

# PL/SQL INTRODUCTION

Introduction à PL/SQ

Les procédures, les fonctions et les packages

Les triggers

Département d'informatique – Collège Lionel Groulx.

Préparé par Saliha Yacoub

## Table des matières.

Introduction.....	2
La section de déclaration des variables :.....	2
Exemples de déclarations.....	3
Le TYPE CURSOR .....	5
La section d'exécution ou contrôle. ....	8
L'instruction de contrôle : L'ALTERNATIVE :.....	8
L'instruction de contrôle : LA RÉPÉTITIVE : .....	11
Les procédures stockées. ....	13
Les fonctions.....	14
Les packages .....	16
Exemple de création d'un Package :.....	17
Les triggers :.....	22
Définition .....	22
Types de TRIGGERS oracle.....	22
Activation et désactivation d'un trigger .....	25
Les prédicats INSERTING, UPDATING, DELETING .....	26
Sources : .....	28

## Introduction

PL/SQL pour Procedural Language/ Structured Query Language est une extension du SQL : des instructions SQL sont intégrées dans les structures de contrôles habituelles. PL/SQL est propriétaire à Oracle.

C'est un langage structuré en BLOCS et un programme PL/SQL est composé de trois blocs dont un obligatoire et deux optionnels.

- Le Bloc déclaratif ou la section déclarative (optionnelle) : dans cette section, on déclare toutes les variables nécessaires à l'exécution du programme PL/SQL. Cette section commence en général par le mot réservé : DECLARE
- Le Bloc ou la section de contrôle ou d'exécution (**obligatoire**) : cette section contient des énoncés SQL ou PL/SQL. Elle débute par le mot réservé BEGIN et se termine par le mot réservé END.
- Le bloc ou la section de gestion des exceptions : (optionnelle) : cette section commence par le mot réservé EXCEPTION. Si une erreur est générée lors de l'exécution d'un programme, celle-ci est envoyée au BLOC EXECPTION, ce qui donne la possibilité de la traiter et de ne pas mettre fin brutalement à l'exécution du Programme.

Un programme PL/SQL peut être anonyme ou stocké dans le serveur sous forme de procédure, fonction ou trigger. Dans tous les cas, le code est **COMPILÉ**

## La section de déclaration des variables :

La section de déclaration de variables commence en général par le mot réservé DECLARE. Tous les types de variables utilisés par SQL peuvent être utilisés dans un bloc PL/SQL. D'autres types de variables sont aussi utilisés.

([http://docs.oracle.com/cd/B10500\\_01/appdev.920/a96624/03\\_types.htm](http://docs.oracle.com/cd/B10500_01/appdev.920/a96624/03_types.htm) )

## Exemples de déclarations

Exemple1

**DECLARE**

NOM VARCHAR2(30);

SALAIRE NUMBER(8,2);

DATEEMBAUCHE DATE;

CHEK BOOLEAN;

Assignation de valeur

PRENOM VARCHAR2(30) := 'PATOCHÉ';

PI CONSTANT NUMBER(6,5) := 3.14159;

LE **%TYPE** permet de déclarer des variable de même type que des variables déjà déclarée.

Exemple2

NOM ETUDIANTS.NOMETUDIANT%TYPE; permet de déclarer la variable NOM avec le même type que la variable NOMETUDIANT de la table ETUDIANT.

Ce type de déclaration est très utile lorsqu'on veut déclarer et manipuler des variables de même type que les colonnes existantes dans les tables de la base de données.

```
Salaire_MIN NUMBER(7,2);
```

```
Salaire_MAX Salaire_MIN%TYPE;
```

```
Acteur VARCHAR2(30);
```

```
Realisateur Acteur%TYPE := 'Spielberg';
```

```
NOM ETUDIANTS.NOMETUDIANT%TYPE
```

Le type **%ROWTYPE**, permet de déclarer une variable de type ligne d'une table.

Exemple 3

```
SET SERVEROUTPUT ON;
DECLARE
ENRETU ETUDIANTS2%ROWTYPE;
NUM ETUDIANTS.NUMAD%TYPE:=12;
BEGIN
SELECT NOM, PRENOM INTO ENRETU FROM ETUDIANTS WHERE NUMAD =NUM;
DBMS_OUTPUT.PUT_LINE('le nom est '|| enretu.nom || 'le prenom est ' ||enretu.prenom);
END;
```

### Le type **RECORD**

Ce type permet de déclarer une variable de type enregistrements.

Exemple 4

```
DECLARE
TYPE ENRE IS RECORD
(
NOM1 ETUDIANTS.NOM%TYPE,
PRENOM1 VARCHAR2(20)
);
numad1 NUMBER :=12;
ENR ENRE;
BEGIN
SELECT NOM, PRENOM INTO ENR.NOM1,ENR.PRENOM1
FROM ETUDIANTS
WHERE numad = numad1;
BMS_OUTPUT.PUT_LINE('le nom est '|| ENR.nom1 || 'le prenom est ' ||ENR.prenom1);
END;
```

## Le TYPE CURSOR

Un curseur est une zone de mémoire utilisée par Oracle pour récupérer les résultats de requêtes SQL qui retourne plusieurs enregistrements.

Il peut être explicite, il est donc associé à une seule requête ou dynamique.

