BE1 - Prise en main de MySQL

Wael Ben Soltana

2012 - 2013

1 Introduction

Le but de cette première partie est de se familiariser avec le fonctionnement des Systèmes de Gestion de Bases de Données Rationnelles (SGBDR). Pour cela nous allons créer une base simple permettant d'avoir un aperçu des différentes techniques que l'on met en oeuvre dans une base de données plus importante. Les manipulations seront faites sur **MySql** via son interface graphique, *MySQL Workbench*.

Commencez par télécharger puis lancer "MySQL Installer" sur le site officiel de MySQL :

http ://www.mysql.com/.

Les packages nécessaires seront *MySQL Server* et *MySQL Workbench*.

Lancez ensuite le *Workbench* puis ouvrez un onglet *SQL Development* à l'aide de la fonction *Open Connection to start querying*. Créez un schéma vierge. Il nous servira de base de travail pour apprendre à manipuler la base de données.

2 Manipulation par l'exemple

Il s'agit d'un exemple simple de gestion des colocataires. L'information concernant un locataire est son nom, son surnom (que l'on suppose unique) et son numéro de portable s'il en a un. Les ingrédients communs utilisés par les colocataires sont identifiés par son nom et sa catégorie. La quantité des ingrédients restant dans le stock est également importante. L'objectif est de construire une base de données permettant de gérer la consommation des ingrédients partagés entre les colocataires.

2.1 Tables et vues

Un exemple de composition des tables dans la base de données est la suivante (les clés sont en gras) :
- Table Colocataire : **surnom**, nom, numero de portable
- Table Ingredient : **objet**, categorie
- Table Conso : **consoID**, surnom, objet, quantite
- Table Stock : **objet**, quantite

2.1.1 Création des tables

Commencez par créer les différentes tables. Tout peut être fait via SQL (commandes CREATE TABLE et CREATE DATABASE) mais pour l'heure on utilisera l'interfaces simplifiée qui nous est proposée. La création d'objets se fait á l'aide du menu *Object Browser*. Créez les tables, les colonnes et les types de données respectivement.

2.1.2 Ajout des dépendances

Notez que le champ "**surnom**" est le même entre Conso et Colocataire. Il en est de même pour "**objet**" : Stock et Conso font référence à "**objet**" dans Ingredient.

L'édition des tables se fait á l'aide d'un clic droit et de la commande *Alter table*. La modification des *Foreign Keys* se fait dans l'onglet correspondant. Attention, l'association *Foreign Keys* porte elle-même un nom. Ne confondez pas le nom des champs associés et le nom donné à l'association.

Jusqu'à cette étape, nous estimons que vous avez créé une base de données dont la relation entre les tables est décrite comme dans la Fig 1.

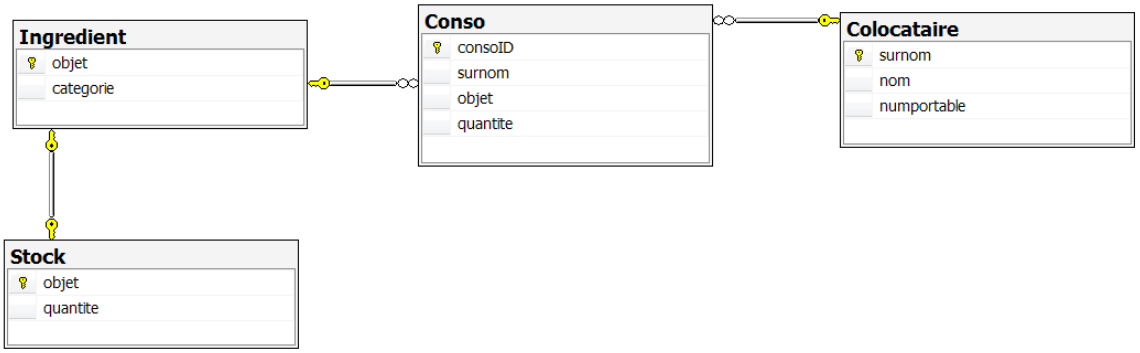

FIGURE 1 – Un exemple du diagramme de relation entre les tables.

2.1.3 Ajout/modification/suppression des valeurs de données

C'est le moment de tester quelques requètes en SQL.

– **Ajout** : peuplez un peu les tables avec quelques lignes de données.

```
USE dbname;
INSERT INTO tblname ( colname, ... ) VALUES ( val, ... );
```

– **Modification** : mettez à jour par exemple un nouveau numéro de portable d'un colocataire.

```
USE dbname;
UPDATE tblname SET colname1 = val1 WHERE colname2 = val2;
```

– **Suppression** : le contenu entier ou partiel d'une table est supprimé.

```
USE dbname;
DELETE FROM tblname WHERE colname1 = val1 AND colname2 = val2 ;
```

Vous pouvez remarquer que le système vous retournera une erreur si vous entrez des lignes incohérentes d'une table à l'autre.

2.1.4 Création d'une vue

Ces tables contiennent toutes les données dont on a besoin pour notre application. Ceci dit on va souvent afficher une donnée critique : *l'état du bar*. Etant donné qu'on l'affiche souvent et que l'on va peut être même faire des requètes dessus, c'est intéressant de créer une vue. Créez une vue correspondant à la requète suivante :

```
SELECT Stock.objet, quantite FROM Stock, Ingredients
WHERE Stock.objet = Ingredients.objet AND categorie = 'boisson';
```

2.2 Procédures stockées (stored procedure)

Il est possible de stocker une opération qui va être exécutée fréquemment, sous forme une fonction avec des **paramètres d'entrée** et de **sortie**.

