EMSI RABAT – Ingénierie Informatique & Réseaux

EMSI 3 – 3°IIR 3 – INFORMATIQUE

Bases de Données ORACLE & PL/SQL

*===> Études Avancée en la Programmation PL/SQL

Professeur : Dr. M. NASSAR
Mr. A. ATTAR

Étudiants : Youssef MAHTAT
Ibrahim MANNANE

Etudes Avancée
en
Les Bases de Données
ORACLE®
&
PL/SQL
–
Étude de
la Programmation
en
PL/SQL

Notes de Cours
en
Informatique :

Mini CHEAT-SHEET
en

Bases de Données

ORACLE®

&

Programmation en PL/SQL

(Selon le cours des Professeurs Dr. NASSAR & Mr. ATTAR)

Developer.

→ Partie 1 « SQL » :

* * → Langage d'interrogation des données :

➢ Syntaxe SELECT :

```
SELECT columns [, group_fonctions, ...] FROM tables
[ WHERE condition ]
[ GROUP BYgroup_by_expression
[ HAVING group_condition ] ]
[ ORDER BY column [ASC| DESC] [, …] ];
```

- Avec :

Les fonctions de groupes	
Fonction	**Description**
Count(*|[DISTINCT|ALL] expr)	Le nombre de ligne de expr
Avg([DISTINCT | ALL] expr)	Valeur moyenne de expr, en ignorant les valeurs NULL
Min([DISTINCT | ALL] expr)	Valeur minimale de expr, en ignorant les valeurs NULL
Max([DISTINCT | ALL] expr)	Valeur maximale de expr, en ignorant les valeurs NULL
Sum([DISTINCT | ALL] expr)	Somme des valeurs de expr, en ignorant les valeurs NULL

➢ Equijointure :

```
SELECT expr
  FROM table1 T1, table2 T2
  WHERE T1.column_in_T1 = T2.column_in_T2;
```

➢ Auto-jointure :

```
SELECT expr
FROM table1 Alias1, table1 Alias2
WHERE Alias1.col1= Alias2.col1 ;
```

➢ Non Equijointure (thêta jointure) :

Une non-équijointure est une condition de jointure contenant un opérateur qui n'est pas un opérateur d'égalité. Par exemple :
La liste des employés et leurs grades :

```
SELECT  EMP.NOM, SAL.GRA
FROM  EMP, SAL
WHERE  EMP.SAL BETWEEN SAL.SALMIN and SAL.SALMAX ;
```

➢ Syntaxe SELECT imbriqué :

```
SELECT colonnes_de_projection
FROM tables
WHERE [conditions and] expr operator (
                                        SELECT colonnes_de_projection
                                        FROM table WHERE conditions
                                         ....
                                    );
```

- Avec operator :

Type de sous requête	opérateur
ramène une seule ligne	=, >, >=, >, <=, <>
ramène plusieurs lignes	IN : appartenance ALL : à tous ANY: au moins un EXISTS : non vide
Plusieurs lignes avec plusieurs colonnes.	EXISTS : nonvide

- Tels Que :
  - IN : la condition est vraie si l'élément appartient au résultat retourné par la sous-requête ;
  - ANY : la condition est vraie si la comparaison est vérifiée pour au moins un élément du résultat retourné par la sous-requête ;
  - ALL : la condition est vraie si la comparaison est vérifiée pour tous les éléments du résultat retourné par la sous-requête ;
  - EXISTS (sous-requête) - Vraie si Résultat(sous-requête) != Ensemble vide, Faux dans le cas contraire ;

- Et avec les opérateurs ensemblistes :
  - INTERSECT
  - UNION
  - UNION ALL
  - MINUS

* * → Langage de Manipulation des Données :

➢ Syntaxe INSERT :

```
INSERT INTO <nom table> [( colonne1 [, colonne2] … )]
    VALUES (<valeur1> [, <valeur2>] … ) | <requête> ;
```

➢ Syntaxe UPDATE :

```
UPDATE <nom table>
SET <colonne> = valeur [, <colonne> = valeur ] …
[WHERE <condition de modification> ];
```