**Les curseurs explicites** : ils sont associés à une seule requête. Ils se déclarent comme suit :

```
CURSOR nom_curseur IS SELECT col1, col2, ... FROM nom_table WHERE ...;
```

Exemple: `CURSOR Resultat IS SELECT nom, prenom from etudiant where nom like %POIT%;`

Pour exploiter les résultats d'un curseur explicite il faut trois étapes :

- 1- Ouvrir le curseur avec la commande PL/SQL `OPEN`
- 2- Lire le curseur avec la commande `FETCH ... INTO` et un `LOOP` : Pour aller chercher chaque enregistrement dans l'ensemble actif, une ligne à la fois, nous utiliserons la commande `FETCH`. À chaque fois que le `FETCH` est utilisé, le curseur avance au prochain enregistrement dans l'ensemble actif.
- 3- Fermer le curseur avec la commande `CLOSE`.

Les curseurs explicites peuvent avoir les attributs suivants :

**%NOTFOUND** : Retourne vrai si le dernier `FETCH` échoue et ne retourne aucun enregistrement

**%FOUND** : inverse de `%NOTFOUND`

**%ISOPEN** : Retourne vrai si le curseur est ouvert

**%ROWCOUNT** : Retourne le nombre d'enregistrements trouvés dans le curseur actif

EXEMPLE :

```
SET SERVEROUTPUT ON;
DECLARE
Nom1 USER1.ETUDIANTS.NOM%TYPE;
Prenom1 varchar2(20);
CURSOR curseur1 IS SELECT nom, Prenom from etudiants;
BEGIN
OPEN curseur1;
LOOP
FETCH curseur1 INTO Nom1, Prenom1;
EXIT WHEN curseur1%NOTFOUND;
DBMS_OUTPUT.PUT_LINE('le nom est '|| nom1 || 'le prenom est '|| prenom1);
END LOOP;
CLOSE Curseur1;
END;
```

**Explications:**

- Une variable Curseur1 de type CURSOR est déclarée pour la requête SELECT.
- Les variables nom1 et prenom1 sont des variables locales destinées à recevoir le contenu du curseur donc de la requête SQL (au moment de la lecture de curseur1)
- On ouvre le curseur curseur1 avec la commande OPEN
- La commande FETCH permet de lire ligne par ligne le contenu du curseur. À chaque fois que cette commande est appelée, le curseur avance au prochain enregistrement dans l'ensemble actif. Le résultat de la lecture est envoyé dans les variables Nom1 et Nom2. LOOP permet de lire l'ensemble des lignes de curseur1
- Curseur1%notfound : retourne vrai si le dernier FETCH échoue et ne retourne aucun enregistrement. %notfound est un attribut du curseur explicite. On sort du LOOP lorsqu'il n'y a aucun enregistrement.

```

SET SERVEROUTPUT ON;
DECLARE
CURSOR CURSEUR1 IS SELECT * FROM etudiants;
ligne CURSEUR1%rowtype;
BEGIN
OPEN CURSEUR1;
LOOP
FETCH CURSEUR1 INTO ligne;
DBMS_OUTPUT.PUT_LINE('le nom est '|| ligne.nom || 'le prenom est '||
LIGNE.prenom);
EXIT WHEN curseur1%NOTFOUND;
END LOOP;
CLOSE CURSEUR1;
END;

```

### Les curseurs dynamiques :

Un curseur dynamique est un curseur qui n'est pas associé à une seule requête SQL. Dans ce cas, lors de sa déclaration, il faudra utiliser le mot réservé **REF**.

Exemple: TYPE enregistrementEtudiant is REF CURSOR

- Un REF CURSOR est vu comme un pointeur sur une zone mémoire dans le serveur (contenant le résultat d'une requête).
- Un REF CURSOR n'est pas Updatable.
- Un REF CURSOR est forward-only.
- Il peut retourner des valeurs ou non.

Exemple

```
TYPE enregistrementEtudiant is REF CURSOR;
```

- Il a un grand avantage lorsque votre procédure ou fonction PL/SQL est appelée par un langage de haut niveau comme C# ou JAVA.

Ce type de curseur est sera détaillé dans la section packages.



## La section d'exécution ou contrôle.

Cette section commence toujours par BEGIN et se termine par END. On retrouve dans cette section (BLOC) des commandes SQL ainsi que les structures de contrôles habituelles.

### L'instruction de contrôle : L'ALTERNATIVE :

L'alternative se présente sous les formats suivant :

#### 1- IF THEN END IF

**IF condition THEN séquence d'instruction END IF**

```
DECLARE  
CHOIX NUMBER ;  
BEGIN  
IF CHOIX =1 THEN UPDATE ETUDIANTS SET CYCLE =2 WHERE NUMAD=12;  
END IF;  
END;
```

#### 2- IF- THEN - ELSE END IF

**IF condition THEN séquence instruction1 ELSE séquence instruction2 END IF**

```
DECLARE  
CHOIX NUMBER ;  
BEGIN  
IF CHOIX =1 THEN UPDATE ETUDIANTS SET CYCLE =2 WHERE NUMAD=12;  
ELSE DELETE FROM ETUDIANTS WHERE NUMAD =12;  
END IF;  
END;
```

### 3- IF- THEN - ELSIF END IF

```
IF condition THEN
Séquence_instructions1;
ELSIF condition THEN
Séquence_instructions2;
ELSIF condition THEN
Séquence_instructions3;
ELSIF condition THEN
Séquence_instructions4;
ELSE
Séquence_instructions5;
END IF;
```

```
DECLARE
CHOIX NUMBER ;
BEGIN
IF CHOIX =1 THEN UPDATE ETUDIANTS SET CYCLE =2 WHERE NUMAD=12;
ELSIF CHOIX =2 THEN UPDATE ETUDIANTS SET CYCLE =2;
ELSIF CHOIX =3 THEN INSERT INTO ETUDIANTS(NUMAD, NOM)
VALUES(13,'PATOCHÉ');
ELSE DELETE FROM ETUDIANTS WHERE NUMAD =12;
END IF;
END;
```

