

# **Stockage de traitements en bases de données**

## **PL/SQL**

# PL/SQL : généralités

- En plus des données, on peut stocker du code dans une base de données par l'intermédiaire de :
  - Triggers
  - Procédures
  - Fonctions
  - Packages
- Ce code est écrit dans un langage procédural propre au SGBD. Avec Oracle, ce langage s'appelle : **PL/SQL : Procedural Language/SQL**

# PL/SQL : syntaxe générale (1/2)

- Un bloc PL/SQL est délimité par les mots-clés *BEGIN* et *END*.
- On peut découper un bloc en deux sous blocs :
  - le premier qui est la partie exécutée s'il n'y a pas d'erreur
  - le deuxième pour le traitement des erreurs.
- Ces deux blocs sont séparés par le mot-clé *EXCEPTION*.

# PL/SQL : syntaxe générale (2/2)

- La syntaxe d'un bloc PL/SQL est

DECLARE -- (mot clé uniquement utilisé pour les triggers)

déclaration des variables

BEGIN

...

...

EXCEPTION

WHEN nomErreur1 THEN

....

WHEN OTHERS THEN

...

END;

# PL/SQL : les déclarations (1/7)

- Variables de type scalaire
- Variables de type composé
  
- Les exceptions (vues plus tard)
- Les curseurs (vus plus tard)

# PL/SQL : les déclarations (2/7)

## Les types scalaires

- Type prédéfini

```
v_nom varchar2(30) ;
```

- Type en référence au type d'une colonne de table

```
v_prenom client.prenom%type;
```

# PL/SQL : les déclarations (3/7)

- Les types composés
  - les types enregistrement (record)
    - définis par le programmeur du bloc
    - en référence à un enregistrement de table
  - les types table PL/SQL
- La déclaration s'effectue en deux temps :
  - définition du type
  - déclaration de la variable

# PL/SQL : les déclarations (5/7)

- Exemple : le type enregistrement
  - Définition du type T\_identite

```
TYPE T_identite IS RECORD
(
  v_nom varchar2(30),
  v_prenom client.prenom%type
);
```
  - Déclaration de la variable v\_identite de type t\_identite :

```
v_identite T_identite ;
```
  - Ou encore (si on veut référencer toutes les colonnes de la table client)

```
v_identite client.%rowtype ;
```

# PL/SQL : les déclarations (6/7)

- Définition du type TABLE

```
TYPE nom_type IS TABLE OF  
{type_colonne | table.colonne%TYPE} [NOT NULL]  
INDEX BY BINARY_INTEGER;
```

- Déclaration de la variable

```
nom_table nom_type;
```

- Pour accéder à une ligne de la table nom\_table

```
Nom_table[2]
```

ou alors

```
Nom_table[i]
```

où i est une variable définie en binary\_integer

# PL/SQL : les déclarations (7/7)

Exemple le type table

- Définition de la variable

```
TYPE t_colonne IS TABLE OF Varchar2(10)  
INDEX BY BINARY_INTEGER;
```

- déclaration de la variable v\_colonne de type t\_colonne  
`v_colonne t_colonne ;`

La variable v\_colonne est un tableau dont les indices varient de 0 au plus grand entier codé dans le type BINARY\_INTEGER.

# PL/SQL : Assignation des variables

- Par l'intermédiaire d'un **INTO** dans un select ne renvoyant qu'une seule ligne de résultat.  
La clause **INTO** suivi du nom de la variable sont situés juste avant la clause **FROM**.
- Par l'intermédiaire d'un **:=**
- Par l'intermédiaire d'un **FETCH** lors de l'utilisation d'un curseur (vu plus tard)

# PL/SQL : traitements conditionnels

IF condition THEN

*commandes*

ELSIF condition THEN

*commandes*

ELSIF condition THEN

*commandes*

....

ELSE conditions THEN

*commandes*

END IF;

# PL/SQL : traitements itératifs (1/3)

- L'instruction LOOP

**Loop**

commandes

**End loop;**

- Pour sortir de la boucle on peut utiliser

**Exit**

**Exit when *condition***

dans le bloc de commandes.

# PL/SQL : traitements itératifs (2/3)

- L'instructions FOR - LOOP

**FOR** compteur IN borne\_inf .. borne\_sup

**Loop**

commandes

**End loop ;**

- Pour sortir avant la fin normale de la boucle on peut comme précédemment utiliser :

**Exit**

**Exit when *condition***

# PL/SQL : traitements itératifs (3/3)

- L'instructions WHILE - LOOP

**WHILE** condition **LOOP**

commandes

**End loop ;**

- Pour sortir avant la fin normale du while on peut comme précédemment utiliser :

Exit

Exit when *condition*

# Le package DBMS\_OUTPUT

- Comme on le verra plus tard, Oracle propose des ensembles de blocs PL/SQL prédéfinis nommés PACKAGE.
- Pour écrire des résultats dans la console, on utilisera le package **DBMS\_OUTPUT** et particulièrement la fonction **put\_line**.

# Le package DBMS\_OUTPUT

- Pour invoquer cette fonction, on écrira :  
`DBMS_OUTPUT.put_line(...);`
- Par défaut, les utilitaires d'Oracle ont leur console fermée. Il faut donc ouvrir cette dernière que les instruction aient un effet.
- Avec sqlplus, il faut lancer la commande  
`set serveroutput on`

# PL/SQL : les triggers (1/3)

- C'est un bloc de code PL/SQL stocké dans la base et associé à une table.
- Il est exécuté implicitement par le noyau chaque fois qu'un évènement de mise à jour du contenu de la base se produit.
- **INSERT**, **UPDATE** et **DELETE** sont les seuls événements déclencheurs.

# PL/SQL : les triggers (2/3)

```
CREATE [OR REPLACE] TRIGGER nom_trigger  
BEFORE|AFTER INSERT | UPDATE | DELETE ON nom_table  
[FOR EACH ROW ]  
    bloc pl/sql
```

- **BEFORE/AFTER** : précise le moment de déclenchement du trigger (avant ou après la mise à jour)
- **FOR EACH ROW** : précise si le trigger doit être déclenché pour chaque ligne mise à jour.  
Dans ce cas, il existe deux variables **:new** et **:old** qui contiennent respectivement le nouveau et l'ancien contenu de la ligne.

# PL/SQL : les triggers (3/3)

- Pour recompiler un trigger :
  - `ALTER TRIGGER Trigger COMPILE;`
- Un trigger peut être activé ou désactivé.
  - `ALTER TRIGGER Trigger {ENABLE|DISABLE};`
- On supprime un trigger par la commande suivante
  - `DROP TRIGGER Trigger;`

# Les séquences

- On a souvent besoin de créer automatiquement des clés entières.
- Une séquence est un entier stocké dans la base de données qui peut être consulté et incrémenté.
- Oracle en suivant la norme SQL a implémenté le concept de séquence.

# Les séquences

- On crée une séquence comme suit  
`CREATE SEQUENCE ma_sequence`  
`[INCREMENT BY 1]`  
`[START WITH 1100]`  
`[MAXVALUE 99999];`
- Par défaut, la séquence est initialisée à 1 et s'incrémente de 1.

# Les séquences

- On accède à la valeur d'une séquence comme suit :

```
SELECT ma_sequence.currval FROM dual;
```

- On accède et incrémente la valeur d'une séquence comme suit :

```
SELECT ma_sequence.nextval FROM dual;
```

ou directement dans une requête de MAJ

```
insert into XXX
```

```
values(ma_sequence.nextval,...);
```