tellora

Administration

Base de données

PostgreSQL

Exploitation

Exploitation d'une instance

- Création de tablespaces
  - L'utilisation de tablespaces permet à une instance de stocker les fichiers à un autre emplacement
    - CREATE TABLESPACE nom_TS [OWNER nomrole] LOCATION 'nom_repertoire' ;
      - Par défaut le tablespace appartient à l'utilisateur qui exécute la commande
      - Seul un super user peut créer un tablespace (mais il peut transmettre le privilège à un autre user)
  - Le tablespace sera utilisable au moment de la création (ou modification) des tables et des index

Exploitation d'une instance

- Modification d'un tablespace
  - Il est possible de modifier le nom d'un tablespace ou le propriétaire du tablespace
    - ALTER TABLESPACE nom_TS RENAME TO nouveau_nom ;
    - ALTER TABLESPACE nom_TS RENAME OWNER TO nouveau_role ;

Exploitation d'une instance

- Suppression d'un tablespace
  - La suppression d'un tablespace peut se faire à condition que celui-ci soit vide
    - DROP TABLESPACE [IF EXISTS] nom_TS ;
      - IF EXISTS = évite l'apparition d'une erreur si celui-ci n'existe pas
  - Une fois que le tablespace est supprimé il est possible de supprimer le répertoire qui y était rattaché

Exploitation d'une instance

- Dans le répertoire « base » PostgreSQL crée un répertoire par nouvelle database créée
  - Le nom du répertoire correspond à un identifiant référencé et associé à la database dans le catalogue système
    - Select oid, datname from pg_catalog.pg_database ;
  - Dans chaque répertoire de database les fichiers qui correspondent aux tables et aux index sont nommés avec un identifiant qui lui est associé
    - Select relfilenode, relname from pg_catalog.pg_class ;
    - Si la taille d'un fichier dépasse 1 giga, PostgreSQL crée un nouveau fichier en utilisant le : relfilenode + numéro séquentiel
      - Relfilenode.1 puis relfinode.2

Exploitation d'une instance

- Certaines tâches de maintenance doivent être réalisées régulièrement dans PostgreSQL
  - Ces tâches peuvent être automatisées
  - Elles participent à l'optimisation du serveur
- Les tâches à effectuées sont :
  - Le « vacuum » ou nettoyage réguliers
  - La ré-indexation
  - Maintenance du fichier de traces

Exploitation d'une instance

- « Vacuum » ou nettoyage régulier
  - La commande VACUUM doit être effectuer régulièrement pour :
    - Récupérer ou ré-utiliser l'espace disque occupé par les lignes supprimées ou modifiées
    - Mettre à jour les statistiques utilisée par l'optimiseur
    - Mettre à jour la carte de visibilité qui accélère les parcours d'index seuls
    - Prévenir la perte des données les plus anciennes due à un cycle de l'identifiant de transaction (XID)
  - ATTENTION : cette commande peut s'effectuer pendant la production elle n'a pas d'impact sur les commandes SQL insert, update, delete mais a un impact sur les commandes ALTER , par contre elle est très consommatrice de CPU
    - Donc préférez une exécution en dehors de la production si possible

Maintenance

- Récupérer l'espace disque
  - Dans son fonctionnement normal, le serveur postgreSQL ne supprime pas immédiatement les versions périmées des lignes de tables après un **UPDATE** ou un **DELETE**
    - Afin de garantir la lecture cohérente et la consistance des accès concurrents
    - les versions de la ligne ne doit pas être supprimée tant qu'elle est susceptible d'être lue par une autre transaction
      - Une ligne qui est plus vieille que toutes les transactions en cours n'est plus utile du tout
      - La place qu'elle utilise doit être rendue (par la commande « vacuum ») pour être réutilisée par d'autres lignes afin d'éviter un accroissement constant du volume occupé sur le disque
        - » (voir le Chapitre 13, Contrôle d'accès simultané) de la documentation en ligne http://docs.postgresqlfr.org/8.3/maintenance.html