#### REMARQUE

Dans une alternative, les mots réservés IF, THEN et END IF sont obligatoires. Les autres (ELSIF et ELSE )sont optionnels

L'alternative : **CASE WHEN.**

Permet d'exécuter un bloc PL/SQL selon la valeur d'une variable.

```
CASE  
  
  WHEN condition1 THEN  
    Séquence_instructions  
  
  WHEN condition2 THEN  
    Séquence_instructions1;  
  
  ELSE  
    Séquence_instructions2;  
  
END;
```

```
CREATE OR REPLACE PROCEDURE MiseAJour(CHOIX IN NUMBER) AS  
BEGIN  
  CASE CHOIX  
  WHEN 1 THEN INSERT INTO employes (nom, salaire )VALUES('pataoche',28000);  
  COMMIT;  
  WHEN 2 THEN UPDATE employes SET salaire = salaire +100 where  
  numemp =1480;  
  COMMIT;  
  ELSE dbms_output.put_line('pas bon choix');  
  END CASE;  
END;
```

## L'instruction de contrôle : LA RÉPÉTITIVE :

L'instruction LOOP

### 1- LOOP Sequence\_instructions EXIT WHEN condition END LOOP

```
CREATE OR REPLACE FUNCTION COMPTER RETURN NUMBER AS
compteur NUMBER :=0;
BEGIN
LOOP
compteur:=compteur+1;
EXIT WHEN compteur=10;
END LOOP ;
RETURN compteur;
END;
```

On peut décider de sortir de la boucle avec l'instruction EXIT et un IF.

Exemple

```
SET SERVEROUTPUT ON;
DECLARE
Credit NUMBER :=0;
BEGIN
LOOP
Credit :=Credit + 1;
IF Credit> 3 THEN
EXIT;
END IF;
END LOOP;
DBMS_OUTPUT.PUT_LINE ('Credit : ' || TO_CHAR(Credit));
END;
```

## 2- WHILE condition LOOP instructions END LOOP

```
SET SERVEROUTPUT ON;
DECLARE
I NUMBER:=1;
BEGIN
WHILE I < 10 LOOP
I:= I+1;
DBMS_OUTPUT.PUT_LINE (TO_CHAR(I));
END LOOP;
END;
```

## 3- FOR compteur IN [REVERSE] valeurs LOOP sequencesInstructions END LOOP

```
SET SERVEROUTPUT ON;
BEGIN
FOR i IN 1..3 LOOP
DBMS_OUTPUT.PUT_LINE (TO_CHAR(i));
END LOOP;
END;
```

REVERSE permet de faire une décrémentation.

```
SET SERVEROUTPUT ON;
BEGIN
FOR i IN REVERSE 1..3 LOOP
DBMS_OUTPUT.PUT_LINE (TO_CHAR(i));
END LOOP;
END;
```

## Les procédures stockées.

Une procédure est un code PL/SQL défini par l'utilisateur et stocké dans la base de données. Ce qui permet d'éliminer la redondance de code.

Avantages:

Le code SQL est **précompilé**.

Exécution plus rapide (puisque stockée dans le serveur)

Moins de code redondant

Syntaxe générale:

```
CREATE OR REPLACE PROCEDURE schema.NOMProcédure
(
  param1 [IN|OUT|IN OUT ] typeparam1,
  param2 [IN|OUT|IN OUT ] typeparam2,
) AS|IS

Déclaration des variables sans le mot DECLARE

BEGIN

BLOC PL/SQL

END;
```

CREATE indique que l'on veut créer une procédure.

OR REPLACE (facultati) permet d'écraser la procédure portant le même nom

Param1, param2 sont les paramètres de la procédure

IN : indique que le paramètre transmis par le programme appelant n'est pas modifiable par la procédure. (entrée) par défaut les paramètres sont IN

OUT: indique que les paramètres sont modifiables par la procédure (sortie)

IN OUT: combinaison de IN et OUT

**Pour exécuter une procédure dans SQLDeveloper on utilise la commande EXECUTE**

Exemple :

```
CREATE OR REPLACE PROCEDURE INSERTION(  
PNOM IN EMPLOYES.NOM%TYPE,  
PPRN IN EMPLOYES.PRENOM%TYPE) AS  
BEGIN  
INSERT INTO EMPLOYES(NOM, PRENOM) VALUES (PNOM, PPRN);  
END;
```

Pour exécuter la procédure : EXECUTE INSERTION('POITRAS', 'REMI');

### **IMPORTANT :**

Les variables ou les paramètres de votre procédure doivent être différents des noms de colonnes que vous manipulez.

Lorsqu'une procédure a besoin de déclarer ses variables, pas besoins de DECLARE. Les variables sont déclarées entre IS et BEGIN

Les mêmes remarques s'appliquent à une fonction.

