PL/SQL

Motivation :

->SQL est un langage non procédural

->Difficile d’écrire une requête complexe sans variables, structures de contrôles de programmation

->Nécessité de lier à un langage procédural des requêtes SQL

->PL/SQL est un langage propriétaire d’Oracle

->A voir comme une extension de SQL, une requête cohabite avec les structures de contrôles

->Un programme est constitué de procédures et de fonctions

->L’échange d’information entre les requêtes SQL et le programme s’effectue par des variables

-> D’autres SGBD utilisent des langages proches de PL/SQL

Utilisation :

->Ecrire des procédures stockées et des triggers.

->Pour écrire des fonctions utilisateurs à utiliser dans des requêtes SQL

Structure d’un programme

Un programme est structuré en bloc de 3 types :

* + - * Procédures anonymes
      * Fonctions nommées
      * Procédures nommées

Un bloc peut contenir d’autre blocs :

DECLARE

--Definition des variables

BEGIN

--Instruction à réaliser

EXCEPTION

--récupération des erreurs

END;

BEGIN et END sont obligatoires

“;” a la fin d’une instruction et d’un bloc

Variables :

* Identtification des variables :
  + 30 caractères au plus
  + Commencent par une lettre
  + Peut contenir lettre, chiffres,-,$,#

->non sensible à la casse

->les portées sont standars

->doivent être déclarés avant d’être utilisé

->les types de variables correspondent aux types SQL2.

Ex : integrer, vector...

->types composites adaptés à la récupération des colones ou des types de tables SQL

%TYPE, %ROWTYPE

->type référence, REF

->Déclaration d’une variable

Identifiant [CONSTANT], type [:=valeur];

Ex :

qye integer;

Nom varchar(30); !DECLARATION MULTIPLE INTERDITE!

DateN date;

Ok boolen := true;

Utilisation %TYPE %ROWTYPE

On peut déclarer qu’une variable est de même type qu’une colonne d’une table ou d’une vue

Nomvariable relation.attribut%TYPE

Une variable peut aussi contenir toutes les colonnes d’une ligne d’une table

Ex :

etu Etudiant%ROWTYPE;

Etu contiendra une ligne de la relation etudiant.

Type RECORD

-> <=> au struct en C

->Type NomR is RECORD (

champ1 type1;

champ2 type2;

…

);

Affectation

-> 2 principaux moyens d’affecter :

* :=
* Par la directive INTO la requete Select

Ex :

-dateN := `10/04/1964`;

-Select Name INTO nom from Emp Where Matricule = 499;

Exemple :

employe emp%ROWTYPE;

nom emp.Name%TYPE;

Select \*INTO employe from emp where matr = 500;

nom := employe.Name;

employe.dept:=20;

…

INSERT into emp values employe;

Si une variables porte le même nome qu’une colonne d’un table c’est la colone qui prédomine

DECLARE

Nome emp.Nome%TYPE=’Dupond’;

BEGIN

Delete from emp where Nome = Nome;

->!Ne pas donner aux variables le meme nom qu’un attribut d’une table!

Structure de contrôle

-> conditionnelles:

* IF contions THEN

Instructions;

END

* IF conditions THEN

Instructions;

ELSE

Instruction2;

END;

* CASE Expression

WHEN expr1 THEN inst1;

WHEN expr2 THEN inst2;

…

ELSE instructionN

ENDCASE;

* LOOP

Instructions;

EXIT WHEN conditions;

Instructions;

ENDLOOP;

* WHILE condtions LOOP

Instructions;

ENDLOOP;

* FOR compteur IN[REVERSE] inf...sup LOOP

Instructions;

ENDLOOP

Ex :

For i IN 1...100 LOOP

Somme := Somme+i;

ENDLOOP;

Extraction de Données :

Select expr1,expr2,… into var1, var2,… from...

Met les valeurs de la BD dans les variables var1, var2,…

!Le select ne doit retourner qu’une seule ligne(=> sinon curseur)!