Maintenance

- Récupérer l'espace disque (suite)
  - Il existe deux variantes de la commande « vacuum »
    - La première connue comme « vacuum »
    - La deuxième est la commande « vacuum FULL »
  - La forme standard de « **vacuum »** est utilisée dans le but de maintenir une utilisation simple de l'espace disque, mais si vous avez besoin de redonner de l'espace disque au système d'exploitation, vous pouvez utiliser « **vacuum FULL»**
    - Remarque : il est préférable d'utiliser des « **vacuum »** standards avec une fréquence modérée plutôt que des « **vacuum FULL»**, même non fréquents, pour maintenir des tables mises à jour fréquemment

Maintenance

- Récupérer l'espace disque (suite)
  - La commande « vacuum »
    - marque les données « périmées » dans les tables et les index pour une utilisation future ;
    - il ne tente *pas* de récupérer l'espace utilisée par cette donnée « périmée » sauf si l'espace est à la fin de la table et qu'un verrou exclusif de table puisse être facilement obtenu
    - L'espace inutilisé au début ou au milieu du fichier ne résulte pas en un raccourcissement du fichier et de l'espace redonné au système d'exploitation
    - Cette variante de « vacuum » peut être lancée en concurrence avec les autres opérations normales de la base de données
      - Exécutée par le process « auto-vacuum »

Maintenance

- Récupérer l'espace disque (suite)
  - La commande « vacuum FULL »
  - utilise un algorithme pour récupérer l'espace consommé par les versions « périmées » des lignes
  - Tout espace qui est libéré par « vacuum FULL » est immédiatement rendu au système d'exploitation et les données de la table sont compressées sur le disque
  - Cette variante de la commande acquiert un verrou exclusif sur chaque table avant que « vacuum FULL » ne la traite
    - Elle bloque le segment et ne peut pas être exécutée en pleine journée
  - Si des tables fréquemment mises à jour n'ont pas eu de « vacuum » fréquent, vous pouvez utiliser « vacuum FULL » ou **CLUSTER** ou **ALTER TABLE** pour obtenir à nouveau de bonnes performances
    - il est bien plus lent de parcourir une table contenant une grosse majorité de lignes périmées
    - Pour une table dont le contenu entier doit être supprimé périodiquement, utilisez la commande **TRUNCATE** plutôt que **DELETE** suivi par un **VACUUM**
      - **TRUNCATE** supprime le contenu entier de la table immédiatement sans nécessiter un **VACUUM** ou **VACUUM FULL**

Maintenance

- Récupérer l'espace disque (suite)
  - Vacuum [FULL] [FREEZE] [VERBOZE] [table] ;
  - Vacuum [FULL] [FREEZE] [VERBOZE] ANALYZE [table [(colonne [, …])…]] ;
    - FULL , effectue un vacuum FULL
      - L'option FULL ne réduit pas la taille des index
      - Un **REINDEX** périodique est toujours recommandé
      - Remarque : il est souvent plus rapide de supprimer tous les index, d'exécuter un **VACUUM FULL** puis de recréer les index
    - FREEZE, est équivalent à réaliser un **VACUUM** avec le paramètre vacuum_freeze_min_age configuré à zéro,
    - VERBOSE, affiche un rapport détaillé de l'activité de vacuum sur chaque table.
      - Peut être utilisé pour aider à déterminer le paramétrage approprié pour max_fsm_pages, max_fsm_relations et default_statistics_target
    - ANALYZE, met à jour les statistiques utilisées par l'optimiseur
    - Table, nom de la table à traiter,
      - par défaut, toutes les tables de la base de données courante sont traitées
    - Colonne, le nom d'une colonne spécifique à analyser
      - Par défaut, toutes les colonnes
  - Exemple : Vacuum verbose analyze client ;

Maintenance

- Récupérer l'espace disque (suite)
  - La commande CLUSTER
    - Réorganiser une table en fonction d'un index, la table est physiquement réorganisée en fonction de l'ordre de l'index
    - Cette commandes écrivent une nouvelle copie de la table et lui adjoint de nouveaux index
      - Ne concerne pas les lignes futures
      - Configurer le paramètre FILLFACTOR à moins de 100% aide à préserver l'ordre du cluster lors des mises à jours dans la page
      - La commande pose un verrou sur la table de type : « access exclusive »
    - sans paramètre, réorganise toutes les tables de la base de données courante qui ont déjà été réorganisées et dont l'utilisateur est propriétaire, ou toutes les tables s'il s'agit d'un superutilisateur
    - Cluster [VERBOSE] nom_table [USING nom_index]
    - Cluster [VERBOSE]
    - Exemple
      - Cluster client ON indx_client ;
    - supportée pour la compatibilité avec les versions de PostgreSQL™ antérieures à la 8.3

