Prise en main de PostgreSQL
Introduction aux différents concepts

03 octobre 2012

Préambule

!M: Passer à 1 séance dédoublée **!**

Objectifs. Le Système de Gestion de Base de Données (SGBD) sur lequel nous allons travailler est PostgreSQL. L'objectif de ce premier TP est de vous familiariser à ce SGBD et d'introduire les différents concepts que vous serez amenés à manipuler dans un SGBD.

Interpréteur psql et éditeur de texte. Nous utiliserons l'interpréteur psql de PostgreSQL. Vous trouverez une documentation de PostgreSQL à l'adresse suivante : http://www.postgresql.org/docs/9.2/static/index.html.

Dans ce TP, les instructions à exécuter seront reportées comme étant saisies directement sous l'interpréteur psql. Il est cependant vivement recommandé, avant d'exécuter une instruction dans l'interpréteur psql, de la saisir dans un fichier dont l'extension sera .sql. Cela vous permet :

1. De conserver une trace de ce que vous faites, ce qui vous sera utile lorsqu'il faudra réutiliser des parties de scripts dans les TP suivants ainsi que pour vos révisions.
2. De contrôler la cohérence de vos instructions SQL par le biais de la coloration syntaxique, sous tout éditeur de texte digne de ce nom (emacs par exemple) bien configuré.

Il est d'ailleurs possible d'exécuter un fichier de script SQL depuis l'interpréteur psql et également de récupérer le résultat des instructions exécutées dans un fichier (cf. partie 1).

1 PostgreSQL et psql

Connectez-vous à la base de données tp_intro que vous venez de créer.

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.

Instruction `SQL`. Une instruction `SQL` peut s'écrire sur une ou plusieurs lignes. Le retour chariot n'a pas d'incidence : c'est le `;` qui termine une instruction et permet son exécution. Ainsi, les deux instructions suivantes sont équivalentes.

```
tp_intro=# SELECT * FROM <nom_base>;
<resultat>
tp_intro=#
```

```
tp_intro=# SELECT *
tp_intro-# FROM <nom_base>;
<resultat>
tp_intro=#
```

Notez la différence dans le prompt entre les caractères `=#` et `-#` selon que l'on a effectué, ou pas, un retour chariot et validé, ou pas, l'instruction. On peut également avoir le signe `(#` lorsqu'une parenthèse n'a pas été fermée.

Des commentaires peuvent être insérés dans une instruction `SQL` :
– soit par les caractères `--`, dans ce cas la partie commentée s'étend jusqu'à la fin de la ligne courante,
– soit par les délimiteurs `/* <mon_commentaire> */`, comme en `C`.

La casse n'a aucune influence sur les instructions `SQL` : `SELECT * FROM <nom_base>;` est équivalent à `select * from <nom_base>;`. Néanmoins, il est plus lisible de mettre en lettres majuscules les `INSTRUCTIONS` et en lettres minuscules les noms des `tables`, des `attributs`, etc.

2 Notion de table

Une base de données *relationnelle* est principalement constituée de *tables* (aussi appelées *relations*, qualificatif résultant du terme relationnel). Comme son nom le laisse supposer, une table est une structure de type tableau dans laquelle l'information va être organisée en colonnes (pour les *attributs*) et en lignes (pour les *instances*, encore appelées *tuples*, ou *enregistrements*).

Créer une table. Nous allons dans un premier temps créer le *schéma* d'une table, c'est-à-dire définir et spécifier le type de ses attributs, par le biais de l'instruction `SQL LDD`[1] `CREATE TABLE`. Exécutez l'instruction suivante.

```
tp_intro=# CREATE TABLE t_etudiant(
tp_intro(# pk_num_secu CHAR(13) PRIMARY KEY,
tp_intro(# ck_num_etu VARCHAR(20) UNIQUE NOT NULL,
tp_intro(# nom VARCHAR(50),
tp_intro(# prenom VARCHAR(50));
```

Quelle est la différence entre le type `CHAR` et le type `VARCHAR` ?

Alimenter une table. Maintenant que la table `t_etudiant` est créée, on peut y insérer des tuples avec une instruction `SQL LMD`[2] `INSERT`.

```
tp_intro=# INSERT INTO t_etudiant (pk_num_secu, ck_num_etu, nom, prenom)
tp_intro-# VALUES ('1840791123456', 'DUP1840703', 'Dupont', 'Pierre');
tp_intro=# INSERT INTO t_etudiant (pk_num_secu, ck_num_etu, nom, prenom)
tp_intro-# VALUES ('2850460789101', 'DUC2850428', 'Durand', 'Catherine');
```

Notez qu'en `SQL`, les chaînes de caractères alphanumériques et les dates doivent être encadrées par des quotes (`'`).

