420-C42

Langages d'exploitation des bases de données

Partie 11

DDL II

Création, modification et suppression d’objets supplémentaires
base de données, schéma, séquence, vue, index et objets temporaires

Objets supplémentaires

- Tous les SGBD permettent la création de plusieurs types d'objets.
- Nous avons utilisé jusqu'à maintenant des tables, des colonnes, des contraintes et des types énumérés (4 types d'objet au sens de PostgreSQL) mais il en existe plusieurs autres :
  - bases de données
  - schémas
  - types de données
  - opérateurs
  - séquences
  - vues
  - index
  - requêtes préparées
  - procédures et fonctions SQL
  - procédures et fonctions PL/pgSQL
  - déclencheurs
  - usagers
  - groupes
  - plusieurs autres types de données
  - …

Objets supplémentaires

- Certains SGBD comme PostgreSQL dispose hiérarchiquement tous les objets dans une structure à deux niveaux (à l'exception des usagers et des groupes) :
  - database
    - schema
      - tous les autres objets
- Ainsi, on peut dire qu'un objet *database* permet l'organisation de plusieurs schémas. À son tour, un objet *schema* permet l'organisation de plusieurs autres types d'objets.

Objet base de données DATABASE

- Considérant un SGBD comme PostgreSQL, un objet *database* est un conteneur logique permettant le regroupement de schémas.
- Ce regroupement logique se retrouve au sommet hiérarchique du SGBD et permet une gestion simplifiée des tous les objets inclus dans l'infrastructure.
- Lors de la connexion au serveur, un accès spécifique est établi à une seule *database*. Ce lien est exclusif à une seule base de données et immuable pendant tout le temps de la connexion. Dès lors, Il est donc impossible de passer d'une *database* à une autre sans établir une nouvelle connexion. Il est toutefois possible d'établir plusieurs connexions simultanées.
- La base de données nommé ***postgres*** est créée et utilisée par défaut.

Objet base de données DATABASE

- Pour manipuler ces objets, voir :
  - CREATE DATABASE …
  - DROP DATABASE …
  - ALTER DATABASE …
- Il faut avoir des privilèges particuliers pour pouvoir créer et manipuler un objet *database*. L'administrateur système possède ces privilèges.
- Pour des raisons évidentes, une attention particulière doit être portée à la clause DROP DATABASE.

Objet schéma SCHEMA

- Un schéma est un conteneur logique (ou un espace de nom) contenu hiérarchiquement sous un objet *database*.
- Ils servent à regrouper plusieurs objets pour en faciliter la gestion. Ils permettent aussi l'existence d'objets de même nom (similairement à des fichiers dans un dossier).
- Contrairement aux bases de données, les schémas appartiennent à un usager. C'est aussi dire que les objets sous-jacents appartiennent au même usager.

Objet schéma SCHEMA

- Contrairement aux *databases*, les schémas sont flexibles d'utilisation : lors d'une même connexion, un usager peut accéder aux objets de n'importe quel schéma issu de la même *database* s'il en a les privilèges (*nom_du_shema.objet*).
- Le schéma nommé ***public*** est créé automatiquement et utilisé par défaut.
- Les clauses permettant la manipulation des schémas sont :
  - CREATE SCHEMA ...
  - DROP SCHEMA ...
  - ALTER SCHEMA ...

Objet séquence SEQUENCE

- Une séquence est un objet générateur de nombres.
- Il est possible de paramétrer la séquence :
  - incrément — amplitude (peut être positif ou négatif), +1 par défaut
  - valeur minimum — par défaut : 1 pour les inc. positifs ou $-2^{63} - 1$ pour les inc. négatifs
  - valeur maximum — par défaut : $2^{63} - 1$ pour les inc. positifs ou -1 pour les inc. négatifs
  - valeur de départ — par défaut : valeur min. pour les inc. positifs ou max. pour les inc. négatifs
  - valeurs cycliques ou non — définie le comportement lorsqu'on atteint le max. ou le min.
  - nombre de valeur mise en mémoire cache — 1 par défaut

