info710 : compléments de bases de données

TP 4 : création de tables, contraintes d'intégrité, déclencheurs

Pierre Hyvernat
Laboratoire de mathématiques de l'université de Savoie
bâtiment Chablais, bureau 22
téléphone : 04 79 75 94 22
email : Pierre.Hyvernat@univ-savoie.fr
www : http://www.lama.univ-savoie.fr/~hyvernat/

Vous devrez m'envoyer un petit compte-rendu de ce TP par email avant la fin de la journée. Le sujet de l'email devra comporter "TP4-info710" et le compte-rendu consistera en un seul fichier "`TP4-nom1-nom2.sql`*" dont le squelette se trouve sur*

`http://www.lama.univ-savoie.fr/~hyvernat/`

(dans la section "Enseignement"...)
Ce fichier devra être un fichier SQL valide : toutes les explications demandées devront donc être en commentaire. N'oublier pas de tester votre script avant de me l'envoyer.
Autant que possible, essayez de garder vos lignes à moins de 78 caractères, et indentez vos commandes SQL pour augmenter la lisibilité.

Remarque : le livre "Practical PostgreSQL" se trouve en ligne sur
`http://www.faqs.org/docs/ppbook/book1.htm`
Gardez un navigateur ouvert sur cette page...

Exercice 1 : création de table, contraintes

On commence par finir le TP3...

Loggez-vous sur la base de données `tp3-login`. La commande SQL pour créer une nouvelle table dans une base données est

```
CREATE TABLE nom_table (att1 typ1, ... , attn typn)
```

où `att1`, ..., `attn` sont des noms d'attributs et `typ1`, ..., `typn` sont leurs types. Les types les plus fréquents sont

- `boolean`,
- `int`, et `bigint`,
- `varchar (n)` où `n` est la longueur maximal des chaînes,
- `text`,
- `date`,
- ...

Vous allez recréer une base de données du monde simplifiée. Elle ne contiendra que deux tables `pays` et `villes`. Les attributs seront

- `nom` pour le noms du pays
- `code` pour le code du pays
- `capitale` pour le code de la capitale
- `population` pour la population totale du pays
- `id` pour le code de la ville
- `pays` pour le code du pays auquel appartient la ville
- `nom` pour le nom de la ville
- `population` pour la population de la ville

Pendant tout cet exercice, n'oublier pas de tester vos tables en ajoutant des lignes, en en supprimant et en faisant des requêtes. Vous pouvez utiliser des villes et pays imaginaires (en suivant une certaine cohérence logique quand-même)

Question 1 : en allant lire l'aide la commande `CREATE TABLE`, créez ces deux tables, en précisant les clés principales que vous aurez choisies. N'oubliez pas d'imposer certaines restrictions quand à l'utilisation de la valeur `NULL`.

Question 2 : allez voir les contraintes utilisées pour les tables correspondantes dans la base `tp3-world` et comparez les avec vos contraintes. Essayer de comprendre ce que font les autres contraintes de `tp3-world`.

Question 3 : Rajouter quelques contraintes (taille des villes supérieure à 10, ...)
Pour rajouter une contrainte à une table existante, il faut utiliser la commande

```
ALTER TABLE nom_table [DROP,ADD,RENAME,MODIFY] CONSTRAINT ... ;
```

Question 4 : rajouter les contraintes suivantes :
- chaque pays doit avoir sa capitale dans la liste des villes,
- chaque ville doit avoir un code de pays valide.

Qu'en pensez-vous ?

Question 5 : certaines contraintes peuvent être déclarées comme "différables". Il faut rajouter dans leur définition :

```
CONSTRAINT nom ... DEFERRABLE
```

Une telle contrainte peut être différée en utilisant

```
BEGIN ;
SET CONSTRAINTS nom DEFERRED ;
...
SET CONSTRAINTS nom IMMEDIATE ;
COMMIT ;
```

Essayez cette technologie sur les tables des villes et des pays. Que ce passe-t'il quand vous faites le `COMMIT` ? (La contrainte peut être ou non satisfaite par les tables...)

Question 6 : on peut supprimer une colonne dans une table avec

```
ALTER TABLE nom_table DROP COLUMN nom_col ;
```

Supprimez la colonne `population` de vos tables.

Question 7 : on peut supprimer une table avec

```
DROP TABLE nom_table ;
```

Si vous avez bien sauvegardé votre session SQL, supprimez les deux tables `ville` et `pays`.

Exercice 2 : déclencheurs

Les contraintes d'intégrité permettent de vérifier , lors de l'insertion d'une ligne, que la table reste cohérente. On veut parfois faire des choses un peu plus complexes comme :
- si on rajoute une langue officielle, l'ancienne langue officielle devient non-officielle ;
- si on modifie la population d'une ville, on met à jours le nombre de ville de plus de 500000 habitants (si nécessaire) ;
- ...

La syntaxe pour déclarer un tels déclencheur est assez compliquée. Il faut commencer par déclarer une fonction dont le "type de retour" est `TRIGGER` puis appeler cette fonction lors d'évènements. Par exemple, avec la table

```
CREATE TABLE pays
   (nom varchar(33) NOT NULL,
    code char(3)    PRIMARY KEY,
    capitale int    NOT NULL,
    nb_villes int)
```

on peut déclarer la fonction `incr_nb_ville` avec

```
CREATE OR REPLACE FUNCTION incr_nb_ville ()  RETURNS TRIGGER AS
  'BEGIN
    UPDATE pays SET nb_villes=nb_villes+1 WHERE NEW.pays = code ;
```

```
        RETURN NULL ;
      END ;'
  LANGUAGE plpgsql ;
```

et déclarer le déclencheur avec

```
  CREATE TRIGGER trig_ville
     AFTER INSERT ON ville
     FOR EACH ROW EXECUTE PROCEDURE incr_nb_ville() ;
```

Pour plus de détails, reportez-vous aux chapitres correspondants dans le livre "Practical PostgreSQL"...

Question 1 : en recréant une table `pays` comme dans l'exemple si dessus, et une table `ville` correspondante, testez le déclencheur.

Question 2 : par analogie, créer un autre déclencheur pour décrémenter le nombre de villes lors de la suppression d'une ville. Vérifiez que ça marche...

Question 3 : créez le déclencheur qui fait la chose suivante : si on supprime la capitale, alors on passe la ville ayant le plus grand identificateur comme nouvelle capitale. (!?!)

Question 4 : modifier le déclencheur de la question 3 pour qu'il fonctionne également lorsqu'on modifie l'identificateur d'une ville.

Question 5 : donnez quelques exemples (sans code SQL) d'endroits où l'utilisation des déclencheurs pourrait servir.