1. Langage de Définition des Données.
2. Langage de Manipulation des Données.

Interroger une table. L'instruction SQL LMD SELECT permet d'interroger une (ou plusieurs) tables. Exécutez les instructions suivantes et décrivez le mode de fonctionnement de l'instruction SELECT.

```
tp_intro=# SELECT pk_num_secu, ck_num_etu, nom, prenom FROM t_etudiant;
tp_intro=# SELECT * FROM t_etudiant;
tp_intro=# SELECT nom, prenom FROM t_etudiant;
```

3 Notion de contrainte d'intégrité

Lors de la création d'une table, on définit en même temps des *contraintes* sur les attributs qui serviront à contrôler leur *intégrité* par rapport à des règles préalablement fixées.

Contrainte de domaine. Les *contraintes de domaine* permettent de vérifier qu'un attribut prend ses valeurs parmi un ensemble déterminé : par exemple les chaînes de 13 caractères au plus, les entiers de 1 à 1000, la possibilité ou l'interdiction d'avoir comme valeur l'ensemble vide (NULL), etc.

Exécutez l'instruction suivante et expliquez pourquoi le système renvoie une erreur.

```
tp_intro=# INSERT INTO t_etudiant (pk_num_secu, ck_num_etu, nom, prenom)
tp_intro-# VALUES ('1861175951357', 'DUP2861128DESLETTRESENPLUS', 'Dupond', 'Paul');
<message_d_erreur>
```

Donnez un exemple de contrainte qui n'est pas spécifiée dans la table t_etudiant et qui vous paraît pertinent, et proposez l'instruction qui aurait permis de prendre en compte cela. Supprimez d'abord la table t_etudiant avant d'entrer la nouvelle instruction CREATE TABLE qui vous permettra de modifier la relation [3], puis insérez de nouveaux les tuples.

```
tp_intro=# DROP TABLE t_etudiant;                              -- Supprime la relation t_etudiant
tp_intro=# CREATE TABLE t_etudiant(                                                -- C'est à vous
tp_intro(# ...);
tp_intro=# INSERT INTO t_etudiant (pk_num_secu, ck_num_etu, nom, prenom)
tp_intro-# VALUES ('1840791123456', 'DUP1840703', 'Dupont', 'Pierre');
tp_intro=# INSERT INTO t_etudiant (pk_num_secu, ck_num_etu, nom, prenom)
tp_intro-# VALUES ('2850460789101', 'DUC2850428', 'Durand', 'Catherine');
```

Contrainte de clé. Les *contraintes de clé* se composent de contraintes d'*unicité* et de *non nullité*. Elles permettent d'assurer que toutes les valeurs d'un attribut seront différentes pour chaque enregistrement. Ainsi, chaque enregistrement peut être *identifié* par le biais de la valeur d'un attribut spécifique (ou d'une combinaison restreinte d'attributs) appelé(e) *clé*. Dans la table t_etudiant, l'attribut pk_num_secu est la *clé primaire* (PRIMARY KEY) et l'attribut ck_num_etu est une *clé candidate*, c'est-à-dire que cet attribut aurait pu être choisi comme clé primaire car il possède les propriétés requises.

Exécutez les instructions suivantes et expliquez chaque résultat.

3. Pour ce genre de modifications, il existe une instruction ALTER TABLE , nous auront l'occasion de la voir ultérieurement.

```
tp_intro=# INSERT INTO t_etudiant (pk_num_secu, ck_num_etu, nom, prenom)
tp_intro-# VALUES ('1840791123456', 'DUL1840703', 'Dupont', 'Laurent');
<message>
tp_intro=# INSERT INTO t_etudiant (ck_num_etu, nom, prenom)
tp_intro-# VALUES ('DUL2820314', 'Duchemin', 'Lila');
<message>
tp_intro=# INSERT INTO t_etudiant (pk_num_secu, ck_num_etu, nom, prenom)
tp_intro-# VALUES ('1840760147523', 'DUP1840703', 'Dupont', 'Paul');
<message>
tp_intro=# INSERT INTO t_etudiant (pk_num_secu, ck_num_etu, nom, prenom)
tp_intro-# VALUES ('2820360978862', 'DUL2820314', 'Duchemin', 'Lila');
<message>
```

Pour confirmer vos explications, regardez ce que contient désormais la table `t_etudiant`. Pourrait-on insérer dans la table `t_etudiant` un élève dont le nom et le prénom sont respectivement `'Dupont'` et `'Pierre'` ?

4 Notion de référence

Clés étrangères. Une base de données est généralement constituée de plusieurs tables liées les unes aux autres par des *clés étrangères*. Plus précisément, un attribut d'une table est utilisé pour faire *référence* à un attribut d'une autre table (généralement, la clé primaire de cette autre table). Ces attributs liés par ce genre de contraintes référentielles doivent partager le même type.

Nous allons créer une seconde table `t_inscription` et y associer des UE et des étudiants.

```
tp_intro=# CREATE TABLE t_inscription(
tp_intro(# pk_code_ue CHAR(4) NOT NULL,
tp_intro(# fk_code_etu CHAR(13) NOT NULL,
tp_intro(# PRIMARY KEY (pk_code_ue, fk_code_etu),
tp_intro(# FOREIGN KEY (fk_code_etu) REFERENCES t_etudiant(pk_num_secu));
```

```
tp_intro=# INSERT INTO t_inscription (pk_code_ue, fk_code_etu)
tp_intro-# VALUES ('IBD', '2820360978862');
tp_intro=# INSERT INTO t_inscription (pk_code_ue, fk_code_etu)
tp_intro-# VALUES ('IPF', '2820360978862');
tp_intro=# INSERT INTO t_inscription (pk_code_ue, fk_code_etu)
tp_intro-# VALUES ('IPI', '2820360978862');
```

Expliquez ces instructions et les résultats.

