Le langage PL/SQL

2ème Année Cycle d'ingénieur « *Informatique et Ingénierie des données* »

ENSA Khouribga

Pr. SOUSSI Nassima

Plan du Cours (Suite)

VII. Les packages

VIII. Les déclencheurs

Les Packages

Package : Définition

– **Définition** :

Un package est un **ensemble** de procédures et/ou fonctions **fonctionnellement dépendantes** regroupées dans un objet nommé.

– **Rôle** :

Permettre au serveur Oracle de lire simultanément plusieurs objet en mémoire.

– **Exemple** :

- *UTL_FILE* : regroupe les procédures et fonctions permettant de lire et écrire des fichiers du système d'exploitation.

Package : Composants de base

Un package est composé de 2 parties : Spécification et Corps.

1. **la spécification (déclaration) :**

Elle permet de déclarer :

- les entêtes des fonctions et procédures *publiques* contenues dans le package;
- les variables, constantes, exceptions et curseurs utilisés dans le package et *visibles par le programme appelant*.

Package : Composants de base

2. Le corps (définition) :

- contient la *définition* de tous les objets publics listés dans la partie spécification.
- peut *inclure* des objets qui ne sont pas listés dans la partie spécification, et sont donc privés.

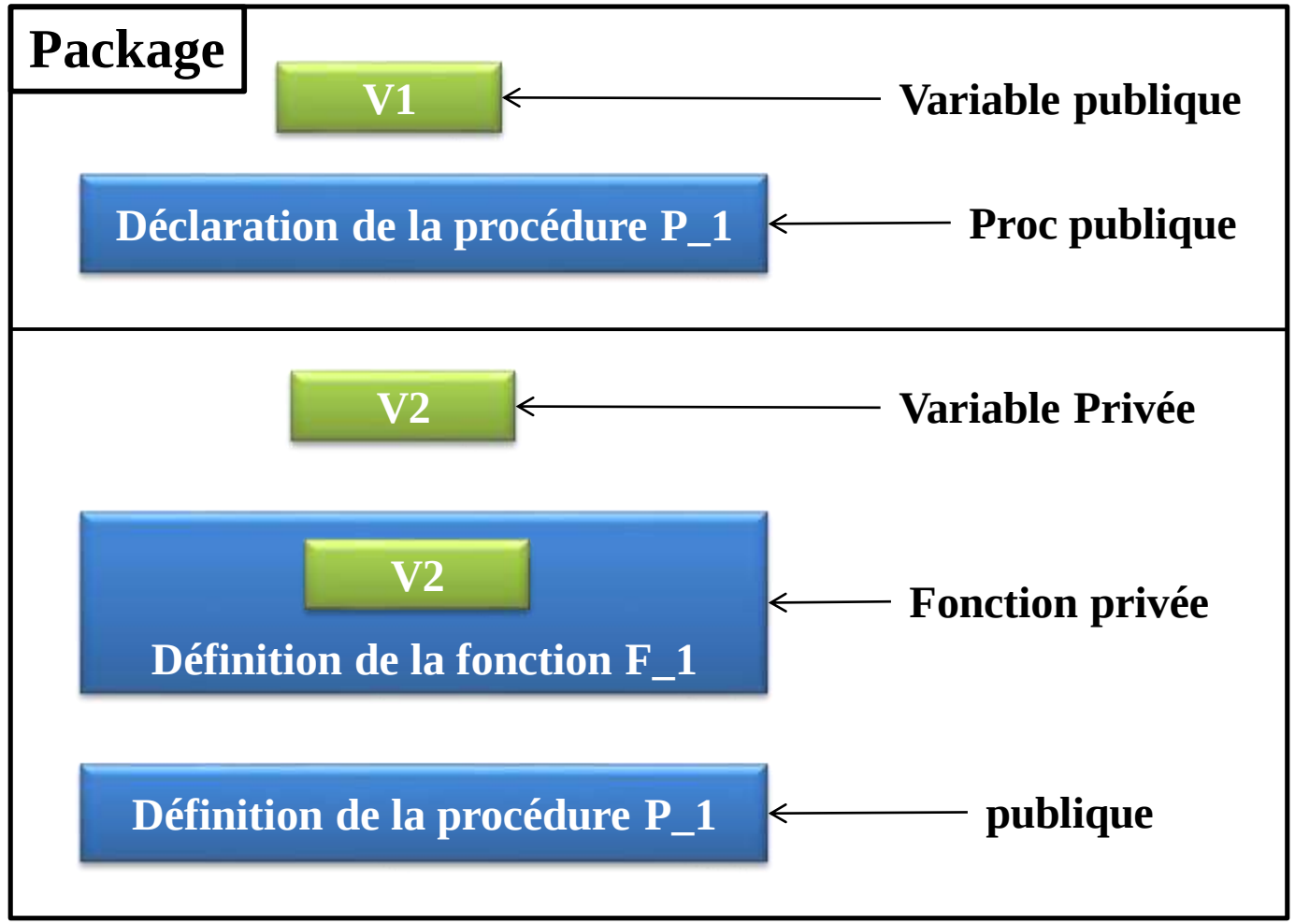
⇒ Ils ne sont pas visibles à l'extérieur du corps du package.

