PLSQL : Cours 1 : Procédure, Fonctions, Triggers

Contents

1 PLSQL : Cours 1 : Procédure, Fonctions, Triggers **2**

1.1 Bloc PLSQL . . . 2
1.2 Déclarer une variable . . . 2
1.3 Assigner des valeurs . . . 2
1.4 Types de données . . . 2
1.5 Interagir avec la base de donnée . . . 2
1.5.1 Récupérer des données . . . 2
1.5.2 Manipuler des données . . . 3
1.5.3 Gestion des Transactions . . . 3
1.6 Structures de contrôle . . . 3
1.7 Boucles . . . 4
1.7.1 Boucle simple (équivalent do while) . . . 4
1.7.2 Boucle FOR . . . 4
1.7.3 Boucle WHILE . . . 4
1.8 Curseur . . . 4
1.8.1 Déclarer un curseur . . . 4
1.8.2 Opérations sur un curseur . . . 4
1.8.3 Attributs d'un curseur . . . 4
1.8.4 Curseur avec boucle FOR . . . 4
1.8.5 Curseur particulier : SQL . . . 4
1.9 Exceptions . . . 5
1.9.1 Syntaxe . . . 5
1.9.2 Exception définies par l'utilisateur . . . 5
1.10 Procédures . . . 5
1.10.1 Mode de passage de paramètres . . . 5
1.11 Fonctions . . . 6
1.12 Packages . . . 6
1.12.1 Entête . . . 6
1.12.2 Corps . . . 6
1.12.3 Exemple . . . 6
1.13 Triggers . . . 6
1.13.1 Syntaxe . . . 6
1.13.2 Timing . . . 7
1.13.3 Evènement . . . 7
1.13.4 Type . . . 7
1.13.5 Prédicats conditionnel . . . 7
1.13.6 Activation . . . 7

1 PLSQL : Cours 1 : Procédure, Fonctions, Triggers

1.1 Bloc PLSQL

```
DECLARE
    ...
BEGIN
    ...
EXCEPTION
    ...
END;
```

1.2 Déclarer une variable

```
DECLARE
    identifier [CONSTANT] datatype [NOT NULL] [:= | DEFAULT initvalueexpr]
```

Exemples :

```
DECLARE
    hiredate DATE;
    deptno NUMBER(2) NOT NULL := 10;
    location VARCHAR(13) := 'Paris';
    comm CONSTANT NUMBER := 1400
```

1.3 Assigner des valeurs

```
identifier := expr;

hiredate := '2020-01-01'
```

1.4 Types de données

- VARCHAR2(maxlength)
- NUMBER[(presicion,scale)]
- DATE
- CHAR[(maxlength)]
- LONG
- LONG RAW
- BOOLEAN
- BINARY_INTEGER

On peut également affecter dynamiquement le type d'une autre variable ou d'une colonne de la base avec l'attributs `%TYPE` :

```
DECLARE
name EMP.NAME%Type;
```

On peut également récupérer un type Tuple depuis une table :

```
DECLARE
dept_record DEPT.%ROWTYPE;
```

Le fonctions SQL (sauf fonctions d'agrégation sont valides en PLSQL)

1.5 Interagir avec la base de donnée

1.5.1 Récupérer des données

```
DECLARE
    v_deptno NUMBER(2);
    v_loc   VARCHAR(15);
BEGIN
    SELECT deptno, loc INTO v_deptno,v_loc
```

```
    FROM dept
    WHERE dname = 'SALES':
```

```
DECLARE
    v_sum_salary EMP.SAL%TYPE;
    v_deptno NUMBER NOT NULL := 10;
BEGIN
    SELECT SUM(sal) INTO v_sum_salary
    FROM EMP
    WHERE deptno = v_deptno
```

