

# Chapitre 8 : Initiation à Oracle © PL/SQL

## INF3080 BASES DE DONNÉES (SGBD)

Guy Francoeur

Aucune reproduction sans autorisation

3 septembre 2019

**UQÀM** | **Département d'informatique**

- ▶ Les droits de lecture sont accordés aux étudiants inscrits au cours INF3080-030 A2019 uniquement;
- ▶ Aucun droit pédagogique ou reproduction n'est accordé sans autorisation;

1. PL/SQL
2. Appel et exécution
3. Gestion des erreurs

# Table des matières

## 1. PL/SQL

initiation

Procédure

Fonction

Gachette

## 2. Appel et exécution

## 3. Gestion des erreurs

## Procedural Language (extension) for SQL

- ▶ Le PL/SQL existe dans Oracle depuis la version 6;
- ▶ Les objets : *procedures, functions, packages, triggers* en v7;
- ▶ La syntaxe ou la forme peut varier entre les objets pl/sql;
- ▶ Le bloc anonyme est une procédure PL/SQL sans nom;
- ▶ Les variables sont initialisées avec NULL;
- ▶ Les procédures et fonctions acceptent des paramètres
- ▶ Il existe un mécanisme pour la gestion des erreurs.

Le PL/SQL Oracle suit une forme générique qui inclut les termes suivants. Certains sont facultatifs d'autres essentiels :

- ▶ DECLARE
- ▶ BEGIN
- ▶ EXCEPTION
- ▶ END

```
SQL> SET SERVEROUTPUT ON; -- attention
BEGIN
  DBMS_OUTPUT.put_line ('Hello World!');
END;
/
```

# Procedure - formalisme

**traduction** Procedure  $\rightarrow$  Procédure;

**définition** La procédure est un programme (bloc de code) qui de façon générale ne retourne pas de valeur. Nous allons voir la fonction.

```
SQL> CREATE [ OR REPLACE ] PROCEDURE [schéma.]<nom_objet> (  
    [ <nom_variable> [IN | OUT | IN OUT] TYPE INTERNE [,] ]  
)  
AS  
-- Section Déclaration (facultatif)  
-- il n'y a pas de DECLARE dans la procédure  
error_message VARCHAR2(30) := 'Une erreur est survenue.';  
courriel Connexion.cCourriel%type;  
  
BEGIN -- Section Exécutable (obligatoire)  
    courriel := icPrenom || '.' || icNom || '@' || icCompagnie;  
  
EXCEPTION -- Gestion des exceptions (facultatif)  
    WHEN VALUE_ERROR THEN  
        DBMS_OUTPUT.PUT_LINE(error_message);  
END <nom>;  
/
```

# Variables - déclaration, affectation

Oracle nous offre une fonctionnalité très intéressante afin de pouvoir déclarer des variables sans savoir ou les associer à des colonnes existantes;

```
SQL>
DECLARE
  variable TABLE_NAME.COLUMN_NAME%type;
  --Exemple
  variable Langue.cLangue%type;

  variable := 'une langue';
  ...
```



# bloc anonyme - procedure

```
SQL> -- PL/SQL bloc anonyme
DECLARE
-- Cette section est facultative
  nNum1 NUMBER(2);
  nNum2 nNum1%TYPE := 17; -- valeur default
  cVal VARCHAR2(12) := 'Hello world';
  dDate DATE := SYSDATE; -- Courante

BEGIN
-- section obligation avec au moins une instruction
  SELECT 'NOUVEAU' INTO cVal FROM DUAL;

EXCEPTION
-- Cette section est facultative
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Err est ' || TO_CHAR(sqlcode)||
      sqlerrm);
END;
/
```

Le bloc est un bout de code qui peut être inclus dans un autre bloc.

# Fonction - formalisme

traduction Function  $\rightarrow$  Fonction;

définition La fonction est un programme qui retourne un résultat.

```
SQL> CREATE [ OR REPLACE ] FUNCTION [schéma.]<nom_objet> (  
[ <nom_variable> [IN | OUT | IN OUT] [NOCOPY] <TYPE_INTERNE> [,]  
    ]  
)  
< RETURN TYPE_INTERNE >  
< AS | IS >  
pl/sql block;
```

# Fonctions - exemple

```
SQL> CREATE OR REPLACE
FUNCTION ma_fonction ( icLike IN VARCHAR2 )
RETURN NUMBER
IS
    nVal NUMBER;
BEGIN
    SELECT COUNT(*) INTO nVal FROM Langue
    WHERE cLangue LIKE '%' || icLike || '%';
    RETURN(nVal);
END;
/
```

# Gachette - formalisme

**traduction** Trigger  $\rightarrow$  Gachette;

**définition** La gachette est un bloc de code (on parlera de procédure) qui est déclenché sur des événements pré-déterminés.

