Département Informatique

Laboratoires de bases de données

Laboratoire n°6
Programmation SQL

par Danièle BAYERS et Louis SWINNEN

Mars 2011

La programmation SQL

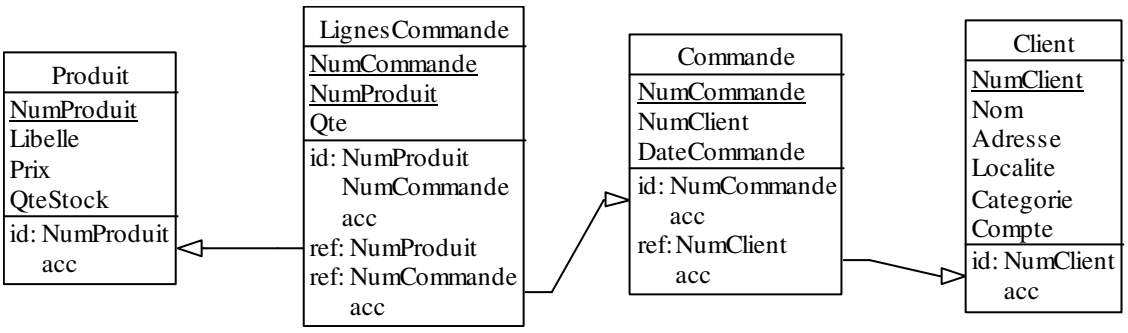
1. Introduction

Les bases de données comme SQL Serveur et Oracle permettent, en plus de stocker des données, d'y ajouter des procédures. Ces procédures sont programmées dans un dialecte SQL qui est propre au SGBD ainsi, en Oracle, la programmation se fait en PL/SQL tandis que sous SQL Serveur, la programmation se fait en Transact-SQL (T-SQL).

On distingue généralement *les procédures stockées* et *les déclencheurs*. Les *procédures stockées* (ou *stored procedures*) sont des procédures attachées à une base de données. L'intérêt principal étant de concentrer les traitements lourds sur les données au niveau du SGBD. Ainsi, lorsque des mises à jour importantes doivent être effectuées sur les données, il est souvent plus aisé de réaliser les modifications par procédure stockée que via une programmation extérieure dans un langage de programmation classique.

Les *déclencheurs* (ou *triggers*) sont des procédures particulières qui se déclenchent automatiquement lorsqu'un événement précis survient, comme l'insertion d'un enregistrement dans une table. Les déclencheurs sont donc attachés à une table particulière et liés à un événement.

2. La base de données



3. Les procédures stockées

Une procédure stockée s'exécute uniquement lorsqu'elle est appelée. Comme elle n'est pas standardisée, sa définition dépend du SGBD employé. Comme toutes les procédures programmées, la procédure stockée peut admettre des paramètres. Elle est écrite dans le dialecte SQL imaginé par le créateur du SGBD. Dans la suite, nous détaillerons la création de procédures stockées en Transact-SQL (sous SQL Server) et, en annexe, en PL/SQL (sous Oracle).

Les procédures stockées utilisent des structures classiques comme des boucles ou des sélections. Une force de la procédure stockée est d'autoriser des requêtes directement.

Les procédures stockées utilisent parfois **des curseurs** que nous détaillerons plus loin.

3.1 Procédure stockée en SQL Serveur

Format :

```
CREATE PROCEDURE nom_procedure
   [@arg1  type [OUTPUT], …, @argn  type [OUTPUT] ]
AS
   DECLARE @Var1 type
   .
   DECLARE @Varn type
   .
   .
   .
```

Dans le format précédent, on remarque que la procédure `nom_procedure` est créée au moyen d'une commande `CREATE PROCEDURE`. La suppression d'une procédure stockée existante se fait au moyen de `DROP PROCEDURE`.