⇒ Ils doivent être déclarées avant d'être utiliser dans les structures publiques.

Package : Composants de base

Spécification

Corps



Package : Syntaxe

```
CREATE  [OR REPLACE] PACKAGE <nom_Package>  IS
        [déclaration-de-variable;]
        [déclaration-de-curseur;]
        [déclaration-de-procédure/fonction;]
END [nom_package] ;

CREATE [OR REPLACE] PACKAGE BODY <nom_Package> IS
        [Définitions_des_elts_privées;]
        [définition-de-variable;]
        [définition-de-curseur;]
        [définition-de-procédure/fonction;]
END [nom_package] ;
```

Package : Exemple

```
CREATE  PACKAGE ENSA_package IS
    PROCEDURE  Resultat_final(v_id   etudiants.id_etd%TYPE);
    …
END ENSA_package;
```

Package : Exemple

```
CREATE  PACKAGE BODY ENSA_package IS
    FUNCTION valider(v_id  etudiants.id_etd%TYPE) RETURN BOOLEAN
    IS v_note NUMBER := 0
    BEGIN
        SELECT note INTO v_note FROM etudiants WHERE id_etd = v_id;
        If   v_note >= 12   THEN   RETURN (true);
        ELSE
              RETURN(false);
        END IF;
    END valider;
    …
END ENSA_package;
```

Package : Exemple

```
CREATE  PACKAGE BODY ENSA_package IS
    …
    PROCEDURE  Resultat_final(v_id   etudiants.id_etd%TYPE)  IS
    BEGIN
        If  valider(v_id) THEN
              dbms_output.put_line('l étudiant' || v_id || 'as validé le module');
        ELSE dbms_output.put_line('l étudiant' || v_id || 'n as pas validé le module');
        END IF;
    END Resultat_final;
END ENSA_package;
```

Appeler un élément du package

- **En PL/SQL :**

  nom-package.nom-élément;

- **SQL*PLUS :**

  EXECUTE nom-package.nom-variable := ...

  EXECUTE nom-package.nom-procedure(valeurs_paramètres)

  EXECUTE :nom-variable := nom-package.nomfunction(valeurs_paramètres);

  Remarque : Pour appeler un élément public du package dans un autre schéma, on le précède par le nom de l'utilisateur.

  nom_utilisateur.nom-package.nom-élément;

Package : Extraction d'informations

– **Pour afficher les informations d'un package :**

```
SELECT * FROM user_objects WHERE object_name='nom_package'
```

– **Pour afficher le code sources d'une spécification :**

```
SELECT * FROM user_source
WHERE  type ='PACKAGE' and name='nom_package'
```

– **Pour afficher le code sources d'un corps :**

```
SELECT * FROM user_source
WHERE  type ='PACKAGE BODY' and name='nom_package'
```

Package : Surcharge de fonction

A l'intérieur d'un package, il est possible de surcharger une procédure ou une fonction.

