches du recyclage à examiner.

  Ce paramètre disparaît en 8.3.

- `bgwriter_lru_maxpages`

  Nombre maximum de tampons qui seront écrits suite au parcours des prochains tampons à recycler.

L'acronyme **LRU** signifie « Least Recently Used », c'est-à-dire « Utilisé le moins récemment ». PostgreSQL détermine selon ce critère les tampons qui doivent être recyclés.

Écriture en tâche de fond (2/2)

- bgwriter_all_percent
  Pourcentage des tampons examinés à chaque tour.

  > Avec les valeurs par défaut de `bgwriter_delay` et `bgwriter_all_percent`, l'ensemble des tampons est parcouru une fois par minute.

  Ce paramètre disparaît en 8.3.
- bgwriter_all_maxpages
  Nombre maximum de tampons écrits suite au parcours de tous les tampons.

Des valeurs plus petites de `bgwriter_all_percent` et `bgwriter_all_maxpages` réduiront la charge supplémentaire en entrée/sortie causée par le processus d'écriture en tâche de fond mais laisseront plus de travail aux points de vérification.

Pour réduire les pointes de charge aux points de vérification, augmentez ces deux valeurs. Pour désactiver totalement le processus d'écriture en tâche de fond, configurez les deux valeurs `maxpages` et/ou les deux valeurs percent à 0.

Ce paramètre disparaît en 8.3.
- bgwriter_lru_multiplier
  Le nombre de tampons modifiés en mémoire durant la précédente période d'observation est multiplié par ce nombre pour estimer le nombre de tampons à écrire sur le disque pendant la prochaine période, et donc déterminer la quantité d'écritures que le bgwriter devra effectuer.

Disponible à partir de la version 8.3.

Des valeurs faibles de `bgwriter_lru_maxpages` et de `bgwriter_lru_multiplier` réduisent la charge des entrées/sorties disque induite par ce processus. En contrepartie, les processus serveur pourraient effectuer eux-même plus d'écriture, ce qui ralentit les requêtes interactives.

Paramétrage des journaux de transaction (1/2)

- `fsync`

  Force l'écriture physique sur le disque en exécutant les appels système `fsync()` après chaque transaction.

  L'activation de cette option dégrade les performances de la base de données, mais est le garant de l'intégrité des données en cas d'arrêt brutal de la machine. S'il est désactivé, une corruption dans ce cas est probable.

  Si vous désactivez cette option (`off`), considérez aussi la désactivation de `full_page_writes`.

- `synchronous_commit`

  Active la synchronisation automatique des journaux de transactions à chaque `COMMIT` (explicite et implicite).

  Lorsque ce paramètre est activé, PostgreSQL écrit sur disque au moment du COMMIT et ne renvoie le code de réussite ou d'échec une fois l'écriture terminée sur disque.

  Lorsque ce paramètre est désactivé, PostgreSQL renvoie immédiatement le code de retour. L'écriture se fait au pire dans 3*"wal_writer_delay", soit 600 ms par défaut).

- `wal_sync_method`

  Méthode utilisée pour forcer les mises à jour des journaux de transaction sur le disque.

  Les valeurs possibles sont :

  - `open_datasync` (écrit les journaux de transaction avec l'option `O_DSYNC` de `open()`)
  - `fdatasync` (appelle `fdatasync()` à chaque validation, conseillé par Sun sous Solaris)
  - `fsync_writethrough` (appelle `fsync()` à chaque validation, forçant une écriture de tous les caches disque)
  - `fsync` (appelle `fsync()` à chaque validation)
  - `open_sync` (écrit les journaux de transaction avec l'option `O_SYNC` de `open()`)

- `full_page_writes`

  Active la méthode d'écriture de pages complètes (c'est le mode par défaut). Le désactiver permet de réduire le volume d'écritures dans le journal (et donc améliorer les performances lors de modifications de données), mais entraîne un risque de pages 'fracturées', c'est à dire écrites partiellement sur le disque. Cela ne devrait être désactivé que sur les systèmes ne risquant pas de fracturer les pages (contrôlleur RAID avec un cache en écriture équippé de batterie, baie SAN).

Paramétrage des journaux de transaction (2/2)

- `wal_buffers`

  Nombre de tampons de page disque alloués en mémoire partagée pour les données des journaux de transaction.

  Sans unité, il s'agit de blocs de 8 Ko.

  Ce paramétrage doit être assez important pour contenir toutes les données des journaux de transaction engendrées par une transaction typique. En effet, ces données sont envoyées sur le disque à chaque validation de transaction.

- `wal_writer_delay`

  Durée d'endormissement en milliseconde du processus chargé des écritures des journaux de transaction sur disque.

  Nouveau paramètre de la version 8.3.

- `commit_delay`

  Délai entre l'enregistrement d'une validation dans le tampon `WAL` et le vidage du tampon sur le disque, en microsecondes.

  Un délai différent de zéro peut autoriser la validation de plusieurs transactions sur un seul appel système `fsync()`. Le délai est traité si au minimum un nombre **N** de transactions sont actives au moment où le processus serveur a écrit son enregistrement de validation.

  La valeur de **N** est définie par le paramètre `commit_siblings`.

- `commit_siblings`

  Nombre minimum de transactions ouvertes en même temps avant d'attendre le délai `commit_delay`.

  Une valeur plus importante rend plus probable le fait qu'au moins une autre transaction sera prête à valider pendant la durée du délai.

Points de vérification (1/2)

Des points de vérification (`checkpoint`) sont réalisés avec une certaine fréquence pour copier les tampons modifiés sur disque et garantir un point de reprise dans l'éventualité d'un incident affectant la base. Ils génèrent donc une activité disque par à-coup.