On trouve ensuite les arguments éventuels. Pour chaque argument, il faut mentionner son **nom**, son **type** et s'il s'agit d'un **paramètre en entrée** (absence de `OUTPUT`) ou **en sortie** (`OUTPUT` – initialisé par la procédure). Par défaut, les paramètres sont considérés comme des paramètres d'entrée. En Transact-SQL, le type de l'argument doit être précisé complètement. Ainsi, il faut mentionner la taille maximale également comme VARCHAR(30).

Particularité : il est possible de mentionner une valeur par défaut pour un paramètre en mentionnant, à la suite de la définition de type, le symbole '=' et la valeur par défaut.

Exemple :

```
CREATE PROCEDURE sp_inc_stock (
      @num_produit CHAR(5),
      @qte_act INTEGER OUTPUT)
AS
   DECLARE @Qte INTEGER

   SELECT @Qte = QteStock
     FROM Produit
     WHERE NumProduit = @num_produit
   SELECT @Qte = @Qte + 1

   UPDATE Produit
     SET QteStock = @Qte
     WHERE NumProduit = num_produit
   SELECT @qte_act = @Qte

-- exécution :
DECLARE @RESULT DECIMAL(2,1)
EXEC SP_INC_STOCK 'P001', @RESULT OUTPUT
GO
```

Dans cet exemple, la procédure `sp_inc_stock` permet d'incrémenter le stock pour le produit `num_produit` d'une unité.

4. Programmation SQL

Suivant le SGBD utilisé, il est tout a fait possible de programmer en SQL en utilisant les structures habituelles comme la sélection, la répétition ou encore l'affectation.

4.1 En SQL Server

- Affectation :

```
SELECT @Var = expression
```

- Sélection :

```
IF condition
BEGIN
.
END
ELSE
BEGIN
.
END
```

- Boucle :

```
WHILE condition
BEGIN
.
END
```

- Affichage à l'écran :

```
PRINT 'information'
```

5. Les curseurs

En programmation SQL (comme dans les déclencheurs et les procédures stockées), lorsqu'une requête retourne plusieurs lignes de résultat, il est possible de traiter chaque ligne au moyen d'un curseur.

Il s'agit donc d'un mécanisme qui va permettre de parcourir chaque enregistrement dans le résultat pour effectuer une opération précise.

L'implémentation des curseurs dépend fortement du SGBD employé. Ils sont disponibles sous Oracle et SQL Server.

5.1 Les curseurs sous SQL Server

Format :

```
DECLARE nom_curseur CURSOR
FOR
   SELECT attr1, …, attrn
   FROM table
    [JOIN …]
    [WHERE condition]
    [GROUP BY …]
    [HAVING …]
    [ORDER BY …]
```

Le curseur est déclaré comme toutes variables avec le mot clé `DECLARE`. Le type de cette variable est `CURSOR` et ensuite, on trouve la requête. La requête peut contenir toutes les options que nous avons déjà vues.

Une fois le curseur déclaré, il est possible de l'utiliser. On distingue les étapes suivantes :

1. Ouverture du curseur au moyen de la commande SQL OPEN :
   ```
   OPEN nom_curseur
   ```

2. Comme dans la lecture d'un fichier, on commence par lire le 1ier résultat :
   ```
   FETCH nom_curseur INTO @Var1, …, @Varn
   ```
   Il faut remarquer que tous les attributs présents dans la requêtes sont affectés à une variable via INTO.

3. Ensuite, la boucle de parcours est écrite et les instructions traitant le résultat sont contenues à l'intérieur.
   ```
   WHILE @@fetch_status = 0
   BEGIN
     Traitement du résultat
   ```

4. Avant la fin de la boucle, il faut procéder à la lecture du résultat suivant :
   ```
     FETCH nom_curseur INTO @Var1, …, @Varn
   END
   ```

5. Une fois la lecture terminée, on ferme le curseur
   ```
   CLOSE nom_curseur
   ```

6. Si le curseur n'est plus utile, il convient de libérer la ressource :
   ```
   DEALLOCATE nom_curseur
   ```

Exemple : Un exemple complet est montré dans la partie déclencheur.