Dans notre exemple, une opération qui va être exécutée fréquemment sera de ***"consulter les consommations de chacun"***. Ceci nécessitant un paramètre (*le nom de la personne*), on ne peut l'effectuer avec une vue. Il va donc falloir ajouter une procédure stockée qui effectue la tâche suivante :

```
SELECT * from Conso WHERE surnom = [ parametre ];
```

Sous MySQL, les paramètres sont utilisés avec un nom défini par l'utilisateur, à l'aide des commandes DECLARE. Leur affectation se fait à l'aide de la fonction SET. Attention, n'utilisez pas de nom existant !

- vous pouvez rajouter des paramètres sur le modèle suivant :

```
// MySQL exécute ligne par ligne
// un délimiteur sert à marquer des blocs de lignes qui seront exécutés ensemble
     DELIMITER $$
     $$
     CREATE PROCEDURE proc_name( IN parametre TYPE, ..., OUT parametreN TYPE )
     BEGIN
     ...
     END
     $$
```

Pour exécuter les procédures stockées sous MySQL, on applique la syntaxe suivante :

```
CALL prog_name( param, ... );
```

2.3 Déclencheurs (trigger)

Il s'agit d'un objet de base de données **associé à une table**, qui s'active lorsqu'un évènement particulier survient.

Il va maintenant falloir faire quelque chose pour gérer la diminution des stocks : il est normal que lorsqu'un ingrédient est consommé, il faille le retirer des stocks. Ceci peut se faire via un ***déclencheur (trigger)*** qui se déclenchera sur une insertion dans la table Conso.

Voici en exemple la résolution de ce problème de mise à jour du stock sous MySQL. Elle vous donne une bonne idée du genre de code qu'on peut mettre dans un déclencheur.

```
CREATE TRIGGER testTrigger BEFORE INSERT ON Conso FOR EACH ROW
BEGIN
  DECLARE qtConso integer;
  DECLARE prodConso varchar(50);
  DECLARE restant integer;
  DECLARE reste integer;

  SET qtConso = NEW.quantite;
  SET prodConso = NEW.objet;
  SELECT quantite INTO restant FROM Stock WHERE objet = prodConso;
  SET reste = restant - qtConso;

  IF reste<0 THEN
    SET NEW.quantite = restant;
    UPDATE Stock SET quantite = 0 WHERE objet = prodConso;
  ELSE
    UPDATE Stock SET quantite = reste WHERE objet = prodConso;
  END IF
END
```

On peut constater que dans ce déclencheur, les champs *quantite* et *objet* sont extraits depuis la table **"NEW"**. La table "NEW" est une table virtuelle contenant tous les champs et les valeurs de la commande "INSERT" ou "UPDATE" actuelle.

Le but est maintenant de générer une liste de courses : créez une nouvelle table "**Courses**" et écrivez un déclencheur qui ajoutera les ingrédients dans cette table lorsque son stock atteindra 0. Dans ce cas, le déclencheur est associé à un évènement "UPDATE". L'instruction SQL correspondante est :

```
CREATE TRIGGER testTrigger ON Stock FOR UPDATE AS
... INSERT INTO Courses Values valeurs_a_inserer; ...
```

2.4 Utilisateurs

Le souci dans l'histoire, c'est que tout utilisateur peut effectuer tout type d'opérations sur la base de données. On peut par exemple supprimer toute trace de sa consommation abusive d'alcool via une instruction SQL "delete" sur la table Conso. On considèrera donc que l'administrateur est honnête (disons qu'on limite les dégâts en n'autorisant qu'une seule personne à tricher) et on interdira aux utilisateurs de base de faire des "delete".

En revenant à l'écran d'accueil du Workbench, il est possible de gérer les permissions des utilisateurs en particulier. Par exemple, pour empêcher à un utilisateur de faire des "delete" sur les tables, il y a deux façons de le faire :

- Soit par le biais de GUI :
  - Depuis le bouton "Manage Security" au niveau de l'écran d'accueil du Workbench.

- Soit par requète SQL, dont voici un exemple.

```
CREATE USER 'log_name'@'localhost'  IDENTIFIED BY 'log_pass';
GRANT SELECT, INSERT ON  etude.* TO 'log_name'@'localhost';
GRANT USAGE ON *.* TO 'log_name'@'localhost' WITH MAX_QUERIES_PER_HOUR 90;
```

3 Projet

Constituez des groupes d'au **maximum 3 personnes**. Essayez d'avoir une personne sachant coder dans chaque groupe. Le SGBD est laissé au choix, des bibliothèques pour pouvoir écrire plus simplement l'application en C++ seront fournies. L'application peut autrement être codée en utilisant Java, C#, Visual Basic.net ou PHP.

On attend du compte rendu final les éléments suivants :

- Proposer une infrastructure matérielle (machines, logiciel, réseau) adaptée à leur projet (une compagnie aérienne et la bibliothèque du coin n'ont pas de besoin du même materiel) et la chiffrer.

- Proposer un schema d'administration (combien d'administrateurs, quels droits pour les utilisateurs de la base de données etc.).

- Proposer une architecture pour la base de données elle-même : schemas des tables, justification des choix, description du système. Vous devez vendre votre conception et convaincre le client que vous avez répondu à ses besoins de la meilleure façon possible. Ce travail sera partiellement réalisé en salle lors de la 2e séance (BE2).

- Architecture logicielle : specifications et détail de la réalisation du programme qui met en place la solution. Ce travail sera partiellement réalisé en salle lors de la 3e séance (BE3).