Maintenance

- Récupérer l'espace disque (suite)
  - Commande ALTER TABLE et options de stockage
  - Cette commandes écrit une nouvelle copie de la table et lui adjoint de nouveaux index
  - Exemples
    - Alter table article set tablespace data1 ;
      - Déplace la table dans le tablespace DATA1
    - Alter table mon_schema.client set SCHEMA commande ;
      - Déplace la table client apartenant à mon_schéma dans le schéma commande

Maintenance

- Commande vacuumdb
  - récupère l'espace inutilisé et, optionnellement, analyse une base de données PostgreSQL
  - outil de nettoyage d'une base de données
    - vacuumdb peut également engendrer des statistiques internes utilisées par l'optimiseur de requêtes PostgreSQL
  - surcouche de la commande « vacuum »
    - Il n'y a aucune différence entre « vacuumdb » et les autres méthodes de vacuum
    - Vaccumdb [option de connexion] [[--full]
| [-f]] [[--verbose]
| [-v]] [[--analyze]
| [-z]] [--table
|-t table [(colonne [, …])]] [base de données] ;
    - Vaccumdb [option de connexion] [[--all]
| [-a]] [[--full]
| [-f]] [[--verbose]
|[-v]] [[--analyze] | [-z]] ;

Maintenance

- Commande vacuumdb (suite)
  - Options :
    - -a, --All, Nettoie toutes les bases de données
    - [-d] nom_base , [--dbname] nom_base, indique le nom de la base à reorganiser ou à analyser
      si aucun nom n'est spécifié, ni ALL, le nom de la base est récupéré dans la variable d'environnement PGDATABASE
      si la variable n'est pas initialisée, c'est le nom d'utilisateur précisé pour la connexion qui est utilisé
    - -e, --echo, affiche les commandes que vacuumdb engendrent
    - -f, --full, exécute un nettoyage complet
    - -q, --quiet, n'affiche pas les message de progression
    - -t table, ne nettoie ou n'analyse que la table nommée « table »

Exploitation d'une instance

- Processus « auto-vacuum »
  - Le processus d'arrière plan auto-vacuum surveille l'activité des tables et se déclenche si nécessaire pour exécuter des VACUUM
    - L' 'autovacuum fonctionne dynamiquement
    - il est souvent meilleur que des opérations de VACUUM programmées
    - Néanmoins il peut être nécessaire de l'exécuter manuellement de temps en temps

Maintenance

- Processus « autovacuum » (suite)
  - Existe depuis la version 8.1, à partir de la version 8.3 le process a une architecture multi-process
    - Le lanceur « Autovacuum launcher » lance un processus travailleur « autovacuum worker » pour toutes les bases de données
    - Le lanceur distribue le travail
    - Il lancera un nouveau processus travailleur toutes les « autovacuum naptime » secondes
    - Un travailleur sera lancé pour chaque base de données, avec un maximum de « autovacuum_max_workers » processus fonctionnant en même temps
    - S'il y a plus de « autovacuum_max_workers » bases à traiter, la prochaine base de données sera traitée dès qu'un autre travailleur aura terminé
    - Les processus travailleurs vérifieront chaque table de leur base de données et exécutera un **VACUUM** et/ou un **ANALYZE** suivant les besoins
      - Le but est d'automatiser les commandes VACUUM et ANALYSE
    - Ces vérification utilisent la génération de statistiques
      - Implique que la directive « track_count » = TRUE
    - le nombre de travailleurs en cours d'exécution ne comptent pas dans les limites de « max_connections » et « superuser_reserved_connections »

Maintenance

- Processus « autovacuum » (suite)
  - Les tables dont la valeur de « relfrozenxid » est plus importante que « autovacuum_freeze_max_age » font toujours l'objet d'un vacuum
  - deux conditions sont utilisées pour déterminer l'opération qui s'applique
    - Si le nombre de lignes obsolètes depuis le dernier **VACUUM** dépasse une « limite de vacuum », la table bénéficie d'un VACUUM
    - Limite du vacuum = limite de base du vacuum + facteur d'echelle du vacuum * nombre de lignes
      - limite de base du vacuum = autovacuum_vacuum_threshol
      - facteur d'echelle du vacuum = atovacuum_vacuum_scale_factor
      - Le nombre de lignes = pg_class.reltuples

