SYNTHESE PL SQL – ORACLE

Bloc PL-SQL

```
DECLARE
    Declaration des types, des variables, des constantes, des
    exceptions et des curseurs.
BEGIN [nom du bloc]
    Instructions
EXCEPTION
    Traitement des erreurs
END [nom du bloc]
```

Déclaration des variables, des constants, des exceptions et des curseurs.

Le bloc peut être exécuté directement sous SQL-PLUS

Commentaires

/* … */

Variables

Déclaration des variables

NomVar Type ;

Type des variables

Char(n), varchar2(n), number(p,s) (s : nombre de décimales), date, boolean, etc.

Définition de type : les tableaux : VARRAY

```
TYPE nomType IS VARRAY[taille_max] OF type_element ;
```

Définition de type : les tables : TABLE

Ce sont des tableaux à une dimension de taille variable.

```
TYPE nomType IS TABLE OF type_element ;
```

Définition de type : les enregistrements : RECORD

```
TYPE nomType IS RECORD (
nomChamp typeChamps
etc
) ;
```

Utilisation des types de la base de données : %TYPE et %ROWTYPE

Avec %type, on utilise le type d'un attribut d'une table comme type de variable.

```
NomVar table.attribut%TYPE
```

Avec %rowtype, on utilise les attributs d'une table comme type record.

```
NomVar table%ROWTYPE
```

Variable globale

On peut définir une variable globale directement sous SQL-PLUS :

```
SQL> variable x NUMBER ;
```

Puis utiliser cette variable dans un bloc;

Ensuite, on peut afficher le contenu de cette variable :

```
SQL>print x
```

Affectation

Affectation classique

:=

Récupération d'une ligne d'un select : INTO

Avec var1, var2 déclarés en %type :

```
SELECT att1, att2
INTO var1, var2
FROM table
Etc.
```

Avec var declare en %rowtype:

```
SELECT *
INTO var
FROM table
Etc.
```

Opérateurs

+, -, *, /

concaténation ||

=, !=, <, <=, >, >=, and, or, not, is null, is not null, like, between, in

exponentielle **

Imbrication de blocs et visibilité des variables

On peut imbriquer un bloc dans un bloc.

```
…
Begin
…
        Declare
        …
        Begin
        …
        End
…
End
```

Les règles de visibilité des variables sont les règles classiques.

Insert into et Update avec %ROWTYPE : SET ROW

```
DECLARE
NomVar table%rowtype;
BEGIN
   NomVar.attClé := … ;
   NomVar.att2 := … ;
   Etc.
   Insert into table values nomVar;
   …
   Update table
   SET ROW = nomVar
   Where table.attClé=100;
END;
```

Retour d'un Insert into et Update : RETURNING

```
Update
Set
RETURNING att1, att2 into var1, var2 ;
```

Structures de contrôle

TEST : if, then, elsif, else, end if

```
IF expression
THEN instructions
END IF;
```

```
IF instructions
THEN instructions
ELSE instructions
END IF;
```

```
IF expression
THEN instructions
ELSIF instructions
ELSE instructions
END IF;
```

CASE : if, then, elsif, else, end if

```
CASE nomVar
WHEN valeur1 THEN instructions;
WHEN valeur2 THEN instructions;
ELSE instructions;
END CASE
```

Boucles : LOOP, FOR, WHILE

```
LOOP
   …
   exit [finDeBoucle] when condition ;
   …
END LOOP [finDeBoucle];
```

```
FOR n IN [reverse]100 .. 110 LOOP
   instructions
END LOOP;
```

```
WHILE condition LOOP
   Instructions
END LOOP;
```

Les curseurs

Présentation

Le curseur permet de récupérer le résultat complet d'un select (plusieurs colonnes et plusieurs lignes).

On peut voir ça comme un fichier en programmation type Pascal ou C.

Déclaration du curseur

CURSOR nomCurseur IS Select … ;

Utilisation d'un curseur comme un fichier

OPEN : Ouverture d'un curseur

OPEN nomCurseur ;

C'est le open qui exécute le select de la déclaration.

NomCurseur est en quelque sorte un pointeur sur la première ligne de la table.

FETCH : lecture d'une ligne et passage à la suivante

FETCH nomCurseur INTO liste de variables ;

NomCurseur pointe sur la ligne suivante.

CLOSE : Fermeture d'un curseur

CLOSE nomCurseur ;

Le close libère la mémoire.

