TP PostgreSql n°5

– PL/PG SQL – curseurs – triggers –

Il est conseillé de travailler le plus possible avec des fichiers sql plutôt qu'avec l'éditeur de PostgreSQL .

PostgreSQL et psql

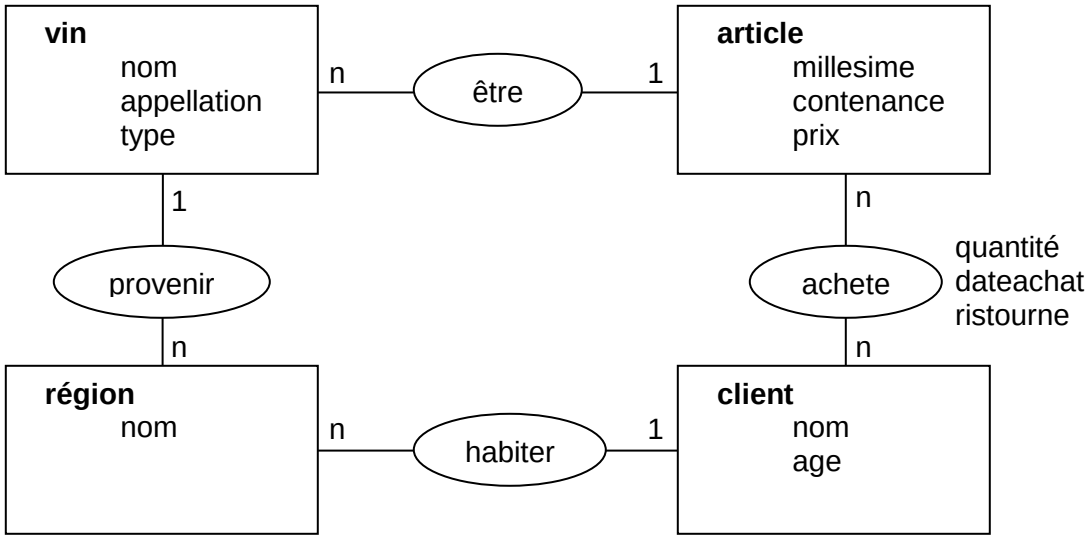
Connectez-vous à la base de données (tp_intro ou autre bdd déjà existante).

Commandes de base. Les principales commandes de psql sont les suivantes :
– \? : Liste des commandes psql,
– \h : liste des instructions SQL,
– \h <une_instruction> : description de l'instruction SQL <une_instruction>.
Essayez avec l'instruction CREATE TABLE,

– \d : liste des tables (ou relations, cf. partie 2),
– \d <nom_table> : description de la table <nom_table>,
– \i <chemin/nom_fichier_script.sql> : exécution d'un fichier de script SQL,
– \o <chemin/nom_fichier_resultat.sql> : écriture des résultats des instructions dans le fichier passé en paramètre,
– \o : retour à un affichage à l'écran,.
– \q : quitter psql.

Retenez ces commandes, elles seront indispensables pour les TP.

Nous utiliserons la même base de travail que pour le TP précédent :



Insérez la ligne suivante dans la table `vin` :

```
INSERT INTO vin VALUES (12,'Côtes de Provence','AOC','rouge',2);
```

Ou ré-exécutez le script de génération du schéma `/users/prof/victorfernandes/data/vin.sql`.

1) Exemple Oracle PL/SQL

Définition :

Le langage PL/SQL est un langage de quatrième génération, fournissant une interface procédurale à PostgreSql .

Le langage PL/SQL permet de manipuler de façon complexe les données contenues dans une base PostgreSql en transmettant un bloc de programmation au système de gestion de bases de données au lieu d'envoyer une requête SQL. De cette façon, les traitements sont directement réalisés par le SGBD. Cela a pour effet notamment de réduire le nombre d'échanges à travers le réseau et donc d'optimiser les performances des applications.

Le langage PL/SQL permet de définir un ensemble de commandes contenues dans ce que l'on appelle un "bloc" PL/SQL.