=> définir plusieurs procédures ou fonctions avec le même nom mais avec une liste de paramètres différente.

Package : Suppression

- **Supprimer la totalité du package :**

```
DROP PACKAGE nom_package ;
```

- **Supprimer seulement le corps :**

```
DROP PACKAGE BODY nom_package ;
```

Les Triggers (Déclencheurs)

Trigger : Définition

– Les *Triggers* ou *déclencheurs*, sont des **actions** qui sont **lancées automatiquement suite à un événement**, généralement une mise à jour de données.

– Ils servent à lancer automatiquement d'autres actions comme une propagation de mise à jour pour maintenir une cohérence.

➢ **Exemple :**

l'incrémentation de 1 de l'effectif des employés d'un département suite à l'insertion dans la base de donnée d'un employé travaillant dans ce département.

Trigger : Définition

En PLSQL, un *trigger* est un **programme PLSQL exécuté quand un événement arrive**. Ces événements peuvent être:

- Une instruction *INSERT*, *UPDATE* ou *DELETE* sur une *table* ou une *vue* => *on parle de* ***Trigger LMD***
- Une instruction *CREATE*, *ALTER* ou *DROP* sur un objet => *on parle de* ***Trigger LDD***
- Le démarrage ou l'arrêt de la base (*STARTUP* ou *SHUTDOWN*), une erreur spécifique (*no_data_found, etc...*), une connexion ou une déconnexion d'un utilisateur => On parle de ***Trigger d'instances***.

Trigger : Avantages

Les déclencheurs peuvent être écrits aux fins suivantes :

- Imposer des autorisations de sécurité
- Empêcher les transactions invalides
- Faire respecter la cohérence des données
- Réplication des données
- …

Trigger DML : Définition

– Ils sont lancés par une opération INSERT ou UPDATE ou DELETE.

– Le même déclencheur peut s'activer par les trois opérations

– Pour *UPDATE*, on peut spécifier une liste de colonnes.

⇒ Dans ce cas, le trigger ne se déclenchera que si l'instruction update porte sur l'une au moins des colonnes précisée dans la liste.

Trigger DML : Types

On distingue deux types de déclencheur :

– **Déclencheur *ligne*** :

est exécuté pour chaque ligne concernée par l'opération LMD.

– **Déclencheur *global/ d'état /d'instruction*** :

ne s'exécute qu'une fois par instruction LMD

Trigger DML : Eléments de base

La création d'un déclencheur comporte les éléments suivants :

1. ***Moment du déclenchement*** : *Avant ou aprés*
2. ***Evénement déclencheur*** : *opération DML*
3. ***Nom de la table***
4. ***Type de déclencheur*** : *ligne ou global*
5. ***Clause WHEN*** : *condition restrictive par ligne*
6. ***Corps du déclencheur*** : *bloc PL/SQL*

Trigger DML : Syntaxe générale

```
CREATE [or REPLACE] TRIGGER <nom_trigger>
{BEFORE | AFTER}   -- Timing
Event_1 [OR event_2 OR event_3]  -- INSERT, UPDATE, DELETE
ON <nom_table_mise_à_jour>
[FOR EACH ROW] [REFERENCING OLD AS old | NEW AS new]
[WHEN (condition)] – dédié au trigger ligne
Trigger_body   -- bloc PL/SQL
```

Remarque :

Les noms des déclencheurs doivent être unique au sein du même schéma.