## **Les fonctions**

Les fonctions sont considérées comme des procédures qui retournent des valeurs.

```
CREATE OR REPLACE FUNCTION schema.NOMFonction  
(  
param1 [IN|OUT|IN OUT ] typeparam1,  
param2 [IN|OUT|IN OUT ] typeparam2,  
) RETURN TypeDeVariable AS|IS  
  
Déclaration des variables sans le mot DECLARE  
BEGIN  
BLOC PL/SQL  
END;
```

Exemple :

```
CREATE OR REPLACE  
FUNCTION CALCULER (code IN SCOTT.EMP.deptno%TYPE)  
RETURN NUMBER AS  
-- DÉCLARATION DE LA VARIABLE total  
total NUMBER;  
BEGIN  
SELECT COUNT(*) INTO total FROM SCOTT.EMP GROUP BY deptno  
HAVING deptno =code;  
RETURN total;  
END CALCULER;
```

Pour exécuter la fonction, vous utilisez la syntaxe :

SELECT nomFonction(paramètres) FROM DUAL

```
SELECT CALCULER(20) FROM DUAL;
```

Exemple 2

```
CREATE OR REPLACE FUNCTION LISTECLIENTS RETURN  
SYS_REFCURSOR AS  
RESULTAT SYS_REFCURSOR;  
BEGIN  
OPEN RESULTAT FOR SELECT * FROM CLIENTS;  
RETURN RESULTAT;  
END LISTECLIENTS;
```

Pour supprimer une procédure ou une fonction, utiliser la commande SQL DROP.

EXEMPLE : DROP FUNCTION LISTECLIENTS;



## Les packages

Un package est un objet de la base de données qui encapsule d'autres objets (procédures, fonctions). IL a essentiellement deux parties:

1. La partie déclaration ou spécification du package: dans cette partie sont déclarées les variables globales, les curseurs, les procédures et les fonctions.
2. La partie corps du programme: dans cette partie, sont définies les procédures et les fonctions et les curseurs.

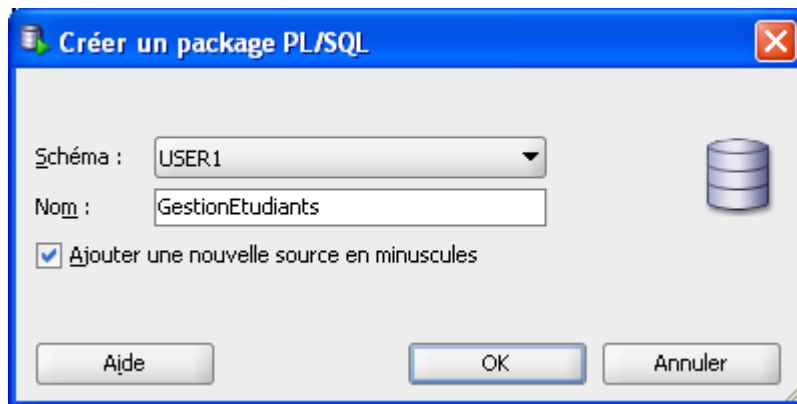
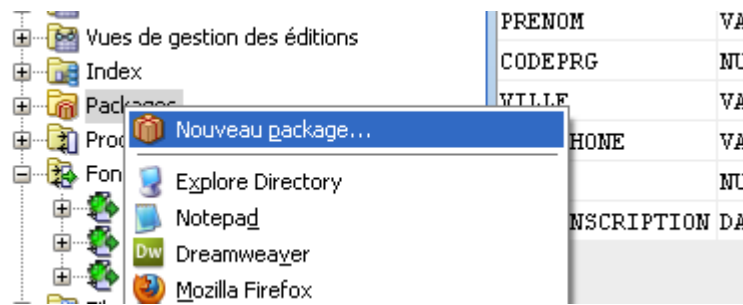
Quelques avantages à utiliser les packages:

- Modularité : le fait de regrouper logiquement les éléments PL/SQL liés rend la compréhension plus facile des différents éléments du package.
- Facilité de développement : il est possible de définir uniquement la partie spécification du package. Le corps du package sera défini que pour l'exécution de l'application
- Meilleure performance : le package est présent en mémoire dès l'appel d'un élément qui le compose (fonction ou procédure) ce qui rend l'accès aux éléments du package beaucoup plus rapide que l'appel à des procédures ou à des fonctions indépendantes.

## Exemple de création d'un Package :

Afin de vous aider à créer facilement les packages pour votre TP no 2, voici un résumé des étapes nécessaires pour la création et d'un package et d'un Body Package.

Sous l'objet Package, choisir nouveau package

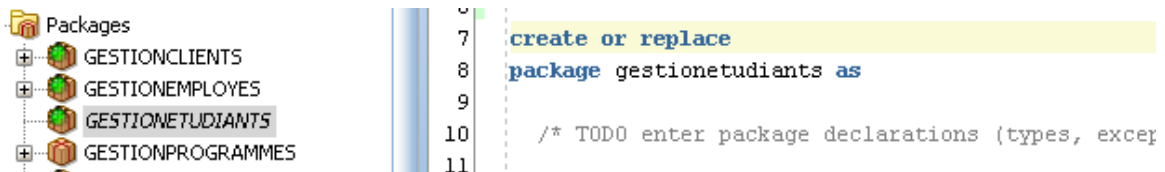


Puis cliquez sur OK.