Exemple de fonction PL/SQL :

```
CREATE OR REPLACE FUNCTION num_auto_vin RETURN NUMBER
IS
 numax NUMBER;
BEGIN
      SELECT MAX (idvin) INTO numax FROM Vin;
      RETURN numax+1;
END num_auto_vin;
/
```

Nous déclarons ici une fonction `num_auto_vin` qui retourne la plus grande valeur de `idvin` dans la table `Vin` et lui rajoute 1. (En gros cela permet de créer un équivalent à une séquence.)

Voici le détail de la fonction :

1. Une fonction PL/SQL est un programme stocké dans la base de données (grâce à `CREATE`) qui retourne un résultat.

2. `num_auto_vin` est le nom de la fonction, il pourrait être suivi par une liste d'arguments formels entre parenthèses.

3. La fonction retourne un résultat dont le type dans notre cas est un nombre (`RETURN NUMBER`).

4. Les instructions comprises avant le corps de la fonction sont des définitions de variables locales à la fonction (dans notre cas `numax` de type `NUMBER`).

5. Le corps de la fonction est compris dans le bloc `BEGIN ... END`. Préciser `num_auto_vin` après `END` est facultatif.

6. L'instruction `SELECT ... INTO` extrait des données de la table `Vin` et affecte le résultat dans la variable `numax`.

7. L'instruction `RETURN` précise la valeur à retourner, ici c'est le contenu de la variable `numax + 1`.

NB : Pour terminer la définition d'une fonction, il faut mette un **/** sur une ligne vide.

Exercice

Créer votre fonction PL/PGSQL sur la base du code PL/SQL (Oracle ci-dessus)

Testez votre fonction

Vérification de l'existence de la fonction

Sous Oracle : Pour vérifier que votre fonction est bien présente dans le système, utilisez la commande :

```
SELECT * FROM ALL_OBJECTS WHERE OWNER='LOGIN' AND OBJECT_TYPE='FUNCTION';
```

en remplaçant LOGIN par votre identifiant de connexion en majuscule.

Dans mon cas : `SELECT * FROM ALL_OBJECTS WHERE OWNER='victorfernandes' AND OBJECT_TYPE='FUNCTION';`

PostgreSql définit des exceptions permettant de gérer certains cas particuliers :

- `NO_DATA_FOUND` : pas de données.
- `TOO_MANY_ROWS` : ligne de retour non unique.
- ...

2) Création de fonctions

Exercices

1. Créez une fonction PL/PGSQL qui prend en paramètre un nom de vin et renvoie l'identifiant de ce vin s'il existe.
   a. Exécutez votre fonction pour connaître le numéro du vin dont le nom est 'Pétrus'.
   b. Exécutez votre fonction pour connaître le numéro du vin dont le nom est 'Nuit Saint Georges'.
   c. Exécutez votre fonction pour connaître le numéro du vin dont le nom est 'Côtes de Provence'.
   d. Que constatez-vous pour cette dernière question ?
2. Afin de prendre en charge cette dernière erreur, supprimez cette fonction et recréez-la en gérant les exceptions prédéfinies `NO_DATA_FOUND` et `TOO_MANY_ROWS`. Ajoutez également le cas où il n'y a pas de réponse et dans ce cas retournez la valeur 0. Quel est le résultat des questions a, b et c ?

3) Les curseurs Oracle

Les curseurs sont des pointeurs sur une zone mémoire pour les données extraites de la base. Il existe des curseurs implicites et explicites. PostgreSql ouvre toujours un curseur implicite pour traiter une instruction SQL, celui-ci ne se rapporte qu'à la dernière instruction SQL exécutée et il se nomme « SQL ». Le curseur contient des attributs (`%NOTFOUND`, `%FOUND`, `%ROWCOUNT`) qui fournissent des informations sur l'exécution des instructions `INSERT`, `UPDATE`, `DELETE`, `SELECT INTO`. Un curseur implicite pour une instruction `SELECT INTO` ne peut gérer qu'une seule ligne. Le curseur explicite quant à lui place le résultat d'une requête multi-lignes dans un tampon mémoire et libère les lignes une après l'autre lors du traitement.