➢ Syntaxe DELETE :

```
DELETE FROM <nom table> [WHERE <condition>] ;
-- ou Bien :
DELETE FROM <nom table>
WHERE expr IN (sous-requet);
```

* * → Langage de Définition des Données :

➢ Syntaxe Création Tables :

```
CREATE TABLE nomTable
(
   colonne type  [contrainte de la colonne]
   [, colonne type  [contrainte de la colonne], ...]
   ......
   [, contraintes de la table]
    .....
    …
);
```

➢ Syntaxe de la commande ALTER TABLE :

```
 ALTER TABLE <nom de la Table>
       | ADD COLUMN <def Colonne>
       | DROP COLUMN <nom Colonne> [RESTRICT|CASCADE]
       | ADD CONSTRAINT <def Contrainte>
       | DROP <nom Contrainte> [RESTRICT|CASCADE] |
 ;
```

// c'est-à-dire avec ALTER Soit ADD, DROP, ou ADD CONSTRAINT ;

➢ Syntaxe de la commande DROP TABLE :

```
DROP TABLE <Nom de la table> ;
```

* * → Les Vues :

➢ Syntaxe de CREATE VIEW :

```
CREATE VIEW <nom vue> [(liste des attributs)]
AS <requête de sélection>
[WITH CHECK OPTION] ;
```

- Avec " WITH CHECK OPTION" :
  Permet de vérifier que les mises à jour ou les insertions faites à travers la vue ne produisent que des lignes qui feront partie de la sélection de la vue.

→ Partie 2 « Intro PL/SQL » :

➢ Syntaxe et STRUCTURE D'UN BLOC PL/SQL :

```
[DECLARE] -- section optionnelle
 -- déclaration variables, constantes, types, curseurs,...
 BEGIN -- section obligatoire
       contient le code PL/SQL

       [EXCEPTION ] --section optionnelle
              traitement des erreurs

END; -- obligatoire
```

➢ Les Variables : `nom_variable [CONSTANT] type [[NOT NULL]  [:= expression | DEFAULT expression] ;`

➢ Sous Types : `SUBTYPE nom_newType IS TYPE OPTIONS ;`

➢ Extraction de type :

- Déclaration d'un champ ou variable avec le type d'un champ existant : `nomVariable NomTable.NomChamp%TYPE;`
- Déclaration d'un champ ou variable comme étant une occurrence (ligne ou enregistrement) d'une table existante :

```
nomVariable NomTable.NomChamp%ROWTYPE;
```

➢ Enregistrement :

- Création de type d'enregistrement :

```
TYPE nom_type_rec  IS RECORD
(
      nom_champ1 type_élément1 [[ NOT NULL] := expression ],
      nom_champ2 type_élément2 [[ NOT NULL] := expression ],
      .... …
      nom_champN type_élémentN[[ NOT NULL] := expression ]
) ;
```

- Déclaration d'une variable du type d'enregistrement :

```
Nom_variable nom_type_rec ;
```

➢ Assignation ou Affectation en PL/SQL :

```
NomVariable := EXPRESSION | valeur ;
```

➢ Récupération du Clavier en PL/SQL :

Lors de la déclaration de variable : `NOM_VARIABLE TYPE := &NomSymbolique ;`

- Directive pour Ne plus afficher les anciens et nouvelles valeurs :

```
SET VERIFY OFF; --pour ne pas afficher les anciennes valeurs & ...
```

- Directive pour Activer l'affichage :

```
SET SERVEROUTPUT ON;
```

- Fonction d'affichage standard :

```
DBMS_OUTPUT.PUT_LINE ('chaine à afficher' | ...) ;
```

- Récupérer un enregistrement d'un SELECT dans des variables :

```
SELECT list_columns INTO list_variables from ..... [....];
```

- Structure IF ... ELIF ... ELSE :

```
IF condition1 THEN
      instruction1;
      instruction 2;
      ...
ELSIF condition2 THEN
      instruction 3;
      instruction 4;
       ...
ELSIF condition3 THEN
       instruction 5;
       instruction 6;
       ...
ELSE
      instruction 7;
      ...
END IF;
```

- Structure LOOP :

```
<<label>>
LOOP
     instruction1 ;
     instruction2 ;
     ...
EXIT [label][WHEN condition1]
END LOOP label;
```