Contraintes d'intégrité référentielle. En définissant la table `t_inscription`, nous avons défini une contrainte dite d'*intégrité référentielle* (`FOREIGN KEY ... REFERENCES ...`).

Exécutez les instructions suivantes et expliquez le rôle d'une contrainte référentielle et ce qu'elle interdit.

```
tp_intro=# INSERT INTO t_inscription (pk_code_ue, fk_code_etu)
tp_intro-# VALUES ('IBD', '1840791123456');
tp_intro=# INSERT INTO t_inscription (pk_code_ue, fk_code_etu)
tp_intro-# VALUES ('IBD', '2850460789101');
tp_intro=# INSERT INTO t_inscription (pk_code_ue, fk_code_etu)
tp_intro-# VALUES ('IBD', '1700792001278');
```

5 Notion de projection, de restriction et de jointure

Projection. Exécutez l'instruction suivante et expliquez pourquoi elle est appelée *projection*.

```
tp_intro=# SELECT nom, prenom
tp_intro-# FROM t_etudiant;
```

Donnez l'instruction permettant d'obtenir le numéro et le nom des étudiants de la table `t_etudiant`.

Restriction. Exécutez l'instruction suivante et expliquez pourquoi elle est appelée *restriction*.

```
tp_intro=# SELECT nom, prenom
tp_intro-# FROM t_etudiant
tp_intro-# WHERE nom = 'Dupont';
```

Donnez l'instruction permettant d'obtenir le nom et le prénom de l'étudiant dont le numéro de sécurité sociale est `'2850460789101'`.

Jointure. Exécutez l'instruction suivante et expliquez pourquoi elle est appelée *jointure*.

```
tp_intro=# SELECT nom, pk_code_ue
tp_intro-# FROM t_etudiant JOIN t_inscription ON pk_num_secu = fk_code_etu;
```

Donnez l'instruction permettant d'obtenir la liste des étudiants inscrits à l'UE `'IBD'`.

Produit cartésien. Exécutez l'instruction suivante et expliquez pourquoi elle est appelée *produit cartésien*.

```
tp_intro=# SELECT nom, pk_code_ue
tp_intro-# FROM t_etudiant, t_inscription;
```

Quelle est la différence avec une jointure ?

6 Notion de fonction et d'agrégation

Afin de mieux appréhender les *fonctions* et les *agrégats*, nous allons réinitialiser la base avec des données contenues dans un fichier.

Réinitialisation de la base de données. Dans un premier temps, exécutez les instructions suivantes (dans l'ordre indiqué) afin de supprimer les données existantes dans les tables (instruction `DELETE` du `SQL LMD`). Expliquez l'origine du message d'erreur.

```
tp_intro=# DELETE FROM t_etudiant;
<message_d_erreur>
tp_intro=# DELETE FROM t_inscription;
tp_intro=# DELETE FROM t_etudiant;
```

Téléchargez les fichiers `etudiant.csv` et `ue.csv` à l'adresse suivante : `http://www.ensiie.fr/~szafranski/downloads/BD/tp1/`. Mettez les dans le répertoire dans lequel vous travaillez puis regardez leur contenu. Insérez les données avec les instructions suivantes.

```
tp_intro=# \copy ...                                        -- Tout doit être sur la même ligne
t_etudiant(pk_num_secu, ck_num_etu, nom, prenom) FROM 'etudiant.csv' WITH DELIMITER AS ';'
tp_intro=# \copy t_inscription(fk_code_etu, pk_code_ue) FROM 'ue.csv' WITH DELIMITER AS ';'
```

Fonction. Exécutez l'instruction suivante et expliquez le résultat obtenu.

```
SELECT COUNT(pk_code_ue)
FROM t_inscription
WHERE fk_code_etu = '1';
```

Agrégation. Exécutez l'instruction suivante et expliquez le résultat obtenu.

```
tp_intro=# SELECT pk_code_ue, COUNT(fk_code_etu)
tp_intro-# FROM t_inscription
tp_intro-# GROUP BY pk_code_ue
tp_intro-# ORDER BY COUNT(fk_code_etu);
```

Que permet une agrégation par rapport à une fonction ?

7 Notion de dictionnaire de données

Le dictionnaire de données permet de connaitre l'ensemble des informations des tables de la base de données, c'est-à-dire le type des attributs, les contraintes associées, etc.

1. Exécutez la commande `\d` et expliquez ce qu'elle renvoie.
2. Exécutez les commandes `\d t_etudiant` et `\d t_instruction` et expliquez les résultats.
3. Consultez l'aide `\?` et cherchez d'autres commandes d'accès au dictionnaire de données.