Événements :

► AFTER ou BEFORE

► INSERT ou DELETE ou UPDATE

```
SQL> CREATE OR REPLACE TRIGGER <nom>
< BEFORE | AFTER | INSTEAD OF >
< INSERT | DELETE | UPDATE [OF COLUMN <nom> ]>
< ON [schéma.]<nom de TABLE> >
[ FOR EACH ROW ]
[ WHEN ]
[ REFERENCING OLD [as vieux] | NEW [as new] | PARENT [as parent]
  ]
```

# Gachette - exemple

```
CREATE TRIGGER schema.tbidu_nom_gachette
BEFORE
DELETE OR INSERT OR UPDATE
ON schema.nom_table
pl/sql_block
```

```
CREATE TRIGGER super_guy.tbu_Connexion_courriel
BEFORE UPDATE OF cCourriel ON super_guy.Connexion
FOR EACH ROW
WHEN (new.cPassword IS NULL)
pl/sql_block
```

Dans le cas de gachette sur les lignes (FOR EACH ROW) les cas suivants s'appliquent :

- ▶ INSERT, OLD ne contient aucune valeur, et NEW contient les nouvelles valeurs.
- ▶ UPDATE, OLD contient les anciennes valeurs, et NEW contient les nouvelles valeurs.
- ▶ DELETE, OLD contient les anciennes valeurs, et NEW ne contient aucune valeur.

# Gachette - détection du mode

Lorsque nous utilisons une gachette unique pour plusieurs modes, il est possible de savoir le mode en utilisant les variables :

## ► INSERTING, UPDATING, DELETING

```
SQL> CREATE OR REPLACE TRIGGER taidu_Langue
  AFTER INSERT OR UPDATE OR DELETE ON LANGUE
DECLARE
  cAction VARCHAR(30);
BEGIN
  IF INSERTING THEN
    cAction := 'Insert';
  ELSIF UPDATING THEN
    cAction := 'Update';
  ELSIF DELETING THEN
    cAction := 'Delete';
  ELSE
    cAction := 'This code is not reachable.';
  END IF;

  DBMS_OUTPUT.PUT_LINE(cAction);
END;
```

## Gachette - clé automatique

Nous savons qu'il est possible de générer les clés primaires à l'aide de SÉQUENCES, maintenant nous allons rendre ceci automatique avec une gachette.

```
SQL> CREATE OR REPLACE TRIGGER tbi_Langue
      BEFORE INSERT
      ON Langue
BEGIN
    -- Gestion de la clé automatique
    SELECT SEQ_Langue.nextval INTO new.pLangue FROM DUAL;
END;
```

```
SQL> INSERT INTO (cLangue) VALUES ('Japonais');
COMMIT;
```



# Table des matières

1. PL/SQL
2. Appel et exécution
3. Gestion des erreurs

- ▶ EXECUTE est une commande SQL\*plus;
- ▶ / est une commande SQL\*plus;

Exemples:

```
SQL> EXECUTE DBMS_OUTPUT.PUT_LINE(100);
```

```
SQL> VAR G_LANGUE VARCHAR2;  
SQL> EXECUTE P_GET_LANGUE(2,:G_LANGUE);  
  
SQL> PRINT :G_LANGUE;
```

- ▶ CALL est une commande SQL;
- ▶ Un appel de fonction avec l'instruction SELECT;

Exemples:

```
SQL> CALL P_CREATION_COMMANDE('2019-10-10',10);
```

```
SQL> SELECT F_NUMROW('LANGUE') FROM DUAL;
```

# Révision - Exemple

## Exemples:

```
SQL>
DECLARE
    vnVal    NUMBER;

    PROCEDURE add(p1    IN    NUMBER,
                  p2    IN    NUMBER,
                  p3    OUT   NUMBER) AS

    BEGIN
        p3 := p1 + p2;
    END;
BEGIN
    -- affectation directe.
    l_number := 1;

    -- affectation via un select.
    SELECT 1 INTO vnVal FROM dual;

    -- affectation via un paramètre de procédure.
    add(1, 2, vnVal);
END;
/
```

# Table des matières

1. PL/SQL
2. Appel et exécution
3. Gestion des erreurs

- Pour voir les erreurs de compilation;

```
SQL> SHOW ERRORS
```

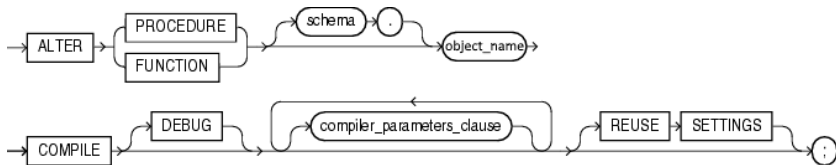
# Gestion des erreurs

- ▶ Déclencher ses propres erreurs;
- ▶ Arrêter le programme avec une erreur spécifique;

```
SQL>
DECLARE
  E_SALAIRE EXCEPTION;
  nSalaire NUMBER := 1000;
  nLimite  NUMBER := 1200;
BEGIN
  IF nSalaire < nLimite THEN
    RAISE E_SALAIRE;
  END IF;
EXCEPTION
  WHEN E_SALAIRE THEN
    DBMS_OUTPUT.PUT_LINE('Salaire trop bas mon ami. ');
    RAISE_APPLICATION_ERROR(-20001, 'trop bas');
END;
/
```

Il est possible qu'une fonction ou procédure ait un statut incertain ou que vous vouliez que le code déjà sauvegardé dans le SGBDR soit implicitement compilé.

Voici le **formalisme officiel provenant d'Oracle Corporation** que j'ai légèrement modifié pour nos besoins.



```
SQL> ALTER FUNCTION ma_fonction COMPILE;
```