- Structure FOR :

```
FOR nom_compteur IN [REVERSE] borne_inf..borne_sup LOOP
      instruction1 ;
      instruction2 ;
      instruction3 ;
      ...
[EXIT WHEN condition];
END LOOP;
```

➢ Structure CASE :

```
CASE nom_selecteur
     WHEN expression1 THEN
            instruction 1 ;
            ...
     WHEN expression2 THEN
            instruction 2 ;
            ...
     …
     ...
     WHEN expressionN THEN
            instructionN ;
            ...
     ELSE
            Autres instruction ;
            ...
END CASE;
```

→ Partie 3 « Transactions & Curseurs en PL/SQL » :

➢ Syntaxes en TRANSACTIONs :

- `SET  TRANSACTION READ [ONLY | WRITE] -- implicitement WRITE`
- `commit ; -- valider les transactions`
- `SAVEPOINT <NOM_POINTSAUVEGARDE> ;`
- `ROLLBACK ; --  ANNULE entièrement les transactions`

➢ Curseurs Implicites :

- Nombre de ligne d'une requête : **SQL%ROWCOUNT**
- Boucle sur un SELECT (Parcourir les occurrences de la SELECT) :

```
FOR NomCurseur IN (SELECT list_columns FROM tables [...]) LOOP
    Code_Traitement
    (Avec NomCurseur représente l'occurrence récupérer)
END LOOP;
```

➢ Curseurs Explicites :

- Création du curseur : `CURSOR nom_curseur [(parametres) ]  IS requête_select;`
- Ouverture du curseur : `OPEN nom_curseur ;`
- Extraction d'une occurrence (récupéré dans des variable ou enregistrement) :
  `FETCH nom_curseur INTO listes_variables|nom_Record ;`
- Fermeture du curseur : `CLOSE nom_curseur ;`
- Savoir si le curseur est ouvert : `Nom_curseur%ISOPEN`
- Savoir si le curseur n'a pas retourné de ligne : `Nom_curseur%NOTFOUND`
- Savoir si le curseur a retourné de ligne : `Nom_curseur%FOUND`
- Le nombre total de lignes traités jusqu'à maintenant : `Nom_curseur%ROWCOUNT`
- Curseur pour MàJ :
  `CURSOR nom_curseur [(parametres) ] [RETURN ROWTYPE] IS requête_select FOR UPDATE;`
- La mise à jour avec le curseur :
  `UPDATE nom_Table SET champ1=valeur 1 [, ...] WHERE CURRENT OF nom_curseur;`

→ Partie 4 « EXEPTIONS & PROCEDURES & FONCTIONS » :

* * → Les EXCEPTIONs :

➢ Syntaxes EXCEPTION :

```
EXCEPTION
    WHEN exception1 [OR exception2…] THEN
          instruction1;
          ….…
    WHEN exception3 [OR exception4…] THEN
          instruction2;
          …
    ....
    [WHEN OTHERS THEN
           instruction1;
           instruction2;
           …]
```

➢ Exemple d'exceptions prédéfinies :

Code d'erreur SQLCODE	Erreur
+1403	NO_DATA_FOUND
-1	DUP_VAL_ON_INDEX
-6502	VALUE_ERROR
-1001	INVALID CURSOR
-1722	INVALID NUMBER
-6501	PROGRAM ERROR
-1017	LOGIN DENIED
-1422	TOO_MANY_ROWS
-1476	ZERO_DIVIDE

- ➢ Syntaxe déclaration de variable exception : `Nom_Exception exception ;`
- ➢ Syntaxe initiation d'une variable exception par son code d'erreur :
  `PRAGMA EXCEPTION_INIT (Nom_Exception, -valeurCode);`
  (valeurCode correspond à la valeur du code à gérer si on connait pas son nom, ou bien utiliser un code pour une erreur qui n'existe pas et qu'on veut créer)
- ➢ Récupérer la valeur du code d'une erreur déclenchée : SQLCODE
- ➢ Récupérer le message associé à une erreur déclenchée : SQLERRM
- ➢ Lever une exception : `RAISE Nom_exception ;`