Objet séquence SEQUENCE

- Le synopsis simplifié de la création d'une séquence est :

```
CREATE SEQUENCE [ IF NOT EXISTS ] name
        [ INCREMENT [ BY ] increment ]
        [ MINVALUE minvalue | NO MINVALUE ]
        [ MAXVALUE maxvalue | NO MAXVALUE ]
        [ START [ WITH ] start ]
        [ CACHE cache ]
        [ [ NO ] CYCLE ];
```

- La suppression et la modification d'une séquence utilisent les clauses :
  - DROP SEQUENCE [IF EXISTS] *sequence* [CASCADE | RESTRICT];
  - ALTER SEQUENCE [IF EXISTS] *sequence* …

Objet séquence SEQUENCE

- Il est ensuite possible de manipuler les séquences avec ces fonctions :
  - currval('*nom_texte_sequence'*)
    retourne la dernière valeur générée de la séquence
  - lastval()
    retourne la dernière valeur générée parmi toutes les séquences existantes dans la base de données
  - nextval('*nom_texte_sequence*')
    génère un nouveau nombre et le retourne
  - setval('*nom_texte_sequence*', *valeur*)
    détermine la valeur de la séquence

  - Une attention particulière doit être portée au fait que le nom de la séquence est donné par une chaîne de caractères.

Objet séquence SEQUENCE

- Les types de données SMALLSERIAL, SERIAL et BIGSERIAL utilisent une séquence pour la génération des identifiants.
- D'ailleurs, il est important de savoir que le nom donné à une séquence issu d'un type SERIAL est automatique et standardisé selon ce patron : *table_colonne*_seq.
- En fait, voici à quoi correspond exactement le type SERIAL :

```
CREATE TABLE employe (
   id SERIAL
);
-- correspond à
CREATE SEQUENCE employe_id_seq START WITH 1 INCREMENT BY 1;
CREATE TABLE employe (
   id INTEGER NOT NULL DEFAULT nextval('employe_id_seq')
);
-- défini que la séquence est une dépendance de la colonne id de
-- la table employe - pour ne pas oublié les suppressions en cascade
ALTER SEQUENCE employe_id_seq OWNED BY employe.id;
```

Objet séquence SEQUENCE

- Utilisation des séquences :

```
-- création de 2 séquences
CREATE SEQUENCE seq_emp_id;
CREATE SEQUENCE seq_dep_id
        INCREMENT BY 2 START WITH 1000 NO CYCLE;
-- utilisation des séquences
INSERT INTO departement (id, nom)
        VALUES (nextval('seq_dep_id'), 'Administration’);

INSERT INTO employe (id, nom, prenom, departement)
        VALUES (nextval('seq_emp_id'), 'Bo', 'Bill', currval('seq_dep_id'));
```

Objet vue VIEW

- Une vue représente le résultat d'une requête telle une table virtuelle.
- Les vues ne stock aucune données en soit. Elles ne font que représenter la *pseudo-table* issue d'une requête.
- Ainsi, une vue peut être utilisée dans n'importe quel type de requête comme toute autre table.
- Le contenu des vues est maintenu en tout temps lorsqu'une table de base est modifiées par les opérations du DML.
- Sous certaines conditions, il est aussi possible d'utiliser le DML directement sur une vue (vues actualisables).

Objet vue

VIEW

- Les vues présentent ces avantages :
  - grande simplification de requêtes complexes
  - puisque les vues sont maintenues à jour, elles peuvent grandement réduire la charge de calcul de requêtes récurrentes
  - peut faciliter la gestion des accès aux données (accès aux vues et non aux tables de base)
- Les instructions DDL liées aux vues sont :
  - CREATE [OR REPLACE] VIEW *nom_vue* AS *requête*;
  - DROP VIEW [IF EXISTS] *nom_vue* [CASCADE | RESTRICT];
  - ALTER VIEW [IF EXISTS] *nom_vue* ...