Dans le Package, définir toutes les procédures, les fonctions et les variables globales :

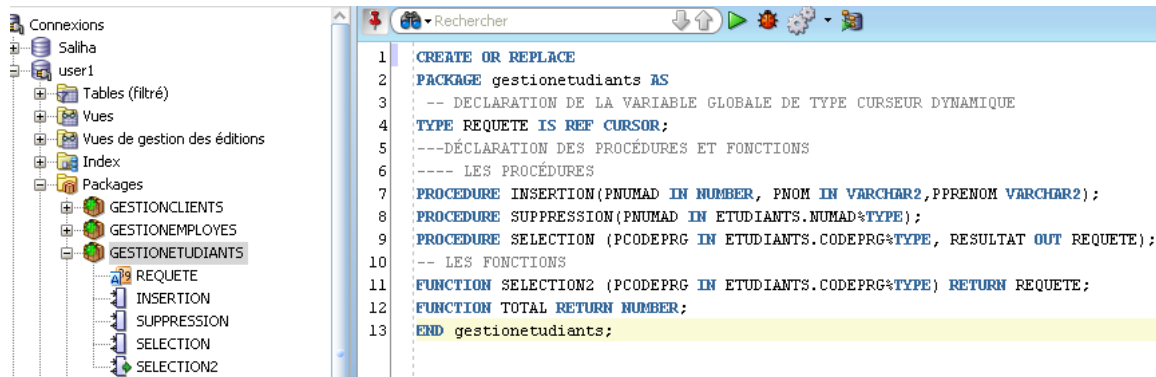
Enregistrez.

Après l'enregistrement, vous allez voir votre package apparaître sous l'onglet Package.



Au fur et à mesure que vous modifiez votre package, enregistrez, en principe la compilation se fait en même temps. Si votre package est vert, alors tout va bien, sinon, regarder dans le journal les messages erreurs.

À la fin de la définition de votre package, vous allez voir toutes les procédures, les fonctions et les variables globales que vous venez de déclarer.



The screenshot shows the SQL Developer interface. On the left, the 'Packages' folder under 'user1' is expanded, showing the package 'GESTIONETUDIANTS' and its sub-items: 'REQUETE', 'INSERTION', 'SUPPRESSION', 'SELECTION', and 'SELECTION2'. The main editor window displays the following SQL code:

```
1 CREATE OR REPLACE
2 PACKAGE gestionetudiants AS
3 -- DECLARATION DE LA VARIABLE GLOBALE DE TYPE CURSEUR DYNAMIQUE
4 TYPE REQUETE IS REF CURSOR;
5 ---DÉCLARATION DES PROCÉDURES ET FONCTIONS
6 ---- LES PROCÉDURES
7 PROCEDURE INSERTION(PNUMAD IN NUMBER, PNOM IN VARCHAR2,PPRENOM VARCHAR2);
8 PROCEDURE SUPPRESSION(PNUMAD IN ETUDIANTS.NUMAD%TYPE);
9 PROCEDURE SELECTION (PCODEPRG IN ETUDIANTS.CODEPRG%TYPE, RESULTAT OUT REQUETE);
10 -- LES FONCTIONS
11 FUNCTION SELECTION2 (PCODEPRG IN ETUDIANTS.CODEPRG%TYPE) RETURN REQUETE;
12 FUNCTION TOTAL RETURN NUMBER;
13 END gestionetudiants;
```

Voici le code correspondant à votre package.

```
CREATE OR REPLACE

PACKAGE gestionetudiants AS

-- DECLARATION DE LA VARIABLE GLOBALE DE TYPE CURSEUR DYNAMIQUE

TYPE REQUETE IS REF CURSOR;

---DÉCLARATION DES PROCÉDURES ET FONCTIONS

---- LES PROCÉDURES

PROCEDURE INSERTION(PNUMAD IN NUMBER, PNOM IN VARCHAR2,PPRENOM
VARCHAR2);

PROCEDURE SUPPRESSION(PNUMAD IN ETUDIANTS.NUMAD%TYPE);

PROCEDURE SELECTION (PCODEPRG IN ETUDIANTS.CODEPRG%TYPE, RESULTAT OUT
REQUETE);

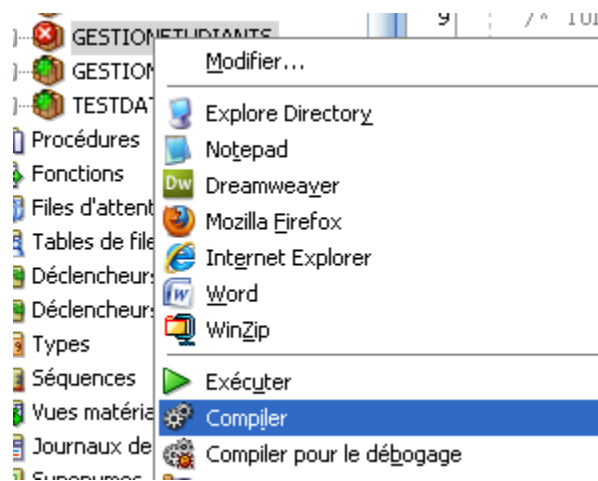
-- LES FONCTIONS

FUNCTION SELECTION2 (PCODEPRG IN ETUDIANTS.CODEPRG%TYPE) RETURN REQUETE;

FUNCTION TOTAL RETURN NUMBER;

END gestionetudiants;
```