Trigger DML : Moment de déclenchement

- **CREATE** [or REPLACE] **TRIGGER** <nom_trigger> :

  Cette clause crée un déclencheur avec le nom donné ou écrase un déclencheur existant avec le même nom.

- **{BEFORE | AFTER} :** Moment de déclenchement

  Elle précise le moment de l'exécution du déclencheur (avant ou après l'opération).

  - *BEFORE* : exécution du corps du trigger *avant le déclenchement de l'événement DML* sur une table.
  - *AFTER* : exécution du corps du trigger *après le déclenchement de l'événement DML* sur une table.

Trigger DML : Evénement

➢ Event_1 [**OR** Event_2 **OR** Event_3]

– Cette clause détermine **l'événement déclencheur** : *quelle instruction LMD entraine l'exécution du déclencheur ?*

⇒ Les événements possibles sont : INSERT, UPDATE et DELETE.

– Plusieurs événements déclencheur peuvent être utilisés ensemble séparés par OR => le déclencheur est déclenché à tout événement déclencheur spécifié.

– ***Pour l'événement UPDATE,*** on peut spécifier les attributs concernés en mettant **UPDATE OF nom-attribut1, ...**

Trigger DML : Type « ligne »

➢ **[FOR EACH ROW]**

– Cette clause est utilisée pour déterminer **si** un déclencheur est de type ligne (*clause omise dans le cas de trigger global*).

– Avec le type de déclencheur ligne, on peut avoir accès (suivant l'opération LMD) à **l'ancienne** et/ou à la **nouvelle** donnée affectant la table.

Trigger DML : Type « ligne »

➢ **[FOR EACH ROW]** *(suite)*

– Pour un **INSERT**, toutes les colonnes de la ligne insérée sont accessibles. Elles se nomment **:new.<nom_colonne>**.

– Pour un **DELETE**, toutes les colonnes de la ligne supprimée sont accessibles. Elles se nomment **:old.<nom_colonne>**.

– Pour un **UPDATE**, toutes les colonnes de la ligne supprimée et insérées sont accessibles. Elles se nomment:

:new.<nom_colonne> **:old.<nom_colonne>**

Trigger DML : Type « ligne »

- **[FOR EACH ROW]** *(suite)*

  Lorsque le même déclencheur peut être exécuté à partir d'opérations LMD différentes, il est possible de tester dans le programme l'événement déclencheur avec les prédicats:

  - If *inserting* then
  - If *deleting* then
  - if *updating* then
  - If *updating*[*('<colonne>')*] then

Trigger DML : Type « ligne »

- **Exemple :** Déclencheur ligne

```
CREATE OR REPLACE TRIGGER SONDAGE_TRIG1
AFTER INSERT OR UPDATE ON SONDAGE
FOR EACH ROW  --Trigger Ligne
BEGIN
    IF inserting THEN
      INSERT  INTO sondage_copie VALUES ( :new.num,
    :new.date_naissance, :new.reponse1, :new.reponse2, :new.val );
    END IF ;
END;
```

Trigger DML : Type « ligne »

⇒ **Remarque :**

Lorsqu'un déclencheur s'exécute pour un update sur quelques colonnes, les attributs non utilisés prennent dans le programme les anciennes valeurs des enregistrements concernés : **new = old**

Trigger DML : Type « ligne »

➢ **[REFERENCEMENT OLD AS** o **NEW AS** n] :

Cette clause permet de renommer les noms de référence **new** et **old** à des autres noms définis par l'utilisateur.

Exemple :

```
CREATE TRIGGER restrict_salaire  BEFORE UPDATE OF salaire
ON employees FOR EACH ROW REFERENCING NEW AS nv
BEGIN
IF :nv.salaire >18000 Then
    Raise_application_error(-20010, 'Vous ne pouvez pas modifier le
    salaire de cet employé'); End if;
END;
```

Trigger DML : Type « ligne »

- **WHEN**(condition) :
  - Cette clause est **valable** uniquement pour les déclencheurs ligne.
  - Elle permet de restreindre *l'exécution du déclencheur* :
    - Si l'expression WHEN n'est pas vérifiée le déclencheur ne s'exécute pas.
    - Sinon, le déclencheur est lancé seulement pour les lignes qui satisferont la condition spécifiée.