* * → Les PROCEDURES :

- Ajouter le droit de création de procédures à un schéma d'utilisateur :

```
GRANT CREATE PROCEDURE  TO Nom_USER;
```

- Ajouter le droit de création de n'importe quel procédures à un schéma d'utilisateur :

```
GRANT CREATE ANY PROCEDURE  TO Nom_USER;
```

- Autoriser un autre schéma à exécuter une procédure :

```
GRANT EXECUTE ON NOM_PROCEDURE TO AUTRE_USER;
```

- Syntaxe procédure :

```
CREATE  [OR REPLACE] PROCEDURE NOM_PROC  [(PARAMETRES)] [AUTHID [CURRENT_USER | DEFINER]]
[DECLARATION de Variables]
[IS | AS]
BEGIN
      BLOC/PLSQL
END [NOM_PROC] ;
```

- Appel à une procédure dans programme ou BLOC PL/SQL :

```
Nom_PROCEDURE(liste_paramètres) ;
```

- Appel à une procédure dans SQL*PLUS :

```
EXECUTE Nom_PROCEDURE(liste_paramètres) ;
Ou bien : EXEC Nom_PROCEDURE(liste_paramètres) ;
```

- Recompilation de Procédure :

```
ALTER PROCEDURE nom_procédure COMPILE;
```

- Suppression de Procédure :

```
DROP PROCEDURE nom_procédure ;
```

* * → Les FONCTIONS :

- Syntaxe Fonctions :

```
CREATE [OR REPLACE] FUNCTION NOM_FCT [(PARAMETRES)] RETURN TYPE_RETOUR_FCT
      [AUTHID [CURRENT USER | DEFINIR]]
[Declaration de variables]
[IS | AS]
BEGIN
      BLOC PL/SQL
      [EXCEPTIONS]
      ...
      RETURN NomValeurRETOUR;
END [NOM_FCT];
```

- Appel à une procédure dans programme ou BLOC PL/SQL :

```
Nom_Variable := Nom_FONCTION(liste_paramètres) ;
```

- Appel à une procédure dans SQL*PLUS :

```
EXECUTE :Nom_Variable := Nom_FONCTION(liste_paramètres) ;
Ou bien : EXEC :Nom_Variable := Nom_FONCTION(liste_paramètres) ;
```

- Recompilation de Fonction :

```
ALTER FUNCTION nom_fonction COMPILE;
```

- Suppression de Fonction :

```
DROP FUNCTION nom_fonction ;
```

* * → Les PACKAGES :

➢ Syntaxe PACKAGE :

```
-- PROTOTYPE (spécification) :
CREATE PACKAGE nom_package
AS
      PROCEDURE P1 (….);
      FUNCTION F1(…) RETURN …..;
      VARIABLES
      EXCEPTIONS
      ....
END [nom_package];

-- BODY(Déclaration du PACKAGE) :

CREATE PACKAGE BODY nom_package
AS
      PROCEDURE P1 (….) IS
      BEGIN ……
      END P1;
      ....
      ……
END [nom_package];
```

➢ Compilation PACKAGE :

```
START nom_fichier_contenant_le_package
```

➢ Recompilation PACKAGE :

```
ALTER PACKAGE nom_package COMPILE BODY;
ALTER PACKAGE nom_package COMPILE PACKAGE;
```

➢ Destruction PACKAGE :

```
DROP PACKAGE BODY nom_package;
DROP PACKAGE nom_package;
```

→ Partie 5 « TRIGGERS » :

➢ Syntaxe TRIGGERS :

```
CREATE [OR REPLACE] TRIGGER nom_trigger
[BEFORE|AFTER] INSERT|DELETE|UPDATE ON nom_table
[FOR EACH ROW]
[FOR EACH ROW WHEN (NEW.NomChamp IS NULL)]
BEGIN
     Corps_de_trigger
END;
```

➢ Condition sur les évènements dans les TRIGGERS :

```
INSERT IF INSERTING THEN ….
DELETE IF DELETING THEN …
UPDATE IF UPDATING('ATTRIBUT') THEN …    (modification de l'attribut)
       IF UPDATING THEN … (n'importe quelle modification sur la table)
```