Maintenance

- Processus « autovacuum » (suite)
  - Le nombre de lignes est obtenu à partir du récupérateur de statistiques
    - c'est un nombre à peu près précis, mis à jour après chaque instruction **UPDATE** et **DELETE**
  - Pour ANALYZE, une condition similaire est utilisée : la limite est comparée au nombre de lignes insérées ou modifiée depuis le dernier analyze,
  - cette limite est définie comme
    - Limite analyze = limite de base du analyze + facteur d'échelle du analyze * nombre de lignes
    - Les limites et facteurs d'échelle par défaut sont pris dans postgresql.conf
      - Voir Section 18.9, « Nettoyage (**vacuum**) automatique » pour plus de détails
      - Le contenu du catalogue système pg_autovacuum n'est actuellement pas pris en compte dans les sauvegardes de bases de données créées par les outils **pg_dump** et **pg_dumpall**
        - » Si vous voulez les préserver après un cycle sauvegarde/restauration, assurez-vous que vous avez sauvegardé manuellement le catalogue

Maintenance

- Gestion des statistiques
  - L'optimiseur de requêtes de PostgreSQL s'appuie sur des informations statistiques sur le contenu des tables dans l'optique de produire des plans d'exécutions efficaces pour les requêtes
  - Ces statistiques sont collectées par la commande ANALYZE, qui peut être invoquée seule ou comme option de **VACUUM**
  - Le démon d'autovacuum, si activé, va automatiquement exécuter des commandes **ANALYZE** à chaque fois que le contenu d'une table aura changé suffisamment
  - Il est possible d'exécuter **ANALYZE** sur des tables spécifiques, voire des colonnes spécifiques
    - **ANALYZE** utilise un système d'échantillonnage des lignes d'une table, ce qui lui évite de lire chaque ligne
    - il peut être intéressant d'ajuster le niveau de détail des statistiques collectées pour chaque colonne, en effet les colonnes très utilisées dans les clauses WHERE et dont la distribution n'est pas uniforme requièrent des histogrammes plus précis
      - générés avec la commande ALTER TABLE SET STATISTICS
      - ou modifier les paramètres par défaut de la base de données en utilisant le paramètre de configuration « default_statistics_target » qui représente les paramètres par défaut des statistiques

Maintenance

- Définition de la carte de visibilité
  - Ce fichier, contient toutes les pages visibles par toutes les transactions en cours
    - stockée sur disque, avec un fichier par table
    - Ce fichier a pour nom le numéro de la table (numéro correspondant à la valeur de la colonne relfilenode dans le catalogue système pg_class) suivi du suffixe « _vm »
  - c'est la liste des pages ne contenant aucune donnée modifiée, donc ne nécessitant pas de VACUUM
    - Ce dernier a été amélioré pour tenir compte de cette information et ne parcourir que les pages vraiment modifiées
    - Néanmoins, pour éviter que la table ne soit jamais parcourue entièrement par un VACUUM, un nouveau paramètre est apparu : vacuum_freeze_max_age.
      - Ce paramètre indique l'âge maximum d'une table avant qu'elle ne subisse un VACUUM complet

Maintenance

- Mise à jour de la carte de visibilité
  - La commande VACUUM maintient le contenu de la carte de visibilité pour chaque table, pour conserver la trace de chaque page contenant seulement des lignes connues pour être visibles par toutes les transactions actives
    - ainsi que les futures transactions, jusqu'à la prochaine modification de la page
    - Utilisé par la commande vacuum pour savoir s'il a quelque chose à faire
    - Le module pg_visibility peut être utilisé pour examiner les informations enregistrées dans la carte de visibilité
  - Cette carte a deux buts :
    - Tout d'abord, le VACUUM peut ignorer ce type de pages à sa prochaine exécution car il n'y a rien à nettoyer dans ces pages
    - Ensuite, il permet à PostgreSQL de répondre à certaines requêtes en utilisant seulement l'index, sans faire référence à la table sous-jacente
  - les index de PostgreSQL ne contiennent pas d'informations sur la visibilité des lignes,
    - un parcours d'index normal récupère la ligne de la table pour chaque entrée d'index correspondante, ce qui permet de vérifier si la ligne correspondante est bien visible par la transaction en cours.
    - Un parcours d'index seuls vérifie en premier lieu la carte de visibilité
      - s'il est connu que toutes les lignes de la page sont visibles, la lecture de la table peut être évitée
      - Ceci est très utile sur les gros ensembles de données où la carte de visibilité peut éviter des accès disques
      - La carte de visibilité est très largement plus petite que la table, donc elle peut facilement être mise en cache même quand la table est très grosse

