SQL Intégré (à l'algorithmique)

Présentation

SQL Intégré est une technique permettant d'introduire des instructions SQL (seulement du LMD) dans un programme en langage classique dans un environnement non client-serveur.

Le programme qui contient des instructions SQL est appelé programme hôte.
La communication entre le programme hôte (non SQL) et les instructions SQL passe par des variables communes : ce sont les variables hôtes. Les variables hôtes peuvent être à la fois utilisées par le programme hôte et par les instructions SQL.

Ces variables particulières sont déclarées dans une section particulière du programme hôte, après les variables "traditionnelles" qui ne seront pas utilisées par SQL. En algorithmique, on fait commencer cette section tout simplement par l'expression Variable(s) hôte(s).

Les variables hôtes sont utilisées par le programme hôte comme toutes les autres variables. Dans les instructions SQL, les variables hôtes utilisées sont précédées d'un préfixe **(:** en algo, @ avec SQL Server) pour les différencier des noms d'attributs des tables.

Pour les distinguer du reste du code, les instructions SQL sont entourées par les mots clé SQL et finSQL. Si l'instruction SQL ne fait qu'une ligne, il suffit de placer SQL au début de la ligne (pas besoin alors de finSQL).

Utilisation de SQL intégré

❑ Pour traiter un t-uple, résultat d'une requête

Pour placer le résultat d'une requête à l'intérieur d'une variable hôte, on utilise la clause INTO suivi de la (ou des) variables hôtes à valoriser. La requête dont on veut extraire le résultat peut être paramétrée par une ou plusieurs variables hôtes.

```
SQL
    SELECT attribut1 INTO :variable_hôte1,  attribut2 INTO :variable_hôte2, …
    FROM table(s)
    WHERE attribut = :variable_hôte
finSQL
```

La clause INTO ne fonctionne que si la requête ne retourne qu'une seule ligne. Si la requête ne retourne aucune ligne ou si au contraire elle retourne plusieurs lignes, l'affectation ne va pas pouvoir se faire : une erreur va apparaître. Il faut traiter ces cas, soit en faisant l'hypothèse qu'il n'y a jamais d'erreur, soit en indiquant les traitements à réaliser le cas échéant. Les erreurs sont signalées par le SGBD au programme hôte par l'intermédiaire d'une variable hôte spéciale, avec un nom spécifique (SQLCODE), qui prend une valeur spécifique (100) dans le cas où la requête précédente n'a renvoyé aucune ligne. Pour les autres types d'erreurs, il n'existe pas de codification standard : soit la notation est précisée dans l'énoncé, soit c'est à vous de préciser celle que vous utiliser.

❑ Pour mettre à jour une base de données

On peut insérer des t-uples dans une base de données à partir de données tirées du programme hôte.

SQL
INSERT INTO *table*
VALUES :*variable_hôte1*, :*variable_hôte2*, …
finSQL

On peut aussi modifier des t-uples à partir de données tirées du programme hôte.

SQL
UPDATE *table*
SET *attribut = :variable_hôte*
WHERE *attribut = :variable_hôte*
AND …
finSQL

Exemple complet

Soit le schéma relationnel simplifié suivant :

CLIENT (clt_num, clt_nom, clt_prenom, clt_ca)
COMMANDE(com_clt_num #, com_prod_ref# , com_qte)
PRODUIT(prod_ref, prod_libelle, prod_prix)

Pour simplifier, nous supposons qu'une commande ne porte que sur un seul produit.
Le numéro de client est un entier séquentiel

Nous allons écrire un algorithme utilisant SQL intégré pour traiter l'arrivée d'une commande. Les informations saisies pour une commande sont le nom et le prénom du client ainsi que la référence et la quantité du produit désiré. L'algorithme doit permettre de mettre à jour les tables client (maj du chiffre d'affaire) et commande (insertion d'une nouvelle commande dans la base). Si le client n'existe pas, il est nécessaire de le créer. On suppose que deux clients ne peuvent pas porter le même nom et le même prénom.

Programme exSQLintégré

VARIABLES :
//déclarer ici les variables utilisées seulement dans le programme hôte et pas dans les
//instructions SQL. Il n'y a pas de telles variables ici.