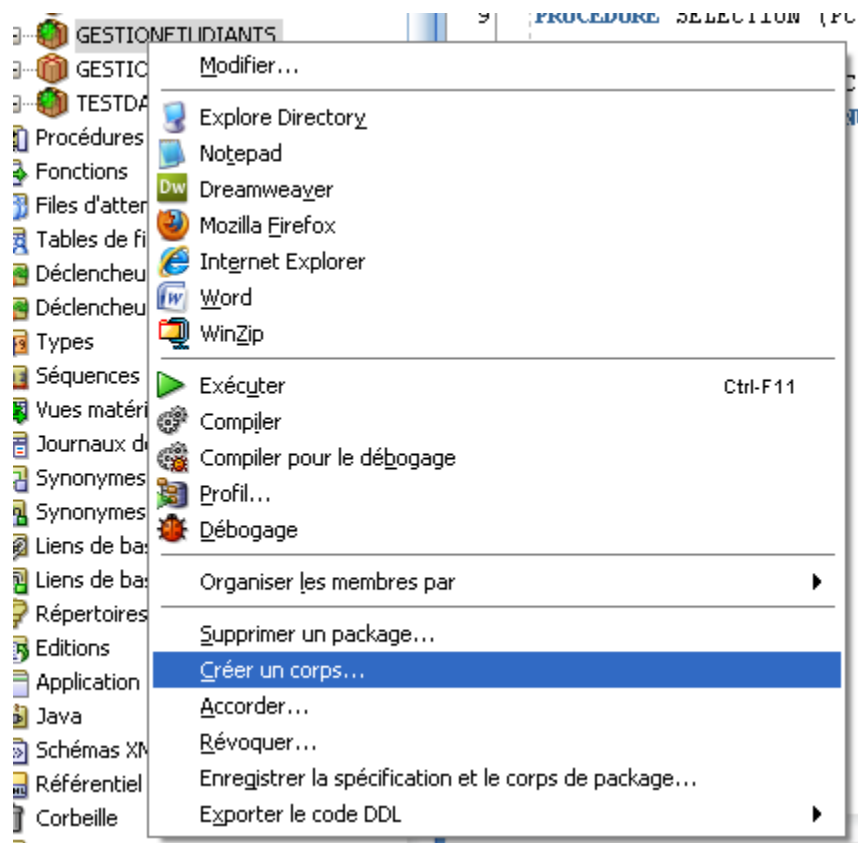
Vous devez compiler votre Package afin qu'il soit sans erreurs.



**Le résultat de la compilation se trouve dans le Journal (en bas à gauche)**

Vos fonctions, procédures étant déclarées, vous devez maintenant définir ce qu'elles devront faire. Ceci doit se faire dans le BODY PACKAGE.

Sur le bouton droit, et sur le package que vous venez de créer, choisir Créer un Corp.



Voici ce que donne l'opération une fois terminée.

```
create or replace
package body gestionetudiants as

) PROCEDURE INSERTION(PNUMAD IN NUMBER, PNOM IN VARCHAR2,PPRENOM VARCHAR2) as
begin
  /* TODO implementation required */
  null;
end INSERTION;

) PROCEDURE SUPPRESSION(PNUMAD IN ETUDIANTS.NUMAD%TYPE) as
begin
  /* TODO implementation required */
  null;
end SUPPRESSION;
```

--- Le reste des procédures et la fin du Package

```
FUNCTION TOTAL RETURN NUMBER as
begin
  /* TODO implementation required */
  return null;
end TOTAL;

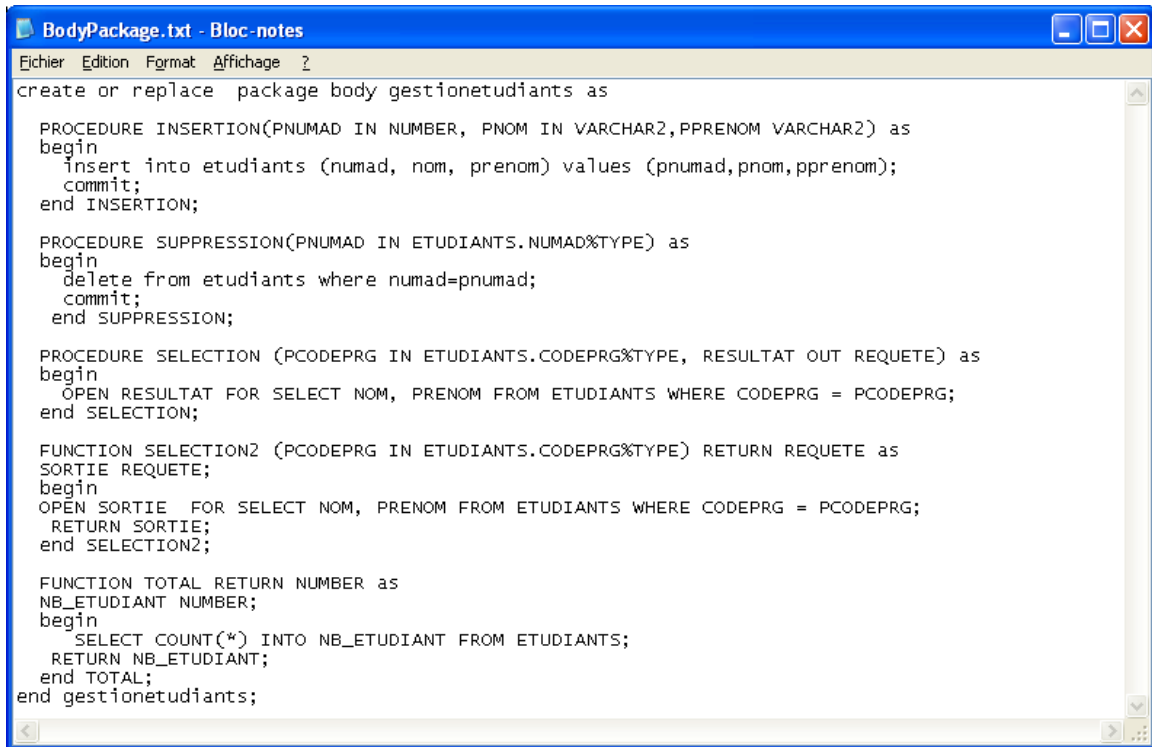
end gestionetudiants;
```

Vous allez définir et compiler les procédures une à une. Il est très important de vérifier les procédures et fonction une à la fois.

