TP1 - Vues, Triggers et Privilèges

Préambule

Tout le TP est basé sur une base de données concernant les bières. Pour rappel, les tables concernées sont au nombre de 6 et sont nommées `buveur`, `bar`, `biere`, `aimer`, `frequenter` et `servir`.
Vous pouvez récupérer la structure et le contenu de ces tables à partir du script joint à cet énoncé. Ce script crée le schéma `bieres` dans lequel seront stockées vos tables.

Exercices

Exercice 1 Vues et confidentialité des données
Lors de l'ouverture d'une session PostgreSQL par un utilisateur autorisé, le nom de cet utilisateur est accessible pendant la session au moyen de l'attribut système `USER`. Ainsi la commande `SELECT USER ;` affichera le nom de l'utilisateur exécutant cette commande.
Les accès en lecture sur les tables de votre base étant interdits à tous les autres utilisateurs (vos camarades de promotion et votre binôme en particulier), proposez une solution à l'aide des vues permettant à chacun des buveurs de voir les informations le concernant :

- ses informations personnelles (dans la table `buveur`) → vue `mes_infos`
- les bières qu'il apprécie → vue `mes_bieres`

Pour tester vos vues, vous devrez au préalable modifier la table `buveur` afin de lui ajouter un attribut `login` qui aura pour valeur un identifiant PostgreSQL. Peuplez la base en ajoutant une valeur à login pour tous les buveurs. Vous ferez en sorte que cet attribut login soit aussi clé candidate de la table buveur. Après que vous aurez donné les bons droits sur votre vue, les buveurs concernés pourront consulter la vue, en particulier votre binôme.

Exercice 2 Indépendance logique grâce aux vues
Supposons que le schéma relationnel de la base des bières est modifié. On suppose maintenant que les buveurs ne fréquentent que les bars qui servent des bières qu'ils aiment. Cela implique donc que la table `frequenter` n'a plus de raison d'être dans le schéma de base. Toutefois, le concepteur de l'interface permettant de gérer la base ne veut pas avoir à modifier ses requêtes qui portaient sur la table fréquenter.
Proposez une vue permettant de "remplacer" la table `frequenter`, c'est à dire faire en sorte qu'on ne voie pas sa disparition.

Exercice 3 Petit retour sur les triggers
On souhaite contrôler qu'on ne puisse modifier l'attribut `degrealcool` de la table `biere` que de 1 degré (en plus ou en moins) maximum. Ecrire le trigger vérifiant cette règle de gestion.

Dans un deuxième temps, faites en sorte que seuls les amateurs des bières visées puissent faire ces modifications.

Exercice 4 Mise à jour au travers d'une vue visant plusieurs tables – Trigger INSTEAD OF
L'option `INSTEAD OF` permet d'associer un trigger à une vue et de permettre de faire des mises à jour des tables sous-jacentes à la vue à travers celle-ci.
On dispose des tables (à créer) :

```
CREATE TABLE buveurs_amateurs (
 idbuveur numeric(3) primary key,
 login varchar(20) not null,
 nomb varchar(30) not null,
 prenomb varchar(30) not null);
```

```
CREATE TABLE buveurs_pros (
  idbuveur numeric(3) primary key,
  login varchar(20) not null,
  nomb varchar(30) not null,
  prenomb varchar(30) not null,
  idbar_prefere numeric(3) not null);
```

et de la vue :

```
CREATE VIEW toutbuveur (idbuveur, login, nomb, prenomb, idbar_prefere, nature)
AS SELECT idbuveur, login, nomb, prenomb, idbar_prefere, 'P'
   FROM buveurs_pros
   UNION
   SELECT idbuveur, login, nomb, prenomb, null, 'A'
   FROM buveurs_amateurs;
```

La sous-requête de définition de la vue comporte un opérateur ensembliste ; on ne peut donc pas insérer directement de nouveaux n-uplets dans les tables sous-jacentes en utilisant la vue. C'est cependant possible en utilisant un trigger `INSTEAD OF` dont voici le début de déclaration :
On pourra alors faire des insertions de la manière suivante :

```
INSERT INTO toutbuveur VALUES(100, 'xxx', 'Ture', 'Tobby', 1, 'P');
INSERT INTO toutbuveur VALUES(101, 'yyy', 'Dupond', 'Dédé', null, 'A');
```

Programmer le trigger qui permet de faire ces types d'insertion ; un code nature erroné (différent de 'P' ou 'A') sera signalé par une erreur d'application.

Exercice 5 Triggers de log
On désire suivre les modifications effectuées sur la table `biere` selon les spécifications suivantes.

- Les traces des modifications seront enregistrées dans la table `audit_biere` (table à créer).
- En cas de suppression d'un n-uplet de biere, on enregistrera `<idbiere, 'DES', utilisateur, date>`.
- En cas d'insertion de nuplet, on enregistrera `<idbiere, 'INS', utilisateur, date>` ; si la valeur idbiere est absente, elle sera remplacée par 'ABS'.
- En cas de mise à jour de degrealcool, on enregistrera `<idbiere, x , utilisateur, date>` où `x` prend la valeur 'AUG' si degrealcool est augmenté et la valeur 'DIM' si degrealcool est diminué. On supposera pour les cas de suppression et de modification que les n-uplets cibles existent et que pour le cas d'insertion il n'y a pas de problème de clé.

Travail à effectuer :

- Ecrire le trigger correspondant.
- Proposer un jeu de tests mettant en évidence le bon fonctionnement du trigger.