PL/PGSQL

Introduction

- SQL est un langage ensembliste
- Il lui manque un certain nombre de structures de données et de contrôle
  - Utilisation de langages hôtes (Java, PHP...)
- L'option procédurale (encore appelée "procédures stockées") apporte une solution intégrée aux SGBD

Introduction

- PL/SQL : solution Oracle
- Dérivée dans les autres SGBD : PL/PGSQL
- PL/SQL = SQL+
  - itérations
  - conditionnelles
  - procédures et fonctions
  - ...

I : Structure d'un programme

Structure

```
CREATE OR REPLACE FUNCTION nom(parametres)
RETURNS type
AS $$ DECLARE
  Déclaration de variables
BEGIN
  Section obligatoire contenant
  des instructions SQL et PL/SQL
END; $$
LANGUAGE ‘plpgsql’
```

Exemple

```
DECLARE
  qteStock NUMBER(5);
BEGIN
  SELECT qte INTO qteStock FROM inventaire
  WHERE libelle='stylo';
  IF qteStock > 0 THEN
    UPDATE inventaire SET qte=qte-1
    WHERE libelle='stylo';
    INSERT INTO vente VALUES('stylo',SYSDATE);
  ELSE
    INSERT INTO aCommander('stylo',SYSDATE);
  END IF;
END;
```

2 : Les déclarations

Généralités

- La partie déclarative est délimitée par DECLARE et BEGIN
- On retrouve les types SQL
- 3 types de déclaration
  - Variables et constantes
  - Curseurs

Variables et constantes

- Référencer une colonne d'une table :
  - **`DECLARE var TABLE.COLONNE%TYPE;`**
- Référencer une ligne d'une table :
  - **`DECLARE var TABLE%ROWTYPE;`**
  - Type **`RECORD`**
- Paramètre de fonction :
  - **`DECLARE numero ALIAS FOR $1`**

Variables et constantes

```
total NUMBER(9,2);
nom varchar(20) := ‘toto’;
numero EMPLOYE.noEmp%TYPE;
emp EMPLOYE%ROWTYPE;
prenom nom%TYPE;
PI CONSTANT integer := 3.14;
```

Assignement de valeur

- Opérateur d'affectation : `:=`
- Clause `SELECT ... INTO`
  - OK : si `SELECT` retourne une et une seule ligne
  - `FOUND` : variable booléenne

3 : Les structures de contrôle

Traitements conditionnels

```
IF condition
  THEN instructions
  [ELSE instructions]
  [ELSEIF condition
    THEN instructions
    [ELSE instructions...]]
END IF;
```

Traitements itératifs

```
LOOP
  instructions
END LOOP;
```

Les commandes `EXIT` et `EXIT WHEN condition` permettent de sortir de la boucle

Traitements itératifs

```
WHILE condition LOOP
  instructions
END LOOP;
```

Traitements itératifs

```
FOR compteur IN [REVERSE] varDebut..varFin [BY expression] LOOP
  instructions
END LOOP;
```

Incrémentation de `1` ou de `expression`

Traitements itératifs

```
FOR record IN expression_select LOOP
  instructions
END LOOP;
```

Itérer sur le résultat d'un `select`

4 La gestion des erreurs

Introduction (1/2)

- Gestionnaire d'exceptions
  - Affecter un traitement approprié aux erreurs
- 2 types d'erreurs
  - Erreurs internes
  - Anomalies utilisateur

Introduction (2/2)

- Principes :
  - Donner un nom à l'erreur
  - Définir les anomalies et leur associer un nom
  - Définir le traitement à effectuer

Les exceptions internes

- Surviennent lorsqu'un bloc PL/SQL viole une règle du SGBD ou du Système d'Exploitation
- `TOO_MANY_ROWS, ZERO_DIVIDE...`

Les exceptions internes

```
DECLARE
   sal employe.salaire%TYPE;
BEGIN
   SELECT salaire INTO sal FROM employe;
EXCEPTION
   WHEN TOO_MANY_ROWS THEN ...;
   WHEN NO_DATA_FOUND THEN ...;
   WHEN OTHERS THEN ...;
END;
```

Les exceptions utilisateur

```
RAISE EXCEPTION 'Nonexistent ID --> %',
user_id

RAISE WARNING ‘Attention…’

RAISE NOTICE ‘Affichage d’’un message’
```

5 Les curseurs

Définition

- Pour traiter un ordre SQL, PL/SQL ouvre une zone de contexte pour stocker le résultat
- Un curseur permet :
  - de nommer cette zone
  - d'accéder aux informations
- C'est une zone mémoire utilisée par le noyau

Utilisation d'un curseur

- 4 étapes
  - Déclaration
  - Ouverture
  - Traitement des lignes
  - Fermeture

Déclaration d'un curseur

Partie DECLARE

```
CURSOR nomCurseur [(nomParametre type [,nomParametre
 type…)]]
 FOR ordre SELECT
```

Exemple

```
DECLARE
  dept10 CURSOR FOR
  SELECT nom,salaire FROM employe WHERE noDept=10;
```

Ouverture et fermeture d'un curseur

- L' ouverture permet :
  - L'allocation mémoire du curseur
  - Analyse syntaxique et sémantique de l'ordre SQL
  - Exécution
- La fermeture permet :
  - La libération de l'espace mémoire

Ouverture et fermeture d'un curseur

```
OPEN nomCurseur [(nomParametre [,nomParametre
…)]];

CLOSE nomCurseur;
```

Le traitement des lignes

Traiter les lignes une par une et renseigner les variables réceptrices

```
 FETCH nomCur INTO {nomVar[,nomVar...] | nomRecord};
```

Exemple

```
BEGIN
  OPEN dept10;
  FETCH dept10 INTO vnom, vsalaire;
    ...
END;
```

Le traitement des lignes

Possibilité de :

- parcourir dans les 2 sens : `next` et `prior`
- se positionner en début ou fin : `first` et `last`
- se positionner à un emplacement précis : `absolute` et `relative`

Une nouvelle structure itérative

```
DECLARE
   c1 CURSOR FOR SELECT nomEmp,salaire FROM employe;
BEGIN
   FOR c1record in c1 LOOP
      IF c1record.salaire > 3000 THEN ...
   END LOOP;
END;
```

6. Les triggers

Les triggers (1/4)

- Ensemble de traitements PL/SQL
- Déclenchés automatiquement par un ou plusieurs événements pré-définis
- Attachés à une et une seule table
- Si une table est supprimée : suppression des triggers associés

Les triggers (2/4)

- Un trigger est défini par :
  - Un séquencement (`BEFORE` ou `AFTER`)
  - Un événement (`INSERT`, `UPDATE`, `DELETE`)
  - Le traitement

Les triggers (3/4)

```
CREATE TRIGGER nom
séquencement événement ON nomTable

FOR EACH ROW EXECUTE PROCEDURE
nomFonction;
```

Les triggers (4/4)

- Audit particulier
- Vérification de l'intégrité référentielle (BD distribuées)
- Implantation de règles de sécurité complexes
- Maintenance de tables miroirs

Variables

- Lors de chaque exécution des variables sont créées
  - `TG_OP` : type de l'opération (insert, update ou delete)
  - `TG_WHEN` : BEFORE ou AFTER
  - `NEW` et `OLD` : nouvelle et ancienne ligne (`RECORD`)