**La compilation de votre Body Package se fait de la même façon que la compilation de votre Package.**

Au fur et à mesure que vous définissez les procédures et ou fonction, enlever le NULL du code.

Voici le code correspondant à la définition de votre Body Package.



```
BodyPackage.txt - Bloc-notes
Fichier Edition Format Affichage ?
create or replace package body gestionetudiants as

  PROCEDURE INSERTION(PNUMAD IN NUMBER, PNOM IN VARCHAR2,PPRENOM VARCHAR2) as
  begin
    insert into etudiants (numad, nom, prenom) values (pnumad,pnom,pprenom);
    commit;
  end INSERTION;

  PROCEDURE SUPPRESSION(PNUMAD IN ETUDIANTS.NUMAD%TYPE) as
  begin
    delete from etudiants where numad=pnumad;
    commit;
  end SUPPRESSION;

  PROCEDURE SELECTION (PCODEPRG IN ETUDIANTS.CODEPRG%TYPE, RESULTAT OUT REQUETE) as
  begin
    OPEN RESULTAT FOR SELECT NOM, PRENOM FROM ETUDIANTS WHERE CODEPRG = PCODEPRG;
  end SELECTION;

  FUNCTION SELECTION2 (PCODEPRG IN ETUDIANTS.CODEPRG%TYPE) RETURN REQUETE as
  SORTIE REQUETE;
  begin
    OPEN SORTIE FOR SELECT NOM, PRENOM FROM ETUDIANTS WHERE CODEPRG = PCODEPRG;
    RETURN SORTIE;
  end SELECTION2;

  FUNCTION TOTAL RETURN NUMBER as
  NB_ETUDIANT NUMBER;
  begin
    SELECT COUNT(*) INTO NB_ETUDIANT FROM ETUDIANTS;
    RETURN NB_ETUDIANT;
  end TOTAL;
end gestionetudiants;
```

Dans SQL Developer, vous pouvez tester certaines de vos procédures et fonctions de la façon suivantes.

```
SELECT GESTIONEMPLOYES.total FROM dual;
```

```
SELECT gestionetudiants.SELECTION2(420)FROM dual;
```

```
EXECUTE gestionetudiants.insertion(33,'Deschamps','Martin');
```

```
EXECUTE gestionetudiants.suppression(18);
```

**Important: lorsque la procédure et ou la fonction se trouve dans un package, le nom de celle-ci doit-être précédé du nom du package.**

## Les triggers :

### Définition

Les triggers sont des procédures stockées qui s'exécutent automatiquement quand un événement se produit. En général cet événement représente une opération DML (Data Manipulation Language ) sur une table. Les instructions DML doivent inclure INSERT, UPDATE ou DELETE

Ils permettent entre autre de contrôler les accès à la base de données.

Syntaxe simplifiée pour créer un trigger avec une opération DML

```
CREATE [OR REPLACE] TRIGGER nomtrigger
BEFORE [ AFTER] INSERT OR UPDATE OR DELETE
ON Nomdetable [FOR EACH ROW] [WHEN condition]
BLOC PL/SQL
```

**Attention: le nom du trigger doit être unique dans la base de données.**

- L'option BEFORE /AFTER indique le moment du déclenchement du trigger.
- Les instructions SQL INSERT OR UPDATE OR DELETE peuvent être toutes présentes comme on peut en avoir juste une.
- Pour un UPDATE, on peut spécifier une liste de colonnes. Dans ce cas, le trigger ne se déclenchera que s'il porte sur l'une des colonnes précisées dans la liste.(voir exemple plus loin)

### Types de TRIGGERS oracle

Oracle propose deux types de triggers :

1. Les triggers lignes qui se déclenchent individuellement pour chaque ligne de la table affectée par le trigger. L'option FOR EACH ROW est alors spécifiée
2. Les triggers globaux qui sont déclenchés une seule fois, L'option FOR EACH ROW n'est pas spécifiée

### Exemple1 (TRIGGER GLOBAL)

```
CREATE OR REPLACE TRIGGER ctrlmiseajour
BEFORE INSERT OR DELETE OR UPDATE ON EMPLOYES
DECLARE MESSAGE EXCEPTION;
BEGIN
IF (TO_CHAR(SYSDATE,'DY')= 'SAM.' OR TO_CHAR(SYSDATE,'DY')= 'DIM.')
THEN RAISE MESSAGE;
END IF;
EXCEPTION
WHEN MESSAGE THEN RAISE_APPLICATION_ERROR(-20324,'on ne met
pas à jour la fin de semaine');
END;
```

On peut également utiliser les triggers pour contrôler les valeurs qui ont été insérées.  
L'exemple le plus souvent utilisé et l'utilisation de trigger pour faire une incrémentation automatique de la clé primaire.