- `checkpoint_segments`

  Distance maximum entre des points de vérifications automatiques.

  Par défaut à 3. Cela signifie que le processus bgwriter s'occupe d'écrire tous les tampons sur disque à chaque fois que 48 Mo (3 x 16 Mo) ont été écrit dans les journaux de transaction. Sur un système très occupé par des opérations d'écriture, cela peut rapidement devenir un problème. À moins d'avoir une très petite configuration, une configuration de 10 segments permet d'avoir de meilleures performances. Pour des systèmes chargés en écriture, des valeurs entre 32 et 256 sont intéressantes. Néanmoins, les valeurs supérieures à 64 sont seulement utilisées dans le cas d'insertions/modifications massives.

- `checkpoint_timeout`

  Temps maximum entre des points de vérification automatiques en secondes.

Points de vérification (2/2)

- `checkpoint_completion_target`

  Fraction de l'écart entre deux checkpoints à utiliser pour réaliser le checkpoint précédent.

  Permet de répartir les écritures sur une fraction du temps restant jusqu'au prochain CHECKPOINT. La surcharge dûe aux écritures peut être encore lissée en augmentant ce paramètre à son maximum : 0,9. Dans ce cas, les écritures se feront pendant 90% du délai amenant au prochain CHECKPOINT. Comme le nombre d'écritures à réaliser pour le checkpoint est constant quelque soit la valeur de ce paramètre, moins d'écritures auront lieu chaque seconde.

- `checkpoint_warning`

  Active l'écriture d'un message dans les journaux applicatifs du serveur si les points de vérification causés par le remplissage des journaux de transaction surviennent dans un intervalle plus rapide que ce nombre de secondes.

Archivage

- `archive_mode`

  Active le mode d'archivage des journaux de transactions.

  Nouveau paramètre de la 8.3

- `archive_command`

  Commande shell pour exécuter l'archivage d'un journal de transaction complet.

  Si cette chaîne est vide (valeur par défaut), l'archivage des journaux de transaction est désactivé (y compris en 8.3, donc quelque soit la valeur de `archive_mode`). Tout `%p` dans la chaîne est remplacé par le chemin absolu vers le fichier à archiver ; et tout `%f` est seulement remplacé par le nom du fichier. Exemple :

  ```
  archive_command = 'cp "%p" /mnt/server/archivedir/"%f"'
  ```

- `archive_timeout`

  Force un changement de journal de transaction après que ce délai (en seconde) se soit écoulé.

  Si 0, fonctionnalité désactivée.

  Attention, un journal ne peut pas être activé s'il n'y a pas eu de `CHECKPOINT` depuis la fin du remplissage de ce journal. Pensez donc à configurer de façon appropriée le paramètre `checkpoint_timeout`.

Paramètres du planificateur

Les paramètres de configuration ci-dessous fournissent une méthode dure pour influencer les plans de requête choisis par l'optimiseur de requêtes.

Par défaut, tous ces paramètres sont activés :

```
enable_bitmapscan
enable_hashagg
enable_hashjoin
enable_indexscan
enable_mergejoin
enable_nestloop
enable_seqscan
enable_sort
enable_tidscan
```

PostgreSQL proprose de nombreuses options pour influencer de manière "dure" l'optimiseur de requête. Ainsi le paramètre enable_seqscan active ou désactive l'utilisation des parcours séquentiel par le planificateur. Cependant, il n'est pas possible de supprimer complètement les parcours séquentiels mais désactiver cette variable "décourage" le planificateur de l'utiliser si d'autres méthodes sont disponibles.

Il s'agit donc là d'un paramètre permettant d'étudier le comportement du planificateur ou de vérifier des hypothèses d'optimisation. Placer le paramètre enable_seqscan (et les autres paramètres du planificateur) à 'off' est donc une mesure **temporaire**, à effectuer sur un serveur de développement ou de prépoduction.

Par ailleurs notons qu'il est possible de définir cette valeur à un niveau de granularité plus fin. Ainsi il est possible de préciser la valeur de ce paramètre pour un utilisateur ou une base. Par exemple :

```
ALTER ROLE nom_utilisateur  SET enable_seqscan TO 'off';
```

Puis tester si les requêtes de cet utilisateur sont plus rapides que celles des autres utilisateurs.

Il est également possible de changer cela uniquement dans une session, ainsi le paramètre n'est modifié que pour la session en cours :

```
SET enable_seqscan = 'off';
```

Pour résumer, on peut dire que le paramètre enable_seqscan=off n'est pas un axe d'optimisation mais un outil d'optimisation. Et à ce titre, comme beaucoup d'outils d'optimisation, il ne doit pas être utilisé dans un environnement de production

Constantes de coût du planificateur (1/2)

- `effective_cache_size`

  Configure **l'idée** du planificateur sur la taille réelle du cache disque disponible pour un simple parcours d'index.

  La valeur est mesurée en pages disque, qui sont normalement de `8192 octets`.

  Pour un serveur dédié utilisant *Linux*, il est d'usage de positionner ce paramètre entre la moitié et deux tiers de la mémoire centrale. En effet, le système *Linux* utilise intensivement la mémoire pour le cache disque.

  Une meilleure estimation est possible en recherchant dans les statistiques de votre système d'exploitation. Sur les systèmes Unix, ajoutez les nombres free+cached provenant des outils `top` ou `free`. Sur Windows, voir « System Cache » dans l'onglet « Performance » du gestionnaire des tâches.

  Ce paramètre permet à PostgreSQL d'affiner les coûts, en termes d'opérations disques, des plans d'exécution, par rapport à la probabilité que les données soient en cache.

- `seq_page_cost`

  Estimation du coût du planificateur pour une page disque récupérée de façon séquentielle.