1.5.2 Manipuler des données

On peut `INSERT` / `UPDATE` / `DELETE` des données :

```
DECLARE
    v_empno EMP.EMPBO%TYPE
BEGIN
    SELECT empno_sequence.NEXTVAL
    INTO v_empno
    FROM DUAL;

    INSERT INTO EMP(EMPNO,ENAME,JOB,DEPTNO)
    VALUES (v_empno),'HARDING','CLERK',10);
END;
```

```
DECLARE
    v_sal_increase EMP.SAL%TYPE := 2000;
BEGIN
    UPDATE EMP
    SET SAK = SAL + v_sal_increase
    WHERE JOB = 'ANALYST';
```

```
DECLARE
    v_deptno EMP.DEPTNO%TYPE := 10
BEGIN
    DELETE FROM EMP
    WHERE DEPTNO = v_deptno
```

1.5.3 Gestion des Transactions

On peut créer un point de sauvegarde :

```
SAVEPOINT a;
```

Pour y revenir plus tard :

```
ROLLBACK TO SAVEPOINT a;
```

Le Rollback n'est pas un "GOTO", il ne contrôle pas le flux d'exécution du code PLSQL, il va simplement annuler les opérations DML depuis le point de sauvegarde.

On peut également valider la transaction :

```
COMMIT;
```

1.6 Structures de contrôle

On peut faire un If :

```
IF ...condition... THEN
    ...statements...
[ELSIF ...condition... THEN
    ...statements...]
[ELSE ...statements...]
END IF;
```

1.7 Boucles

1.7.1 Boucle simple (équivalent do while)

```
LOOP
    ...statement...
    EXIT [WHEN condition]
END LOOP;
```

1.7.2 Boucle FOR

```
FOR ...index variable... IN [REVERSE] ...start... .. ...end... LOOP
    ...statements...
END LOOP;
```

1.7.3 Boucle WHILE

```
WHILE ...condition... LOOP
    ...statements...
END LOOP;
```

1.8 Curseur

Structure de donnée permettant d'itérer sur les résultats d'une requête SQL.

1.8.1 Déclarer un curseur

```
CURSOR employees IS
SELECT * FROM EMP;
```

1.8.2 Opérations sur un curseur

- Open : ouvre le curseur

```
OPEN employees;
```

- Fetch : récupère le prochain élément du curseur dans une variable

```
FETCH employees INTO employee
```

- Close : ferme le curseur :

```
CLOSE employees;
```

1.8.3 Attributs d'un curseur

- `%ISOPEN` : `TRUE` si le curseur est ouvert
- `%NOTFOUND` : `TRUE` si le dernier `FETCH` n'a pas retourné d'élément
- `%FOUND` : `TRUE` si le dernier `FETCH` a retourné d'élément
- `%ROWCOUNT` : Nombre d'éléments récupérer jusque là

1.8.4 Curseur avec boucle FOR

On peut itérer sur un curseur directement avec une boucle FOR, à la manière d'un FOR EACH.

```
FOR employee IN employees LOOP
    ...statments...
END LOOP;
```

1.8.5 Curseur particulier : SQL

Le curseur SQL permet de récupérer des informations sur la dernière instruction DML exécutée :

- `SQL%ROWCOUNT` : Nombre de ligne affectées
- `SQL%FOUND` : `TRUE` si au moins une ligne affectée

- SQL%NOTFOUND : TRUE si aucune ligne affectée
- SQL%ISOPEN : toujours FALSE

1.9 Exceptions

On peut utiliser une clause EXCEPTION dans un bloc PLSQL pour gérer les exceptions levées.

```
DECLARE
    ...
BEGIN
    ...
EXCEPTION
    ...
END;
```

1.9.1 Syntaxe

```
EXCEPTION
    WHEN ...exception1... [OR ...exception2...] THEN
        ...statements...
    [WHEN ...exception1... [OR ...exception2...] THEN
        ...statements...]
```

Exemple :

```
DECLARE
    v_name EMP.NAME%TYPE;
BEGIN
    SELECT NAME INTO v_empno
    FROM EMP
    WHERE EMPNO = 10 ;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('No employee with this Id');
END;
```

Pour récupérer respectivement le code de l'erreur et son message, utiliser les macros SQLCODE et SQLREM.