%FOUND et %NOTFOUND : état du curseul

Avant le premier fetch : les deux valent null.

%FOUND vaut vrai après tous les fetch sauf pour le dernier.

%NOTFOUND vaut faux après tous les fetch sauf pour le dernier.

On teste : nomCurseur%FOUND ou nomCurseur%NOTFOUND

Utilisation d'un curseur comme un tableau : FOR avec curseur

On peut utiliser un curseur sans fetch :

```
FOR nomVar IN nomCurseur LOOP
       Utilisation possible de NomVar.att1
END LOOP
```

NomVar est une variable de type %ROWTYPE

Masquez le curseur avec le FOR

On peut aussi ne pas déclarer le curseur et faire :

```
FOR nomVar IN (select * FROM clients) LOOP
       Instructions
END LOOP;
```

Les exceptions

Présentation

Quand une erreur système ou applicative se produit, une exception est générée.

Les traitements en cours dans la section d'exécution (section du BEGIN) sont stoppés.

Les traitements de la section d'exception (section du EXCEPTION) commence.

Syntaxe

```
DECLARE
   déclaration de variables
BEGIN
   instructions
EXCEPTION
   WHEN nom d'exceptions THEN
       instructions
END;
```

Les 4 types d'exception

Type d'exception	Nommée	Anonyme (numérotée)
Système	1 : Exception système nommée	2 : Exception système anonyme
Utilisateur	3 : Exception utilisateur nommée	4 : Exception utilisateur anonyme

SQLCODE et SQLERRM

SQLCODE : pour afficher le numéro de l'erreur.

SQLERRM : pour afficher le numéro et le message d'erreur.

```
       dbms_output.put_line('SQLCODE = '||SQLCODE);
       dbms_output.put_line('SQLERRM = '||SQLERRM);
```

Exceptions système nommées et exceptions système anonymes (numérotées)

Exceptions prédéfinies nommées

DUP_VAL_ON_INDEX : violation de l'unicité lors d'un insert ou d'un update (-1).

NO_DATA_FOUND : un select into ne ramène aucune ligne (-1403 ; 100).

Etc.

Exceptions prédéfinies anonymes

```
dbms_output.put_line(sqlerrm(-2299));

ORA-02299: impossible de valider (.) - clés en double
   trouvées
```

WHEN exceptionPrédéfinie : exception système nommée (1)

```
EXCEPTION
   WHEN NO_DATA_FOUND THEN
```

WHEN OTHERS : exception système anonyme (2)

```
EXCEPTION
   WHEN OTHERS THEN
       dbms_output.put_line('SQLCODE = '||SQLCODE);
       dbms_output.put_line('SQLERRM = '||SQLERRM);
```

EXCEPTION_INIT : exception système anonyme nommée par l'utilisateur (2 bis)

```
DECLARE
   NOM_EXCEPTION EXCEPTION;
   PRAGMA EXCEPTION_INIT(NOM_EXCEPTION, NUMERO_EXCEPTION);
EXCEPTION
   WHEN NOM_EXCEPTION THEN
```

RAISE nomException : exception utilisateur nommée (3)

```
DECLARE
   UPDATE_SAL_EMP EXCEPTION;

EXCEPTION
   WHEN UPDATE_SAL_EMP THEN
```

RAISE_APPLICATION_ERROR : exception utilisateur anonyme (4)

```
BEGIN
   ...
   RAISE_APPLICATION_ERROR(-20000,
       'Problème d''intégrité des données') ;
   ...
EXCEPTION
   WHEN OTHERS THEN
```

Propagation des exceptions : exception et blocs imbriqués.

1) Dans un bloc, en cas d'erreur, on passe à la zone d'exception du bloc.

2) Si l'exception est prise en compte (par son nom ou par le OTHERS), les instructions correspondantes sont exécutées et on quitte le bloc : on revient donc dans le bloc appelant.

3) Si l'exception n'est pas prise en compte, on passe à la zone d'exception du bloc appelant si il existe et on revient à l'étape 2. S'il n'y a pas de bloc appelant, le programme va s'arrêter (planter !) sur cette erreur.