  Nouveau paramètre de la 8.3.

- `random_page_cost`

  Estimation du coût du planificateur pour une page disque récupérée de façon non séquentielle.

  Une valeur plus haute rend plus probable l'utilisation d'un parcours séquentiel, une valeur basse l'utilisation d'un parcours d'index.

  3 est la valeur la plus communément appliquée pour ce paramètre.

Constantes de coût du planificateur (2/2)

- `cpu_tuple_cost`

  Estimation du coût du planificateur pour le traitement de chaque ligne lors d'une requête.
  Même unité de mesure arbitraire que seq_page_cost.

- `cpu_index_tuple_cost`

  Estimation du coût du planificateur pour le traitement de chaque ligne lors d'un parcours d'index.
  Même unité de mesure arbitraire que seq_page_cost.

- `cpu_operator_cost`

  Estimation du coût du planificateur de requêtes pour le traitement de chaque opérateur dans une clause `WHERE`.
  Même unité de mesure arbitraire que seq_page_cost.

Diverses options du planificateur

- `default_statistics_target`

  Finesse des statistique à collecter (taille des histogrammes) pour chaque colonne lors d'un `ANALYZE`. Détermine aussi la finesse d'échantillonnage.

  Par défaut, PostgreSQL calcule des histogrammes grossiers (10 entrées). Un consensus s'est dégagé durant le cycle de développement de la version 8.4 montrant que le bon compromis par défaut était aux alentours de 100 : les plans d'exécution sont meilleurs, sans engendrer de surcoût mesurable au moment du calcul des statistiques ou du plan d'exécution. Si vous obtenez de mauvais plans d'exécution, pensez à augmenter cette valeur, puis à exécuter un ANALYZE.

  La valeur par défaut est passée de 10 à 100 en 8.4. La valeur maximum est passée de 1000 à 10000 en 8.4.

  Ce paramètre contrôle aussi directement le nombre de MCV (most common values, valeurs les plus fréquentes relevées dans les statistiques), ainsi que la finesse d'échantillonnage sur la table (nombre d'enregistrements pris aléatoirement pour calculer les statistiques).

- `constraint_exclusion`

  Active l'utilisation des contraintes de tables pour l'optimisation des requêtes.

  Intéressant dans le cas des tables partitionnées.

  Booléen strict sur les versions antérieures à la 8.4. En 8.4, il est configuré par défaut à `partition`, ce qui veut dire que les contraintes sont vérifiées uniquement pour les tables enfants et pour les sous-requêtes en UNION ALL.

- `cursor_tuple_fraction`
  Estimation de la fraction de données réellement lues à partir d'un curseur.

  Disponible à partir de la 8.4.

  La valeur par défaut, 10%, correspond à la valeur en dur pour les anciennes versions de PostgreSQL.

- `from_collapse_limit`

  Le planificateur assemble les sous-requêtes dans des requêtes supérieures tant que la liste FROM résultante ne contient par plus que ce nombre d'éléments.

  À conserver inférieur à la valeur de geqo_threshold.

- `join_collapse_limit`

  Le planificateur réécrit les constructions JOIN explicites (à l'exception de FULL JOIN) en une liste d'éléments FROM tant que la liste résultante ne contient pas plus que ce nombre d'éléments.

Optimiseur génétique de requêtes (1/2)

- `geqo`

  Active ou désactive l'optimiseur génétique de requêtes.

- `geqo_threshold`

  Utilise l'optimisation génétique des requêtes pour des requêtes comportant dans leur clause `FROM` un nombre d'éléments au moins égal à ce paramètre.

  Pour des requêtes simples, il est généralement préférable d'utiliser le planificateur déterministe, mais pour les requêtes comprenant beaucoup de tables, le planificateur déterministe prendrait trop de temps.

- `geqo_effort`

  Contrôle le compromis entre le temps de planification et l'efficacité du plan de requête dans `GEQO`.

  Les valeurs plus importantes augmentent le temps passé pendant la planification de la requête mais augmentent aussi la possibilité qu'un plan de requête efficace soit choisi.

Optimiseur génétique de requêtes (2/2)

- `geqo_pool_size`

  Contrôle la taille de la queue utilisée par `GEQO`.

  Cette taille est le nombre d'individus dans une population génétique. Elle doit être d'au moins deux, et les valeurs utiles sont typiquement `100` et `1000`.

  Si elle est configurée à zéro (la valeur par défaut), alors une valeur par défaut convenable est choisie suivant `geqo_effort` et le nombre de tables dans la requête.

- `geqo_generations`

  Contrôle le nombre de générations utilisé par `GEQO` (nombre d'itérations de l'algorithme).

  Si ce paramètre est positionné à 0, une valeur par défaut convenable est choisie en fonction de `geqo_pool_size`.

- `geqo_selection_bias`

  Contrôle le biais de sélection utilisé par `GEQO`.

  Pression de sélection à l'intérieur de la population. Les valeurs peuvent aller de `1,50` à `2,00`.

Configuration des logs (1/9)

- `log_destination`

  Journalisation des messages du serveur (`stderr`, `eventlog`, `syslog` et `csvlog`).

  `eventlog` sert sous Windows pour envoyer les messages du serveur dans le journal des événements Windows.

  `csvlog` n'est disponible qu'à partir de la version 8.3 et permet l'enregistrement des messages au format CSV.

- `redirect_stderr` (renommé `logging_collector` en 8.3)

  Autorise la capture et la redirection des messages envoyés vers `stderr` dans des journaux de traces.

Configuration des logs (2/9)

- `log_directory` et `log_filename`

  Répertoire et nom de fichiers dans lequel les journaux seront créés.

- `log_rotation_age`

  Durée de vie maximum d'un journal individuel.