1.9.2 Exception définies par l'utilisateur

On peut aussi déclarer nos propres exception, et leur assigner un code d'erreur (entre -20 000 et -20 999) :

```
DECLARE
    e_no_such_emp EXCEPTION;
    PRAGMA EXCEPTION_INIT(e_no_such_emp, -22292)
```

Pour lever une exception :

```
RAISE e_no_such_emp;
```

1.10 Procédures

Une procédure est une bloc PLSQL réutilisable qui effectue une action.

```
CREATE [OR REPLACE] PROCEDURE ...nom... (...arg1... [...mode...] ...type, ...)
IS
... bloc plsql...
END ...nom...;
```

Supprimer une procédure : DROP PROCEDURE ...nom...

1.10.1 Mode de passage de paramètres

- IN : Données passée au sous programme à laquelle il accèdera en lecture uniquement
- OUT : Utilisé pour retourner une valeur au programme appelant par effet de bord
- IN OUT : Données passée au sous programme auquel il accède en lecture et écriture qui est retournée au programme parent.

1.11 Fonctions

Une fonction est une procédure qui retourne une valeur.

```
CREATE [OR REPLACE] FUNCTION ...nom... (...arg1... [...mode...] ...type..., ...)
RETURN ...type...
IS
... bloc plsql...
END ...nom...;
```

1.12 Packages

Permet de lier les éléments qui ont un rapport logique ou sémantique entre eux. Un package est constitué d'un entête et d'un corp :

1.12.1 Entête

Pour les prototype des éléments:

```
CREATE [OR REPLACE] PACKAGE ...nom...
IS
    ...déclaration des prototypes...
END ...name...;
```

1.12.2 Corps

Pour les implémentations :

```
CREATE [OR REPLACE] PACKAGE BODY ...nom...
IS
    ...déclaration des implémentations...
END ...name...;
```

1.12.3 Exemple

```
CREATE OR REAPLACE PACKAGE time IS
    FUNCTION GetTimestamp RETURN DATE;
    PROCEDURE resetTimeStamp;
END time;

CREATE OR REPLACE PACKAGE BODY time IS
        StartTimesStamp DATE := SYSDATE; -- donnée privée du package

        FUNCTION GetTimeStamp RETURN DATE IS
        BEGIN
            RETURN StartTimesStamp;
        END GetTimeStamp;

        PROCEDURE ResetTimeStamp IS
        BEGIN
            StartTimestamp := SYSDATE;
        END ResetTimeStamp;
END time;
```

1.13 Triggers

Un trigger est un bloc PLSQL qui s'exécute automatiquement sur un événement particulier, les interactions avec la base de donnée.

1.13.1 Syntaxe

```
CREATE [OR REPLACE] TRIGGER ...nom...
...timing... ...event... [OR ...event2..., ]
ON ...table...
```

```
[FOR EACH ROW]
[WHEN ...condition...]
...bloc PLSQL...
```

1.13.2 Timing

- `BEFORE` : le trigger s'exécute avant le déclanchement de l'événement.
- `AFTER` : le trigger s'exécute après le déclanchement de l'événement.

1.13.3 Evènement

L'événement en question peut être une des trois opérations DML : `INSERT`, `UPDATE` ou `DELETE`.

1.13.4 Type

Il existe deux types de trigger :

- Statement : s'exécute une seule fois pour un événement (défaut) pas d'accès au contenu de chaque ligne
- Row : s'exécute une fois pour chaque ligne affectée par l'événement. Possibilité d'accéder aux valeurs avec `:OLD` et `:NEW`.

1.13.5 Prédicats conditionnel

Pour les triggers qui se déclenchent sur plusieurs événements, on peut vérifier de quel événement il s'agit dans une condition :

```
IF INSERTING THEN
    ... statements...
END IF;
```

Le prédicat peut également porter sur une colonne précise de la table :

```
IF UPDATING('SAL') THEN
    ... statements...
END IF;
```

1.13.6 Activation

```
ALTER TRIGGER ...nom trigger... DISABLE | ENABLE
```

Pour tous les trigger d'une table :

```
ALTER TRIGGER ...nom table... DISABLE | ENABLE ALL TRIGGERS
```