Objet vue VIEW

- Un exemple :

```
-- création d'une vue retournant le nombre d'employé par département
CREATE VIEW nbr_emp_dep AS
        SELECT departement AS id, COUNT(*) AS nbr
                FROM employe
                GROUP BY departement;


-- donne le nom des départements et du nombre d'employé y travaillant
SELECT departement.nom, nbr_emp_dep.nbr
        FROM departement
                INNER JOIN nbr_emp_dep
                        ON nbr_emp_dep.id = departement.id;
```

Objet index

INDEX

- Un index est une structure de données complémentaire aux tables permettant une augmentation substantielle de la performance des SGBD lors d'accès aux données.
- Un parallèle facile : l'index d'un livre (consulter l'index pour rechercher telle information; on la retrouve aux pages 2, 7 et 72).
- L'utilisation des index est d'une très grande simplicité :
  - ils sont définis par le concepteur à l'aide du DDL
  - automatiquement maintenue à jour pour chaque opération du DML
  - automatiquement utilisé dans toutes les requêtes du DQL
- Il importe d'être prudent car un mauvais usage peut tout de même résulter en une perte de performance.

Objet index

INDEX

- Un index est automatiquement créé lors de la création d'une contrainte d'unicité. Implicitement un index est donc créé pour les contraintes de clé primaire.
- Les index peuvent :
  - être à valeur unique (interdit les doublons);
  - défini selon :
    - une colonne
    - plusieurs colonnes : nom, prenom
    - une expression : UPPER(nom)
  - déterminer l'algorithme d'indexation : btree, hash, gist et gin
  - selon un tri ascendant ou descendant;
  - positionner les valeurs nulles au début ou à la fin.

Objet index

INDEX

- Synopsis de base :

CREATE [UNIQUE] INDEX [*nom_index*]
ON *table*
[USING { BTREE | HASH | GIST | GIN }]
({ *colonne* | (*expression*) }
[ASC | DESC]
[NULLS { FIRST | LAST }]
[, ...]);

- Les autres instructions du DDL sont :
  - DROP INDEX [IF EXISTS] nom_index [CASCADE | RESTRICT];
  - ALTER INDEX [IF EXISTS] *nom_index* ...

Objet index

INDEX

- Un exemple :

```
-- la requête suivante peut être lente selon le nombre d'employés
SELECT nom, prenom, salaire, genre
        FROM employe
        WHERE UPPER(ville) IN ( 'MONTRÉAL', 'LONGUEUIL');

CREATE INDEX idx_emp_ville
        ON employe( (UPPER(ville)) ASC NULLS LAST);

-- suite à la création de cet index,
-- la requête précédente sera beaucoup plus rapide
```

Objets temporaires

- PostgreSQL possède un mécanisme puissant pour faciliter la création de certains objets temporaires.
- Leur usage est important et voici quelques exemples concrets :
  - Import/Export de données: L'un des usages importants (dans le cours du moins).
  - Isolation des modifications (transactions complexes):
  - Les usages suivants sont aussi des bons exemples mais moins pertinents avec les ETC que nous verrons plus loin :
    - Travaux intermédiaires complexes (calculs et transformations)
    - Tests et prototypage rapide.
    - Développement et débogage de requêtes.

Objets temporaires

- Portée exclusive dans la session courante :
  - Les objets temporaires sont uniquement visibles et accessibles au sein de la session de base de données qui les a créés.
  - Chaque session a son propre espace temporaire isolé, ce qui signifie que des sessions concurrentes ne peuvent pas interférer ou voir les objets temporaires des autres.
- Gestion automatique du schéma temporaire :
  - En coulisses, PostgreSQL utilise un schéma temporaire propre à chaque session pour stocker les objets temporaires (schéma souvent nommé pg_temp).
  - Ce schéma est créé et géré automatiquement par PostgreSQL. Bien qu'on ne le manipule pas directement, c'est ce mécanisme qui permet l'isolation et le nettoyage automatique des objets temporaires.
  - Lorsque la session qui a créé l'objet temporaire se termine (normalement ou anormalement), PostgreSQL supprime automatiquement cet objet (et toutes ses dépendances).

Objets temporaires

- Les objets temporaires sont :
  - Table temporaire
    - Un index temporaire sur une table temporaire.
    - Un déclencheur temporaire (*trigger*) sur une table temporaire (nous verrons plus loin).
    - Il est important de savoir que tous les objets internes à une table temporaire le sont aussi : colonne, contrainte, séquence créée automatiquement par SERIAL, …
  - Vue temporaire
  - Séquence temporaire
- L'instruction du DML est :
  CREATE [TEMPORARY | TEMP] *objet* ...
- Il est possible de modifier ou supprimer les objets avec ALTER et DROP.
- Nous y reviendrons dans le contexte d'importation/exportation dans le DML II.