- `log_rotation_size`

  Taille maximum (sauf précision, en octets) d'un journal individuel.

- `log_truncate_on_rotation`

  Tronque un journal si ce dernier existe déjà avec le même nom.

Configuration des logs (3/9)

- `syslog_facility`

  Détermine le niveau (*facility*) utilisé par syslog.

- `syslog_ident`

  Nom du programme utilisé pour identifier les messages de PostgreSQL dans les journaux de traces de syslog.

Configuration des logs (4/9)

Il existe de nombreux niveaux de trace. Chaque niveau inclut tous les niveaux qui le suivent. La valeur par défaut est `NOTICE`.

- `client_min_messages`

  Niveau des messages envoyés au client.

Les niveaux de traces peuvent être `DEBUG5`, `DEBUG4`, `DEBUG3`, `DEBUG2`, `DEBUG1`, `LOG`, `NOTICE`, `WARNING`, `ERROR`.

- `log_min_messages`

  Niveau des messages écrits dans les journaux de traces.

Les niveaux de traces peuvent être `DEBUG5`, `DEBUG4`, `DEBUG3`, `DEBUG2`, `DEBUG1`, `INFO`, `NOTICE`, `WARNING`, `ERROR`, `LOG`, `FATAL` et `PANIC`.

- `log_error_verbosity`

  Niveau de détails écrit dans les journaux de traces pour chaque message tracé.

  Les valeurs valides sont TERSE, DEFAULT et VERBOSE, chacune ajoutant plus de champs aux messages affichés.

Configuration des logs (5/9)

- `log_min_error_statement`

  Niveau d'erreur au dessus duquel un ordre `SQL` ayant généré une erreur sera tracé.

- `log_min_duration_statement`

  Trace l'instruction et sa durée si son exécution dure au moins ce nombre de millisecondes.
  Configurer cette valeur à zéro affichera toutes les instructions avec leur durée. `-1` (la valeur par défaut) désactive cette fonctionnalité.

- `silent_mode`

  Si activé, aucun message d'erreur n'est engendré.

- `log_connections`, `log_disconnections`

  Tracent respectivement chaque connexion réussie, chaque déconnexion et la durée des sessions.

Configuration des logs (6/9)

- `log_duration`

  Trace la durée d'une instruction terminée satisfaisant `log_statement`.

- `log_line_prefix`

  Chaîne affichée au début de chaque ligne. Exemple : `'%t %d %u [%p] '`
  Utile principalement si `log_destination` vaut `stderr`.

- `log_statement`

  Contrôle les instructions `SQL` tracées.
  Les valeurs valides sont `none`, `ddl`, `mod` et `all`. `ddl` trace toutes les commandes de définition comme `CREATE`, `ALTER` et `DROP`. `mod` trace toutes les instructions `ddl`, plus `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE` et `COPY FROM`. Les instructions `PREPARE` et `EXPLAIN ANALYZE` sont aussi tracées si leur commande contenue est d'un type approprié.

- `log_hostname`

  Trace du nom de l'hôte (utilise la résolution de nom).

Configuration des logs (7/9)

Ces paramètres sont apparus avec la version 8.3.

- `log_checkpoints`

  Trace l'exécution de points de vérification.

- `log_lock_waits`

  Trace les verrous qui durent plus de `deadlock_timeout`.

- `log_temp_files`

  Trace les fichiers temporaires dont la taille dépasse ce nombre (en Ko).
  Une valeur de -1 a pour effet de désactiver cette trace.

- `log_timezone`

  Fuseau horaire utilisé pour l'horodatage des traces.

Configuration des logs (8/9)

- `debug_print_parse`
- `debug_print_rewritten`
- `debug_print_plan`
- `debug_pretty_print`

  Activent plusieurs sorties de débogage.

Ces drapeaux activent la sortie d'informations de débogage vers les journaux du serveur. Pour chaque requête exécutée, écrit, respectivement, le texte de la requête, l'arbre syntaxique résultant, la sortie du rédacteur de requête ou le plan d'exécution. `debug_pretty_print` indente les affichages pour faciliter la lecture, au prix d'une écriture plus longue.

Ces options sont utiles pour détecter les requêtes lentes, sous réserve de réussir à parcourir un journal volumineux. Particulièrement utiles dans un mode interactif de surveillance des journaux applicatifs lorsque les procédures stagnent ; il est parfois possible de voir à quel endroit la procédure stagne (parfois, cela n'est pas possible, parce que le journal attend une information de la base).

Configuration des logs (9/9)

- `log_statement_stats`

  Rapporte des statistiques complètes sur les requêtes exécutées : utilisation CPU, utilisation I/O, défauts de page, utilisation des tampons.

  Exemple de trace pour l'instruction CREATE TABLE :

```
LOG :  instruction : create table t1 (id integer) ;
LOG :  QUERY STATISTICS
DÉTAIL : ! system usage stats :
        !       0.181959 elapsed 0.008000 user 0.004000 system sec
        !       [0.012000 user 0.004000 sys total]
        !       736/176 [904/176] filesystem blocks in/out
        !       0/211 [0/872] page faults/reclaims, 0 [0] swaps
        !       0 [0] signals rcvd, 0/0 [0/0] messages rcvd/sent
        !       39/0 [50/0] voluntary/involuntary context switches
        ! buffer usage stats :
        !       Shared blocks :         34 read,          0 writ-
ten, buffer hit rate = 82.83%
        !       Local  blocks :          0 read,          0 writ-
ten, buffer hit rate = 0.00%
        !       Direct blocks :          0 read,          0 written
INSTRUCTION :  create table t1 (id integer) ;
```

- `log_parser_stats`, `log_planner_stats`, `log_executor_stats`

  Active ou désactive les statistiques pour chacun des modules.