### Exemple 2 TRIGGER SUR CHAQUE LIGNE

```
CREATE OR REPLACE TRIGGER EMPINS
BEFORE INSERT ON EMPLOYES
FOR EACH ROW
BEGIN
IF :NEW.NUMEMP IS NULL THEN
SELECT SQEMP2 .NEXTVAL INTO :NEW.NUMEMP FROM DUAL;
END IF;
END;
```

La séquence a été créée comme suit :

```
CREATE SEQUENCE SQEMP2 INCREMENT BY 1
```



START WITH 10000;

On exécute ensuite l'instruction INSERT sans le numemp;

IMPORTANT : lors de la création d'un trigger, les anciennes valeurs sont nommées

**:old.colonne** (faire attention aux deux points)

Les nouvelles valeurs sont nommées

**:new.colonne** (faire attention aux deux points)

### Exemple 3

Contrôler les salaires : ils doivent être plus grands que le plus petit des salaires et plus petit que le plus grand des salaires

```
CREATE OR REPLACE
TRIGGER CTRLSAL2
BEFORE INSERT ON employes
FOR EACH ROW
DECLARE
minsal number;
maxsal number;
BEGIN
/* retrouver le salaire minimum et maximum dans la table employes */
SELECT MIN(salaire) INTO minsal FROM Employes;
SELECT Max(salaire) INTO maxsal FROM Employes;
/* s'il y a un problème, on provoque une erreur */
IF (:new.salaire<minsal OR :new.salaire>maxsal)
THEN
raise_application_error (-20300,'Salaire incorrect');
END IF;
END;
```

### Exercice :

Écrire un trigger qui permet de mettre le salaire à 5000\$ si un employé n'a pas de salaire.

Solution:

```
CREATE TRIGGER DefaultSalaire
BEFORE INSERT OR UPDATE OF salaire ON employe
FOR EACH ROW WHEN (new.salaire is null)
BEGIN
SELECT 5000
INTO :new.salaire
FROM dual;
END;
```

### Activation et désactivation d'un trigger

Pour qu'un trigger soit déclenché celui-ci doit être ACTIVÉ. Par défaut, les triggers d'oracle sont activés dès leur création. On peut activer un triggers par SQL Développeur ou par la commande **ALTER TRIGGER nomtrigger ENABLE**.

Si un trigger est désactivé, alors il est stocké mais ignoré

Pour désactiver un trigger, utiliser la même commande ALTER TRIGGER avec l'option DISABLE.

On peut désactiver TOUS les triggers associés à une table par la commande **ALTER TABLE DISABLE ALL TRIGGERS**.

Pour les réactiver de nouveau, utiliser **ALTER TABLE ENABLE ALL TRIGGERS**.

Pour supprimer un trigger, utiliser **DROP TRIGGER nomduTrigger**

## Les prédicats INSERTING, UPDATING, DELETING

Lorsqu'un trigger porte sur les opérations DML, nous pouvons ajouter des prédicats dans notre code, pour indiquer les opérations de déclenchement :

```
CREATE TRIGGER ...  
BEFORE INSERT OR UPDATE ON employe  
.....  
BEGIN  
.....  
IF INSERTING THEN ..... END IF;  
IF UPDATING THEN ..... END IF;  
.....  
END;
```

Exemple :

On veut enregistrer les opérations réalisées sur notre table

1

```
CREATE TRIGGER ctrlOperation  
BEFORE  
DELETE OR INSERT OR UPDATE  
ON Employe FOR EACH ROW  
BEGIN  
IF INSERTING THEN  
INSERT INTO table_ctrl VALUES ( USER, 'Insertion');  
ELSIF DELETING THEN  
INSERT INTO table_ctrl VALUES ( USER, 'Destruction');  
ELSIF UPDATING THEN  
INSERT INTO table_ctrl VALUES ( USER, 'Modification');  
ELSIF UPDATING(salaire) THEN  
INSERT INTO table_ctrl VALUES ( USER, 'Modification du salaire');  
END IF;  
END;
```

---

<sup>1</sup> Source : SQL, Denis Brunet

## IMPORTANT :

Un trigger ligne ne peut pas lire (SELECT) et/ou modifier la table concernée (appelée table mutante) par l'instruction (INSERT, UPDATE ou DELETE) qui a déclenché ce trigger.

### **Exercice 1 :**

Écrire un trigger qui permet de contrôler les insertions et les modifications dans la table Commandes, comme suit :

**La date de livraison ne doit pas être inférieure à la date de commande**

### **Exercice 2**

Écrire un trigger qui permet de contrôler les ajouts et les modifications des salaires. Si le JOB n'est pas 'PRESIDENT', alors Lors de l'ajout, modification d'un nouvel employé, nous devons vérifier si le salaire de celui-ci se retrouve dans la fourchette des salaires. Les salaires sont en fonction de JOB et sont dans la table Grille

**Sources :**

[http://download.oracle.com/docs/html/B16022\\_01/ch3.htm#i1017636](http://download.oracle.com/docs/html/B16022_01/ch3.htm#i1017636)

[http://download.oracle.com/docs/cd/B19306\\_01/server.102/b14200/statements\\_7004.htm](http://download.oracle.com/docs/cd/B19306_01/server.102/b14200/statements_7004.htm)

[http://docs.oracle.com/cd/B19306\\_01/appdev.102/b14261/packages.htm](http://docs.oracle.com/cd/B19306_01/appdev.102/b14261/packages.htm)

[http://docs.oracle.com/cd/B12037\\_01/appdev.101/b10807/13\\_elems011.htm](http://docs.oracle.com/cd/B12037_01/appdev.101/b10807/13_elems011.htm)