Maintenance

- Postgresql stock les informations relatives au fonctionnement d'une base de données dans espace disque appelé : pg_catalog
- Il contient un ensemble de tables et de vues qui permettent de retrouver des métas données des bases et des données d'exécution
  - Pg_class : informations sur les tables, les index, les séquences et les vues
    - La colonne relname correspond au nom de l'objet
    - La colonne oid correspond à son identifiant
    - La colonne relnamespace correspond à l'identifiant de l'espace de nom de l'objet
    - La colonne reltablespace correspond à l'identifiant du tablespace
    - La colonne relpages correspond au nombre de blocs de l'objet (de 8 KB)
    - La colonne retuples correspond au nombre de lignes de l'objet
      - Les colonnes relpages et retuples sont dépendantes des statistiques
    - La colonne relfilenode correspond au nom du fichier de l'objet
      - Ce nom peut être identique à l'identifiant de la table
    - La colonne relkind correspond au type d'objet :
      - R = table
      - T = table TOAST
      - I = index
      - S = séquence
      - V = vue

Maintenance

- : pg_catalog (suite)
  - Pg_autovacuum : chaque colonne correspond au paramètre de configuration de l'autovacuum (job automatique de statistiques et de maintenance)
    - Chaque ligne correspond à une table
  - Pg_database : informations sur les bases de données
  - Pg_namespace : informations sur les espaces noms
  - Pg_tablespace : liste des tablespaces de l'instance
  - Pg_roles : liste des rôles
  - Pg_stat_activity : liste des connexions courantes

Maintenance

- Vues statiques
  - Activité des processus
    - Pg_stat_activity : contient une ligne par processus système
    - Pg_stat_bgwriter : une seule ligne qui montre des stats du processus d'écritures en tâche de fond et concernant le cluster dans son ensemble
  - Activité des bases de données
    - Pg_stat_all_database : une ligne par base, affiche le nombre de processus actifs connecté à la base, le nombre de transactions commitées et annulées, les lectures sur disque, les « buffer hits » , etc

Maintenance

- Vues statiques (suite)
  - Activité sur les lignes
    - Pg_stat_all_tables : un ligne par table. Affiche le nombre de scans séquentiel et de scans d'index, le nombre d'insertion, de mises à jour et de suppression, la date du dernier vacuum, etc…
    - Pg_stat_all_indexes : une ligne par index, affiche le nombre de scans de l'index, le nombre d'entrées retournées par index scan, etc …
  - Activité sur le disque
    - Pg_statio_all_tables : une ligne par table, montre le nombre de blocs lus, la quantité de « buffer hits » , le nombre de blocs lus, la quantité de « buffer hits » sur les index
    - Pg_statio_all_indexes, une ligne par index, affiche le nombre de blocs lus et de buffers hits pour chaque index
    - Pg_statio_all_sequences, une ligne par séquence, affiche le nombre de blocs lus et de buffers hits pour chaque séquence

Maintenance

- Fonctions d'exploitation
  - Pg_relation_size(oid|text) : retourne la taille d'un objet tel qu'une table ou un index
  - Pg_total_relation_size(oid|texte) : retourne la taillle total d'une table et ses objets dépendants comme les index ou les tables TOAST
  - Pg_database_size(oid|texte) : retourne la taille d'une database (en octet)
  - Pg_tablespace_size(oid|texte) : indique l'espace disque utilisé (en octet)
  - Pg_size_pretty(bigint) accepte en entrée un entier et le converti en chaine de caractère, indiquant une taille lisible

Maintenance

- Requêtes intéressantes
  - Select pg_size_pretty (pg_relation_size(‘clients’)) ;
    pg_size_pretty
    ---
    256 MB

Travaux pratiques

- Dans le cahier de travaux pratiques
  - Faire l'exercice 05_exploitation