Configuration des statistiques (1/3)

- `stats_start_collector` (renommé `track_activities` en 8.3)

  Contrôle du lancement du processus de récupération des statistiques.

- `stats_command_string` (renommé `update_process_title` en 8.3)

  Active la récupération de statistiques sur les commandes en cours d'exécution par chaque session.
  La donnée est accessible via la vue système `pg_stat_activity` pour les utilisateurs possédant la session et les super-utilisateurs.

- `track_activity_query_size` (disponible à partir de la 8.4)

  Indique la taille maximum de la requête affichée par `pg_stat_activity`.
  Par défaut, 1024 (valeur utilisée en dur dans les versions précédentes).

Configuration des statistiques (2/3)

- `stats_block_level` (renommé `track_counts` en 8.3)

  Active la récupération des statistiques au niveau bloc sur l'activité de la base de données.
  Les données produites sont accessibles via la famille de vues système `pg_statio`.

- `stats_row_level` (renommé `track_counts` en 8.3)

  Active la récupération de statistiques au niveau ligne sur l'activité de la base de données.
  Les données produites sont accessibles via la famille de vues système `pg_stat`.
  En 8.3, `stats_block_level` et `stats_row_level` sont assemblés pour ne proposer plus qu'un seul paramètre, `track_counts`.

- `stats_reset_on_server_start`

  Les statistiques récupérées sont vidées à chaque fois que le serveur est redémarré.
  Disparaît à partir de la version 8.3.

Configuration des statistiques (3/3)

- `track_functions`

  Active la récupération des statistiques pour les fonctions.
  Disponible à partir de la version 8.4.
  Trois valeurs possibles : `none`, `pl`, `all`.
  Les données produites sont accessibles via la vue système `pg_stat_user_functions`.

- `stats_temp_directory`

  Précise l'emplacement du répertoire de stockage temporaire des statistiques. Permet de placer ce fichier sur un répertoire en RAM.

VACUUM automatique (1/3)

- `autovacuum`

  Activation/désactivation du processus autovacuum.

  `stats_start_collector` (`track_activities` en 8.3) et `stats_row_level` (`track_counts` en 8.3) doivent aussi être actifs pour que ce démon soit exécuté.

- `log_autovacuum_min_duration`

  Trace la durée de l'autovacuum si son exécution dure au moins ce nombre de millisecondes.

  Configurer cette valeur à zéro affichera toutes les actions de l'autovacuum avec leur durée. `-1` (la valeur par défaut) désactive cette fonctionnalité.

  Paramètre disponible à partir de la 8.3

- `autovacuum_max_workers`

  Nombre maximum de sous-processus autovacuum exécutés à un instant `t`.

  Paramètre disponible à partir de la 8.3.

  Plusieurs « autovacuum workers » peuvent travailler en même temps. Ils peuvent même travailler sur la même base en même temps grâce à une mémoire partagée qui leur permet de savoir sur quelle table/index un autre « autovacuum worker » est en train de travailler.

  Attention, chaque « worker » peut s'attribuer de la mémoire jusqu'à la limite imposée par « maintenance_work_mem ».

- `autovacuum_naptime`

  Délai (en secondes) entre les tours d'activité pour le sous-processus autovacuum.

  En 8.2, autovacuum est un processus exécuté tous les autovacuum_naptime secondes alors qu'en 8.3, le processus « autovacuum launcher » est un démon qui exécute un « autovacuum worker » sur toutes les bases tous les `autovacuum_naptime` secondes.

VACUUM automatique (2/3)

- autovacuum_vacuum_threshold

  Nombre minimum de lignes mises à jour ou supprimées nécessaire pour déclencher un VACUUM sur une table.

- autovacuum_analyze_threshold

  Nombre minimum de lignes insérées, mises à jour ou supprimées pour déclencher une commande ANALYZE sur une table.

- autovacuum_vacuum_scale_factor

  Fraction de la taille de la table à ajouter à autovacuum_vacuum_threshold pour décider du moment pour déclencher un VACUUM.

- autovacuum_analyze_scale_factor

  Fraction de la taille de la table à ajouter à autovacuum_analyze_threshold pour décider de déclencher une commande ANALYZE.

VACUUM automatique (3/3)

- autovacuum_vacuum_cost_delay

  Délai d'attente une fois que la limite autovacuum_vacuum_cost_limit est atteinte dans les opérations de VACUUM.

  En 8.3, ce délai est partagé entre tous les « autovacuum workers » en cours d'exécution.

  Sans précision, ce délai est en millisecondes.

- autovacuum_vacuum_cost_limit

  Valeur limite du coût utilisée dans les opérations de VACUUM automatiques.

  Les paramètres suivants peuvent être surchargés individuellement pour chaque table :

  - par des entrées dans le catalogue système pg_autovacuum pour les versions antérieures à la version 8.4 ;
  - par les options de stockage de chaque table pour la version 8.4.

- autovacuum_freeze_max_age

  Age maximum du XID avant de forcer un VACUUM.

  Ce VACUUM survient même si l'autovacuum est désactivé.

Valeurs par défaut des connexions client (1/3)

- `search_path`

  Ordre dans lequel les schémas sont recherchés lorsqu'un objet est référencé par un simple nom sans son composant schéma.

- `default_tablespace`

  Tablespace par défaut dans lequel sont créés les objets (tables et index).

- `temp_tablespaces`

  Tablespaces par défaut dans lequel sont créés les objets temporaires (tables et index, mais aussi les fichiers temporaires pour les tris). Les tablespaces sont utilisés à tour de rôle, dans chaque session.

- `check_function_bodies`

  Active/désactive la validation du corps des fonctions lors de la création.