VARIABLES HOTES :
nom, prenom, refprod : chaînes
qte, num : entier
SQLCODE : entier //variable permettant de communiquer les erreurs de SQL au programme
ca : réel

DEBUT
Afficher "Nom et prénom du client : "
Saisir nom, prenom
Afficher "Référence du produit et quantité désirée"
Saisir refprod, qte

```
SQL
    SELECT clt_num INTO :num
    FROM CLIENT
    WHERE clt_nom = :nom AND clt_prenom = :prenom
finSQL
Si SQLCODE = 100 Alors // aucune ligne retournée par la requête
   //création du nouveau client
   SQL
       SELECT max(clt_num) INTO :num      //on récupère le plus grand numéro de client
       FROM CLIENT;
  finSQL
  num ← num + 1 // le numéro du nouveau client est le numéro du dernier + 1
  SQL
       INSERT INTO CLIENT                 //on ajoute le nouveau client dans la table CLIENT
       VALUES :num, :nom, :prenom, 0;   //le chiffre d'affaire est initialisé à 0 (maj plus bas)
   finSQL
FinSi
//mise à jour du chiffre d'affaire du client
//il faut d'abord calculer le montant de la commande
// on cherche le prix du produit
SQL
  SELECT prod_prix INTO :ca
  FROM PRODUIT
  WHERE prod_ref = :refprod
finSQL
Si SQLCODE = 100 Alors  //si le résultat de la requête n'a aucune ligne
  Afficher "Erreur : la référence du produit n'existe pas"
  Sortir
FinSi
// on calcule le chiffre d'affaire de la commande= prix (résultat de la requête)* quantité
ca ← ca * qte
// on met à jour le chiffre d'affaire du client
SQL
  UPDATE CLIENT
  SET clt_ca = clt_ca + :ca
  WHERE clt_num = :num
finSQL
//ajout de la commande dans la table COMMANDE
SQL
  INSERT INTO COMMANDE
   VALUES :num, :refprod, :qte
finSQL

 SQL COMMIT WORK
  //indique la fin de la transaction (les mises à jour deviennent permanentes)
FIN
```

❑ les curseurs : pour traiter le résultat d'une requête contenant plusieurs tuples

Rappel sur les curseurs :

Un curseur représente un ensemble de tuple résultat d'une requête.
Curseur et jeu d'enregistrement sont synonymes. La différence entre ces deux notions provient du langage qui les manipule. Les curseurs sont manipulés par des instructions en langage SQL, alors que les jeux d'enregistrement sont manipulés par des instructions spécifiques du langage hôte (ex: les RecordSet en VB).
La manipulation des curseurs suit le standard SQL alors que la manipulation des jeux d'enregistrement dépend des langages. (A l'étude de cas du BTS, lorsque ce n'est pas précisé dans l'énoncé, et que vous devez traiter le résultat d'une requête comportant plusieurs tuples, utilisez les curseurs et la notation SQL).

On peut comparer une curseur à une sorte de fichier séquentiel défini par un ordre SQL, qu'il est possible d'ouvrir (OPEN) et de lire en séquentiel (FETCH).

Pour utiliser un curseur :

- DECLARATION

Déclarer le ou les curseurs utilisés dans une section de déclaration spécifique intitulée Curseur(s). Attention, la déclaration n'entraîne pas l'exécution de la requête associée.

```
Curseur(s) :

DECLARE moncurseur CURSOR FOR
SELECT …
```

- OUVERTURE

Elle **permet d'exécuter la requête** correspondant au curseur. Le résultat de la requête est affecté au curseur, et le "pointeur" du curseur est positionné juste avant le premier tuple.

```
SQL OPEN nomcurseur
```

- LECTURE

La lecture par l'instruction FETCH permet de copier le tuple suivant (tuple courant) dans les variables hôtes déclarées préalablement (à chaque champ du tuple doit correspondre une variable hôte). Lorsqu'il n'y a plus de tuple à lire, l'instruction FETCH affecte la valeur 100 à la variable SQLCODE.

```
//lecture du premier tuple
SQL FETCH moncurseur INTO :variablehote1, : variablehote2, :variablehote3
Tantque SQLCODE ? 100 Faire                    // tant que la lecture est possible
        // traitement sur le tuple par l'intermédiaire des variables hôtes
        …
        //puis lecture du tuple suivant
        SQL FETCH moncurseur INTO :variablehote1, : variablehote2, :variablehote
FinTq
```

- FERMETURE

Il faut fermer les curseurs (tout comme il faut fermer les fichiers), pour désallouer les ressources mémoires utilisées.

```
SQL CLOSE moncurseur
```