Trigger DML : Type « ligne »

- **WHEN**(condition) : *Exemple*

```
CREATE OR REPLACE TRIGGER emp_salaire
BEFORE  INSERT OR UPDATE OF salaire ON employees
FOR EACH ROW
WHEN(:new.dpt = 11)
BEGIN
      IF UPDATING('salaire') THEN
          IF :new.salaire < :old.salaire THEN
              Raise_application_error(-20001, 'Attention, Diminution de salaire');
          END IF;
      END IF;
END;
```

Trigger DML : Type « global »

– Un **trigger DML global** n‘as pas accès aux valeurs mises à jours (:new et :old) par l'opération puisqu'ils se déclenchent une seule fois même si la requête LMD met à jour plusieurs lignes.

– Il présente l'avantage de pouvoir effectuer des manipulations sur la table qui a déclenché le trigger.

Trigger DML : Type « global »

Exemple : rendre les données disponibles uniquement dans les horaires de travail, sinon une exception est levée

```
CREATE OR REPLACE TRIGGER securiser_modif_employees
BEFORE INSERT OR UPDATE OR DELETE ON employees
BEGIN
   IF ( to_char(sysdate,‘DY') IN (‘SAT’,’SAN’) )  OR
      ( to_char(sysdate,'hh24:mi') NOT BETWEEN '8h30' AND '18h30' )
THEN
    RAISE_APPLICATION_ERROR(-20001, 'vous n avez pas le droit de
manipuler les données des employées hors horaire de travail !!!');
END;
```

Trigger DDL : Définition & Syntaxe

Les ordres LDD pouvant provoquer l'exécution du déclencheur sont : CREATE, ALTER, DROP, GRANT, RENAME, REVOKE.

⇒ Ils réagissent aux modifications de la structure de la base de données.

CREATE [or REPLACE] **TRIGGER** <nom_trigger>
{**BEFORE** | **AFTER**} Évnt1 [**OR** Évnt2 **OR** …]
ON {[nomSchema] **SCHEMA** | **DATABASE**}
Trigger_body -- *bloc PL/SQL*

Trigger DDL : Syntaxe (description)

– Ils sont sensibles aux options BEFORE et AFTER.

– la directive DATABASE : précise que le déclencheur peut s'exécuter à partir d'un événement provoqué par n'importe quel schéma.

– la directive SCHEMA : précise que le déclencheur peut s'exécuter à partir d'un événement provoqué par le schéma lui-même.

Trigger d'Instance : Définition

– Des événements systèmes peuvent provoquer le déclenchement d'un code PL/SQL.

– Des événements comme :

⇒ LOGON, STARTUP, SERVERRROR, SUSPEND utilisent l'option **AFTER**.

⇒ LOGOFF, SHUTDOWN utilisent l'option **BEFORE**.

Remarque : AFTER STARUP et BEFORE SHUTDOWN s'appliquent avec des déclencheurs de type **DATABASE**.

Trigger d'Instance : Exemple

```
CREATE TRIGGER deconnexion
BEFORE LOGOFF ON DATABASE
BEGIN
    INSERT INTO trace VALUES (user, sysdate);
END;
```

Gérer les Triggers

– Désactiver ou réactiver un déclencheur :

```
ALTER TRIGGER trigger_name {ENABLE| DISABLE};
```

– Désactiver ou réactiver tous les déclencheurs d'une table :

```
ALTER TRIGGER table_name {ENABLE| DISABLE}
ALL TRIGGERS;
```

– Détruire un déclencheur :

```
DROP TRIGGER trigger_name ;
```

The End