Valeurs par défaut des connexions client (2/3)

- `default_transaction_isolation`

  Contrôle le niveau d'isolation par défaut de chaque nouvelle transaction (« read uncommitted », « read committed », « repeatable read » ou « serializable »).

- `default_transaction_read_only`

  Si activé, bloque l'accès en écriture aux tables non temporaires.

- `statement_timeout`

  Annule toute instruction prenant plus que le nombre spécifié de millisecondes.
  Une valeur à 0 désactive le timeout.

Valeurs par défaut des connexions client (3/3)

- `session_replication_role`
  Contrôle le déclenchement des triggers et des règles pour la session en cours.
  Paramètre disponible à partir de la 8.3
- `vacuum_freeze_max_age`
  Indique l'âge limite (en nombre de transactions) que `VACUUM` utilise pour remplacer les identifiants de transactions par FrozenXID.
- `vacuum_freeze_table_age`
  Contrôle le déclenchement d'un parcours complet de la table par un `VACUUM` en indiquant l'âge maximum de la table (à comparer à `pg_class.relfrozenxid`).
  Paramètre disponible à partir de la 8.4
- `xmlbinary`
  Codage des valeurs binaires en XML. Par défaut, `base64`.
  Paramètre disponible à partir de la 8.3
- `xmloption`
  Configure si une chaîne de caractère convertie implicitement en `XML` est un document (`DOCUMENT`) ou un fragment (`CONTENT`).
  Paramètre disponible à partir de la 8.3

Locale et formatage (1/2)

- `datestyle`

  Configure le format d'affichage et les règles d'interprétation pour les valeurs de type date et heure.

  Les mots clés Euro et European sont un synonyme pour DMY ; les mots clés US, NonEuro et NonEuropean sont des synonymes pour MDY.

- `timezone`

  Fuseau horaire pour l'affichage et l'interprétation de la date et de l'heure.

- `timezone_abbreviations`

  Liste des abrévitations de fuseaux horaires acceptés par le serveur.

- `extra_float_digits`

  Nombre de chiffres affichés pour les valeurs à virgule flottante.

  Ce paramètre permet d'ajuster le nombre de chiffres affichés pour les valeurs à virgule flottante, ce qui inclut les float4, float8 et les types de données géométriques.

  La valeur du paramètre est ajoutée au nombre standard de chiffres (FLT_DIG ou DBL_DIG, selon les cas). La valeur 2, la plus haute possible, permet d'inclure les chiffres partiellement significatifs ; cela est particulièrement utile pour copier des flottants dont la restauration doit être exacte. Les valeurs négatives permettent de supprimer les chiffres non souhaités.

Locale et formatage (2/2)

- `client_encoding`

  Codage côté client (par défaut celui de la base de données).

- `lc_messages`, `lc_monetary`, `lc_numeric`, `lc_time`

  Locales utilisées pour l'affichage des messages, des montants de monnaie, des nombres et des valeurs de date et d'heure.

- `default_text_search_config`

  Configuration par défaut de la recherche plein texte.
  Paramètre disponible à partir de la 8.3

Gestion des verrous

- `deadlock_timeout`

  Temps total, en millisecondes, d'attente d'un verrou avant de vérifier s'il s'agit d'un deadlock.

  La vérification d'un deadlock est assez lente, donc le serveur ne le fait pas à chaque fois qu'il attend pour un verrou. Augmenter cette valeur réduit le temps perdu en recherche inutile de verrous morts mais ralentit la détection de vraies erreurs de deadlocks.

  Idéalement, ce paramétrage devrait dépasser le temps typique d'une transaction de façon à améliorer la probabilité qu'un verrou soit abandonné avant que le processus en attente ne décide de lancer une recherche de deadlock.

- `max_locks_per_transaction`

  Nombre de verrous maximal par transaction.

  Cette valeur est utilisée pour allouer l'espace nécessaire pour la table des verrous partagés.

  Une valeur trop basse force PostgreSQL à utiliser plus de mémoire partagée si la limite est atteinte, ce qui peut mener à des erreurs de type « Mémoire partagée insuffisante » (« not enough shared memory »). Cependant, une valeur trop haute augmente la probabilité de situation de « deadlock » ainsi que l'espace mémoire attribué à la table des verrous (qui se calcule avec la formule ci-dessous). Il convient donc de contrebalancer cette valeur en diminuant la valeur du paramètre `max_connections` si possible.

  Taille (en octets) en mémoire partagée pour la table des verrous :

  ```
  ~ 270 x (MLPT x MC + MPT)
  ```

  avec :
  - MLPT = max_locks_per_transaction
  - MC = max_connections
  - MPT = max_prepared_transactions

Compatibilité de version (1/3)

Ces options servent à conserver une compatibilité avec des versions antérieures lors des migrations.

- `add_missing_from`

  Les tables référencées par une requête seront automatiquement ajoutées à la clause `FROM` si elles n'y sont pas déjà présentes.

- `arrays-nulls`

  Contrôle si la saisie de tableau reconnaît les NULL sans guillemets dans des éléments de tableaux.

  Paramètre disponible à partir de la 8.2.

- `regex_flavor`

  « niveau » des expressions rationnelles (`advanced`, `extended`, ou `basic`).

  Le paramétrage `extended` est utile pour conserver une compatibilité exacte avec les versions précédant la 7.4.

- `sql_inheritance`

  Contrôle la sémantique de l'héritage, en particulier si les sous-tables sont incluses par les différentes commandes par défaut.

  Pas de contrôle pour les versions antérieures à la 7.1. Il est recommandé d'utiliser le mot clé ONLY qui exclut les sous-tables.

Compatibilité de version (2/3)

- `backslash_quote`

  Contrôle si un guillemet peut être représenté par un \' dans une chaîne.