Cas d’erreur

TOO\_MANY\_ROWS : le select retourne plusieurs lignes

NO\_DATA\_FOUND : le select retourne rien

Modifications des données

* Les requêtes SQL insert, update, Delete peuvent utilsier les variables PL/SQL
* Les commit et rollback doivent être explicite

Ex d’ajout :

Declare

V\_emp emp%ROWTYPE;

V\_nom emp.Nome%TYPE;

BEGIN

V\_nom = ‘DUPOND’;

INSERT into emp(matr, nome) values (600,v\_nom);

V\_emp.matr :=610;

V\_emp.nome:=’Durand’;

INSERT into emp(Matr,Nome) values(V\_emp.Matr,V\_emp.nome);

Commit;

END

Curseurs

* Toutes requêtes SQL est associé à un curseur
* Ce curseur représente la zone mémoire utilisée pour parser et exécuter la requête
* Le curseur peut etre implicite (Non déclaré par l’utilisateur) ou explicite
* Les curseurs servent à retourner plusieurs lignes avec un Select.
* Tous les curseurs ont des attributs que l’utilisateurs peut utiliser

%ROW COUNT : NB de lignes traitées par le curseurs

%FOUND : vrai, si au moins une ligne a été traité par la requete

%NOTFOUND: vrai si aucune ligne n’a été traité par la requête

%ISOPEN : vrai si le curseur est ouvert.

* Les curseurs implicites sont tous nommés SQL

Exemple de cursuer implicite

DECLARE

NbLigne integer;

BEGIN

Delete from emp Where dept = 10;

Nbligne := SQL%ROWCOUNT;

Curseur explicite

* Pour traiter les Select qui retournent plusieurs lignes
* Doivent être déclarés
* Le code doit utiliser les ordres OPEN, FETCH, CLOSE
* On les utilise dans une boucle dont on sort quand NOTFOUND est à vrai.

Exemple de déclaration :

DECLARE

CURSOR salaire IS Select sal from emp wher dept =10;

Salaire Numeric(8,2);

Total Numeric(10,2):=0;

BEGIN

Open Salaires;

LOOP

fetch Salaire into Salaire;

Exit WHEN Salaires %NOTFOUND;

If Salaire is Not Null then

Total := total + Salaire;

DBMS\_OUTPUT put line(total);

Endif;

EndLoop;

Close Salaires;

DBMS OUTPUT.PUTLINE(total)

END

TYPE “ROW” associé à un curseur

Exemple

DECLARE

Cursor c is Select matr,nome,sal from emp;

Employe C%ROWTYPE;

BEGIN

Open c

Fetch c into employe

…

Boucle FOR et curseur

* Elle simplifie la programmation car elle evite open fetch close
* En plus elle déclare implicitement une variable de type ROW associé au curseur

Exemple :

DECLARE

Cursor c is Select Matr, Nome from emp where dept = 10;

BEGIN

FOR employe in c LOOP

dbms output.putline(employe.Matr,employe.nome);

ENDLOOP

END

Curseurs paramétrés

* Un curseur paramétré peut servir plusieurs fois avec des valeurs de paramètres différentes
* On doit fermer le curseur entre chaque utilisation

Ex :

DECLARE

Cursor c (p\_dept integer) is

Select dept, nome from emp where dept=p\_dept;

BEGIN

for employe in c(10) LOOP

…

End;

For employe in c(20) LOOP

…

End;

END;

Ligne courante d’un curseur

* La ligne courante d’un curseur est déplacée à chaque appel de fetch
* On est parfois amené à modifier la ligne courant pendant le parcours du curseur
* Pour cela on utilise la clause “where current of” pour désigner cette ligne courante dans un ordre LMD(insert, update, delete)
* Il est nécessaire d’avoir déclaré le curseur avec la clause FOR UPDATE pour que le bloc compile

FOR UPDATE [OF cd1,… cdn]

* Cette clause bloque toute la ligne ou les colonnes précisées
* Les autres transactions ne pourront modifier les valeurs que quand le curseur sera passé à une autre ligne.

Ex :

DECLARE

Cursor c is Select matr,nome,sal from emp where dept=10 FOR UPDATE OF emp sal;

…

If salaire is not null then

Total :=total=salaire;

Else --met 0 à la place de NULL

Update emp set sal=0 where current of c;

End if ;