6. Les déclencheurs

Pour gérer des contraintes complexes au niveau du SGBD, il est souvent nécessaire de recourir aux déclencheurs (ou *triggers*). Un déclencheur est une procédure programmée au niveau du SGBD qui s'exécute **automatiquement** lorsqu'un événement précis survient.

6.1 Les événements

Les événements *déclencheurs* sont l'ajout, la suppression ou la mise à jour d'un enregistrement dans la table ou même d'une colonne particulière. Les déclencheurs ne sont pas normalisés dans la norme SQL-2, il convient dès lors de se référer au dialecte de son SGBD (PL/SQL pour Oracle ou Transact-SQL pour SQL Server). Les événements déclencheurs sont généralement :

- `BEFORE INSERT` : *exécution du code avant une insertion*
- `BEFORE UPDATE` : *exécution du code avant une modification*
- `BEFORE DELETE` : *exécution du code avant une suppression*
- `AFTER INSERT` : *exécution du code après une insertion*
- `AFTER UPDATE` : *exécution du code après une mise à jour*
- `AFTER DELETE` : *exécution du code après une suppression*

Sous SQL Server, les triggers BEFORE ne sont pas disponibles.

Les événements *BEFORE* sont souvent utilisés pour vérifier une contrainte particulière et éventuellement **arrêter l'insertion, la mise à jour ou la suppression** si la contrainte n'est pas respectée.

Les événements *AFTER* sont souvent utilisés pour mettre à jour des données en fonction de la demande : mettre à jour un solde, un stock, … suite à l'ajout, la modification ou la suppression d'un enregistrement, par exemple.

Les déclencheurs sont également très souvent utilisés pour maintenir une certaine « dénormalisation » du schéma de la base de données. Ainsi, les *attributs dérivables* (i.e. résultat d'une opération entre des informations présentes dans la base de données comme le total d'une commande par exemple) peuvent, pour des questions de performances, être présents dans le schéma du SGBD. Afin d'assurer la cohérence des informations et être sûr que ces attributs dérivables sont toujours à jour, les déclencheurs sont alors très souvent utilisés.

Finalement, il est possible de **restreindre l'exécution** du déclencheur à l'ajout, la modification, ou la suppression d'**un champ particulier** dans la table.

6.2 Les déclencheurs sous SQL Server

Format :

```
CREATE TRIGGER nom_trigger ON nom_table
   (FOR|AFTER|INSTEAD OF) (INSERT [,] | UPDATE [,] | DELETE [,])+
   AS
     [IF [NOT] UPDATE (attrx)
        BEGIN
        END]
     [DECLARE @Var1 type, …, @Varn type]
       .
       .
       .
```

La déclaration d'un déclencheur commence par les mots CREATE TRIGGER. Il faut ensuite spécifier le nom du déclencheur (`nom_trigger`) et la table sur laquelle il porte (`nom_table`). Il faut ensuite préciser l'événement déclencheur. Enfin, le code du déclencheur commence. Si on souhaite limiter un déclencheur à un champ particulier, il est nécessaire de spécifier l'option IF [NOT] UPDATE et l'attribut concerné. Cette option **n'est possible que pour des déclencheurs INSERT et UPDATE**. Exemple de syntaxe :

```
IF NOT UPDATE (attr)
     RETURN
```

On trouve ensuite la déclaration des variables et le code du déclencheur lui-même.

6.2.1 Evénements déclencheurs

SQL Server permet les options suivantes :

- `FOR` ou `FOR AFTER` ou `AFTER` : Ces déclencheurs s'exécutent après que l'instruction SQL ait été complètement exécutée. Le déclencheur ne sera exécuté que si l'instruction SQL s'est passée correctement. En particulier, si toutes les contraintes programmées (contraintes d'intégrité, contrainte référentielle, clause CHECK) ont été vérifiées. `FOR`, `FOR AFTER` et `AFTER` sont des synonymes.

Ceci est un exemple, cliquez sur le lien de téléchargement pour obtenir le cours complet.