  Dans le standard SQL, pour représenter un guillemet, il faut le doubler (") mais historiquement, PostgreSQL accepte aussi \'.

- `default_with_oids`

  Contrôle si les commandes `CREATE TABLE` et `CREATE TABLE AS` incluent une colonne `OID` dans les tables nouvellement créées.

- `escape_string_warning`

  Un message d'avertissement est affiché si un antislash (\) apparaît dans une chaîne litérale ordinaire.

- `transform_NULL_equals`

  Si cette option est activée, les expressions de la forme `expr = NULL` (ou `NULL = expr`) sont traitées comme expr `IS NULL`.

Compatibilité de version (3/3)

- `synchronize_seqscans`

  Contrôle l'activation des parcours séquentiels synchronisés.

  Les parcours séquentiels synchronisés ne sont disponibles qu'en version 8.3. Leur utilisation modifie l'ordre des données récupérées... du coup, une sauvegarde via pg_dump sera différente suivant l'activation ou non de cette option.

Options pré-configurées (1/3)

Les « paramètres » suivant sont en lecture seule et sont déterminés lors de la compilation (`block_size`, `max_function_args`, `max_identifier_length`, `max_index_keys`, `server_version`) ou de l'installation de PostgreSQL (`lc_collate`, `lc_ctype`, `server_encoding`).

- `block_size`

  Taille d'un bloc disque pour un fichier de données. Elle est déterminée par la valeur de `BLCKSZ` à la construction du serveur.

- `wal_block_size`

  Taille d'un bloc disque pour un journal de transactions.
  Disponible à partir de la 8.4.

- `segment_size`

  Taille d'un segment pour un fichier de données.

- `wal_segment_size`

  Taille d'un segment pour un journal de transactions.
  Disponible à partir de la 8.4.

- `integer_datetimes`

  Support des dates et heures sur des entiers de 64 bits.
  Activé par défaut en 8.4. Désactivé par défaut pour les versions antérieures.

Options pré-configurées (2/3)

- `lc_collate`, `lc_ctype`

  Locales utilisées pour le tri des données de type texte et les classifications de caractères.

- `max_function_args`

  Nombre maximum d'arguments des fonctions.

- `max_identifier_length`

  Longueur maximum d'un identifiant.

- `max_index_keys`

  Nombre maximum de clés d'index.

Options pré-configurées (3/3)

- `server_encoding`

  Encodage par défaut du cluster et des nouvelles bases.

- `server_version`

  Numéro de version du serveur.

- `standard_conforming_strings`

  Rapporte si des chaînes littérales ordinaires (`'...'`) traitent les *antislashs* de façon littérale comme spécifié dans le standard `SQL`.

Authentification du client

Lorsqu'un client se connecte au serveur, il utilise :
- un nom d'utilisateur ;
- un moyen de connexion (adresse IP ou socket Unix) ;
- un nom de base de données ;
- un mot de passe (optionnel).

PostgreSQL propose des méthodes d'authentifications spécifiques pour chacun des clients.

La politique d'authentification est consignée dans le fichier `pg_hba.conf`.

Configuration du fichier pg_hba.conf (1/4)

Chaque enregistrement détermine :

- un type de connexion ;
- une plage d'adresses IP (si approprié au type de connexion) ;
- un ou plusieurs noms de base de données ;
- un ou plusieurs noms d'utilisateurs, de groupe ;
- une méthode d'authentification à utiliser.
  Le premier enregistrement qui correspond aux critères est utilisé pour effectuer l'authentification.
  D'autre part, il est possible d'utiliser le caractère @ pour inclure un fichier contenant une liste de noms.

Configuration du fichier pg_hba.conf (2/4)

Un enregistrement peut avoir l'un des formats suivants :

```
local      database  user  auth-method  [auth-option]
host       database  user  CIDR-address  auth-method  [auth-option]
hostssl    database  user  CIDR-address  auth-method  [auth-option]
hostnossl  database  user  CIDR-address  auth-method  [auth-option]
host       database  user  IP-address  IP-mask  auth-method  [auth-option]
hostssl    database  user  IP-address  IP-mask  auth-method  [auth-option]
hostnossl  database  user  IP-address  IP-mask  auth-method  [auth-option]
```

La version `8.2` de PostgreSQL supporte désormais l'authentification par `LDAP`. Elle se configure comme suit dans le fichier `pg_hba.conf` :

```
ldap[s] ://nomduserveur[ :port]/base dn[ ;prefix[ ;suffix]]

ldap ://ldap.exemple.net/dc=exemple,dc=net ;EXEMPLE\
```

Pour utiliser SSL, vous devez avoir configuré `ssl` à `on`.

Configuration du fichier pg_hba.conf (3/4)

- `local`

  Intercepte les tentatives de connexion utilisant les sockets du domaine Unix.

- `host`

  Intercepte les tentatives de connexion utilisant les réseaux TCP/IP (SSL et non SSL).

- `hostssl`

  Intercepte les tentatives de connexion utilisant TCP/IP avec le chiffrement SSL seulement.

- `hostnossl`

  Intercepte les tentatives de connexion utilisant TCP/IP sans le chiffrement SSL seulement.

Configuration du fichier pg_hba.conf (4/4)

- `database`

  Noms de bases de données (all pour toutes les bases).

  La valeur `sameuser` spécifie que l'enregistrement n'intercepte que si la base de données demandée a le même nom que l'utilisateur demandé.

  La valeur `samerole` spécifie que l'utilisateur demandé doit être membre du rôle portant le même nom que la base de données demandée

  Des noms de bases de données multiples peuvent être fournis en les séparant par des virgules.