Le curseur se définit dans la partie déclarative du bloc PL/SQL. Dans cette déclaration, il est possible de donner une clause `FOR UPDATE OF nom_colonne(s)` qui permet de verrouiller les lignes sélectionnées (mise en place d'un verrou pour qu'aucun utilisateur ne puisse mettre la ligne à jour tant que le verrou est en place). La commande `OPEN nom_curseur;` exécute la requête et place le curseur en mémoire, elle ne retourne aucun résultat. L'instruction `FETCH nom_curseur INTO variable;` extrait la ligne courante du curseur, la place dans une variable et fait avancer le curseur à la ligne suivante. Pour parcourir toutes les lignes du curseur, il faut utiliser une boucle `LOOP`. La clause `CURRENT OF nom_curseur` est utilisée dans la clause `WHERE` d'une commande `UPDATE` pour modifier la ligne courante (si une clause `FOR UPDATE` a été utilisée préalablement). L'arrêt de la boucle est obtenu grâce à `nom_curseur%NOTFOUND` qui retourne `false` s'il ne reste plus de lignes. Pour libérer l'espace mémoire, il faut fermer explicitement le curseur en utilisant `CLOSE nom_curseur;`.

Un curseur peut accepter des paramètres en entrée, ils servent à passer des informations au curseur et sont généralement utilisés dans une clause `WHERE` pour limiter la requête. Les paramètres ont un type associé qui ne peut pas avoir d'indication de longueur. Ils sont passés lors de la commande `OPEN`.

Exercices

1. Créez une procédure SQL (une procédure est une fonction qui ne retourne rien) qui augmente de *n* % (*n* en paramètre, compris entre 0 et 100 inclus) le prix de tous les articles antérieurs à une date *d* donnée en paramètre. Définissez une exception dans le cas où le paramètre *n* n'est pas compris entre 0 et 100.
2. Ajoutez la colonne `stock` dans la table `Article`. Créez une procédure qui met à jour tous les articles pour renseigner cette colonne. Les articles antérieurs à 1991 sont uniques en stock. Les articles entre 1991 et 1992 sont au nombre de 4. Les articles entre 1992 et 1995 sont au nombre de 10. Tous les articles postérieurs sont au nombre de 25.

4) Triggers

Les triggers permettent de gérer des évènements de la base de données. Ils sont définis dans la base de données pour une table spécifique et se déclenchent lorsqu'il y a modification de données pour cette table (`INSERT`, `UPDATE`, `DELETE`). Les triggers contiennent du code PL/SQL, ils peuvent appeler toute procédure ou fonction stockée et peuvent référencer d'autres objets de la base de données. Un trigger peut être défini au niveau instruction. Le trigger peut être activé avant que la ligne ne soit affectée (`BEFORE`) ou après (`AFTER`). Un trigger est créé et stocké dans la base de données avec la commande suivante :

```
CREATE [OR REPLACE] TRIGGER nom_trigger
 {BEFORE | AFTER} {INSERT | UPDATE | DELETE}
 [OF nom_col1, ..., nom_colN]
 ON nom_table [FOR EACH ROW];
```

La liste des colonnes ne sert que pour un trigger `UPDATE`. La clause `FOR EACH ROW` indique un trigger de niveau ligne. Le code du trigger est mémorisé dans la table `USER_TRIGGER`.

Pour référencer les valeurs de colonnes dans des trigger de niveau ligne, il faut utiliser les identificateur `:NEW` et `:OLD`. Le premier n'est disponible que pour `INSERT` ou `UPDATE`, le second n'est disponible que pour `UPDATE` ou `DELETE`.

Exercices

1. Créez un trigger PL/PGSQL qui empêche d'effectuer une requête la nuit (entre 20H00 et 8H00) sur la table `Vin`.
2. Créez un trigger PL/PGSQL qui ajoute automatiquement une nouvelle valeur de clé primaire lorsqu'un nouveau vin est entré, permettant ainsi d'insérer des lignes dans la table `Vin` sans devoir préciser un identifiant.