Procédures et fonctions

Structure d'une procédure

```
CREATE [OR REPLACE] PROCEDURE nomProcedure [(
   nomVar [{IN | OUT | IN OUT }] TYPE
   [, ...]
)]
{IS | AS}
   [nomVarLocale TYPE;]
   [...]
BEGIN
   instructions;
[EXCEPTION
   instructions d'exception]
END [nomProcedure];
```

IS ou AS sont équivalent

Structure d'une fonction

```
CREATE [OR REPLACE] FUNCTION nomFonction [(
   nomVar [{IN | OUT | IN OUT }] type
   [, ...]
)]
RETURN type
{IS | AS}
   [nomVarLocale TYPE;]
   [...]
BEGIN
   instructions;
[EXCEPTION
   instructions d'exception]
END [nomFonction];
```

IS ou AS sont équivalent

Triggers

3 types de triggers

Les triggers DML

Ils sont déclenchés avant (BEFORE) ou après (AFTER) une instruction du DML : INSERT, UPDATE ou DELETE.

Les triggers INSTEAD OF

L'exécution du trigger remplace celle de l'instruction déclenchante.

Les triggers Système

Ils sont déclenchés à l'ouverture ou la fermeture d'une BD, ou lors d'une opération du DDL (create table, drop table, etc.). Pas abordé dans ce poly.

Caractéristiques des triggers DML

syntaxe

```
CREATE [OR REPLACE] TRIGGER [nomSchema.]nomTrigger
   {BEFORE | AFTER | INSTEAD OF} instructionDeclencheur
   [REFERENCING reference]
   [FOR EACH ROW]
   [WHEN condition]
[DECLARE ...]
BEGIN
...
[EXCEPTION ...]
END [nomTrigger];
```

Moment d'exécution : BEFORE et AFTER

Niveau d'exécution : niveau tuple FOR EACH ROW et niveau table

WHEN : conditionner le déclenchement

REFERENCING : renommer NEW et OLD

Packages utilisateur

Principe

Un package permet de regrouper des fonctions et des procédures qui vont ensemble.

On peut y ajouter des variables, des types et des curseurs.

Spécifications et corps du package

Le package fonctionne avec la même logique qu'une classe.

On distingue deux parties dans un package : les spécifications et le corps .

Les spécifications : partie publique

Les spécifications du package décrivent les en-têtes des procédures et des fonctions.

On y trouve aussi les variables globales accessibles au niveau du package, ainsi que les types et les curseurs.

Le corps : partie privée

Le corps du package décrit le corps des procédures et des fonctions publiques, mais aussi des procédures et des fonctions totalement privées dont les en-têtes apparaissent à ce niveau.

Syntaxe

Création d'un package : spécifications

```
Create [or replace] package nom_package
{is | as}
   [Déclaration des variables]
   [Déclaration des variables]
   [Déclaration des en-têtes des procedures et des
   fonctions]
End [nom_package]
```

Accès aux éléments d'un package :

```
nom_package.nom_element
```

Création du corps d'un package

```
Create [or replace] package body nom_package
{is | as}
   [Déclaration des variables]
   [Déclaration des variables]
   [Déclaration des en-têtes des procédures et des
   fonctions]
Begin
   Instructions
[Exceptions
   Instructions d'exceptions
]
End [nom_package]
```

A noter qu'il peut exister des packages sans corps : ils ne concerneront donc que des variables et des types. En général, ils contiennent des exceptions.

Suppression d'un package

```
Drop package body nom_package;
```

```
Drop package nom_package;
```

Modification d'un package

On peut faire des « Alter package » mais mieux vaut faire un « create or replace » à partir d'un script sauvegardé.

Packages ORACLE

http://download.oracle.com/docs/cd/B19306_01/appdev.102/b14258/toc.htm

Il existe plus de 100 packages ORACLE.

Les packages ORACLE offrent un ensemble de fonctionnalités supplémentaires.

Il faut les exploiter pour éviter de réécrire des procédures et des fonctions qui existent déjà.

On connaît déjà le dbms_output.put_line. Le package dbms_ouput offre la fonction put_line qui permet de faire de l'affichage dans la calculette SQL.

Exemples de packages :

- DBMS_OUTPUT : pour afficher des messages
- DBMS_LOB : pour travailler avec des données binaires
- DBMS_JOB : pour soumettre des travaux à une fréquence déterminée (scheduler).
- UTL_FILE : pour manipuler des fichiers
- UTL_MAIL : pour envoyer des mails
- DBMS_SQL : pour faire du DDL avec du PLSQL, est avantageusement remplacé par EXECUTE IMMEDIATE
- DBMS_RANDOM : pour générer des nombres aléatoires
- Etc.