  Un fichier séparé contenant des noms de bases de données peut être indiqué en faisant précéder le nom de fichier de @.

- `user`

  Utilisateurs ou groupes de la base de données

  Précéder le nom des groupes par un '+'. Plusieurs noms peuvent être fournis en les séparant avec des virgules.

- `CIDR-address`

  Echelle d'adresses IP du client.

  Exemple pour une machine :

  ```
  192.168.143.37/32
  ```

  Pour un réseau :

  ```
  192.68.143.0/24
  ```

- `IP-address, IP-mask`

  Alternative à la notation CIDR-address.

  Exemple :

  ```
  192.168.143.37,255.255.255.255
  ```

- `auth-method`

  Méthode d'authentification à utiliser lors de la connexion.

- `auth-option`

  Champ optionnel dépendant de la méthode d'authentification choisie.

Méthodes d'authentification (1/3)

- `trust`

  Autorise la connexion sans conditions.

- `reject`

  Rejette la connexion sans conditions.

- `md5`

  Demande au client de fournir un mot de passe chiffré MD5.

- `crypt`

  Demande au client de fournir un mot de passe chiffré avec `crypt()`.
  Cette option est seulement recommandée pour pouvoir communiquer avec les clients de version antérieure à la 7.2.
  Disparaît en 8.4.

Méthodes d'authentification (2/3)

- `password`

  Demande au client de fournir un mot de passe non chiffré.

- `krb5`

  Utilise Kerberos V5 pour authentifier l'utilisateur. Ceci n'est disponible que pour les connexions TCP/IP.

- `GSSAPI`

  Protocole du standard de l'industrie pour l'authentification sécurisée définie dans RFC 2743. PostgreSQL supporte GSSAPI avec l'authentification Kerberos suivant la RFC 1964. Permet le « Single Sign-On ». Disponible à partir de la 8.3.

- `LDAP`

  Utilise LDAP pour authentifier l'utilisateur. Ceci n'est disponible qu'à partir de la version 8.2.

Méthodes d'authentification (3/3)

- `ident`

  Permet d'associer les noms des utilisateurs du système d'exploitation aux noms des utilisateurs du système de gestion de bases de données.

  Récupère le nom de l'utilisateur du système d'exploitation du client et vérifie si l'utilisateur est autorisé à se connecter en tant qu'utilisateur de la base de données. Cette méthode nécessite un paramètre supplémentaire qui est la correspondance indiquée après le mot clé ident.

  Le mot clé `sameuser` permet à n'importe quel utilisateur du système d'exploitation de se connecter en tant qu'utilisateur de base de données du même nom. Ce mot clé disparaît en 8.4 où cette option est la valeur par défaut si aucun `usermap` n'est défini.

  Les correspondances d'identité autres que sameuser sont définies dans le fichier de correspondance d'identité, `pg_ident.conf`. Ce fichier contient des lignes de la forme suivante

  ```
  nom-correspondance nomutilisateur-ident base-donnee-utilisateur
  ```

  Un démon fournissant le service `ident` est nécessaire.

- `pam`

  Authentifie en utilisant les Pluggable Authentication Modules (PAM) fournis par le système d'exploitation.

Correspondance de clients (1/3)

- Pour les authentifications externes telles que Ident ou GSSAPI
- Table de correspondance d'identités pour faire correspondre le nom d'utilisateur système au nom d'utilisateur base de donnée
- map=nom-table dans le champ options de pg_hba.conf
- Tout le reste se fait dans le fichier `pg_ident.conf`

Correspondance de clients (2/3)

Chaque enregistrement détermine :
- le nom de la table de correspondance ;
- le nom de l'utilisateur système ;
- le nom de l'utilisateur base de données.

Les expressions rationnelles sont acceptées dans le second et le troisième champs.

Correspondance de clients (3/3)

- Voici un exemple de fichier pg_ident.conf :

```
# MAPNAME     SYSTEM-USERNAME   PG-USERNAME
pg1           jean              jean
pg1           pierre            pierre
pg1           pierre            jean
pg1           charles           boss
pg1           /^(.*)@dalibo$    \1
```

- Jean et Pierre peuvent se connecter avec leur nom
- Pierre peut aussi se connecter en tant que Jean
- Charles peut se connecter en tant que boss, mais pas en tant que charles
- Pour la dernière ligne, quelquun@dalibo se connecte en tant que quelquun

Pour aller plus loin

- « Gestion mémoire avec PostgreSQL » : http ://dalibo.org/glmf107_gestion_memoire_avec_postgresql
- « Nouvelle gestion des journaux applicatifs sous PostgreSQL 8.3 » http ://dalibo.org/glmf105_nouvelle_gesti
- Un postgresql.conf complètement commenté !

En anglais (version 8.0) :

```
http ://www.powerpostgresql.com/Downloads/annotated_conf_80.html
```

En français (version 8.1) :

```
http ://docs.postgresql.fr/annotated_postgresql_conf_81.html
```

En anglais (version 8.3) :

```
http ://www.pgexperts.com/document.html?id=1
```

Conclusion

- Paramétrer un serveur PostgreSQL, au minimum :
  - ⇒ `shared_buffer`, `work_mem`, `maintenance_work_mem`
  - ⇒ `max_fsm_pages` et `max_fsm_relations` (pour les versions antérieures à la 8.4)
  - ⇒ `effective_cache_size`, `checkpoint_segments`
  - ⇒ `log_line_prefix`, `stats_start_collector`
- Attention à l'*autovacuum*, en particulier sous *debian* (dans certaines versions, un script `cron` est installé par défaut et génère un `VACUUM` tous les jours à cinq heures)
- Régler avec soin `pg_hba.conf`

Questions

N’hésitez pas, c’est le moment !