Groupement de procédure et packages

1. Structure d'un package
2. Développement d'un package
3. Utilisation d'un package
4. Privilèges requis pour la gestion et l'utilisation d'un package
5. Package STANDARD

- Les Packages
  - Équivaut à la notion de librairie
  - Permettent de stocker des modules logiques de traitement dans une BD en regroupant ;
    - des procédure,
    - des fonctions,
    - des variables,
    - des constantes
    - des types (Record, ..)
    - des curseurs
    - des exceptions
    - …

1. Structure d'un package

- Les packages peuvent contenir deux types de procedures(fonctions) :
  - Publiques (Public) : peuvent être appelées depuis l'extérieur du package
  - Privées (private) : sont invisible de l'extérieur et accessible uniquement par d'autres procédures du même package.

- Deux parties dans un package :
  - La partie spécification :
    - déclaration des types, variables, constants, exceptions, curseurs et sous-programmes *de type publiques*.
  - La partie corps (*body*) :
    - Implémentation de la spécification : la définition des curseurs et sous-programmes déclarés dans la partie spécification.

2. Développement d'un package

2.1. Création d'un package

– Chaque partie du package doit être créée et compilée séparément.

2.1.1. Partie spécification : contient les déclarations publiques

Create [or replace] **Package** *[schéma.]nom_package*
[IS|AS]
{ [*déclarations de variables;*] | [*déclarations de curseur ;*]
| [*déclarations d'exception;*] |[*déclarations de procédure;*]
| [*déclarations de fonction;]* … }
End [nom_package] ;
/

Note) il faut déclarer les sous-programmes à la fin de la partie spécification.

2.1.2. Partie corps : contient

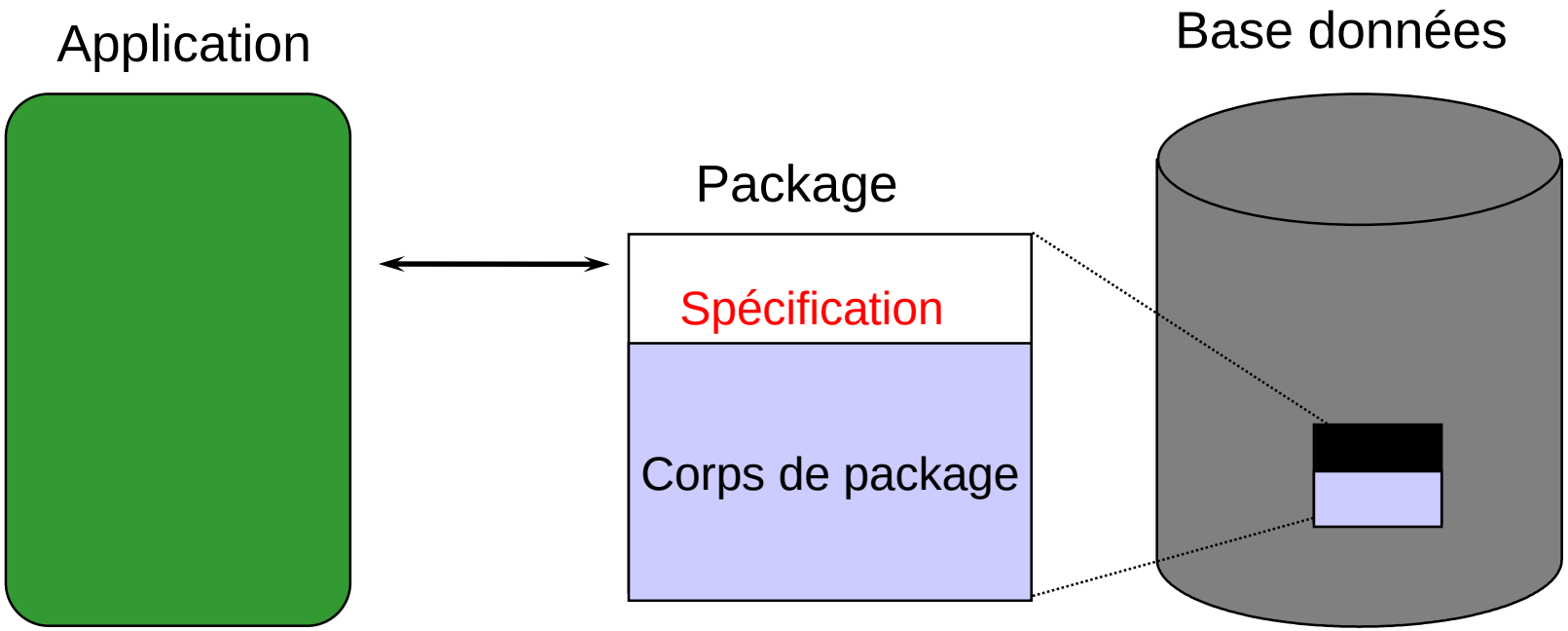
- implémentation de la spécification
- déclarations privées qui sont invisibles à partir de l'extérieur.

Create [or replace] **Package Body** *[schéma.]nom_package*
[IS|AS]
{[déclaration des variables, exception, …] |
[*définition de procédure;*] | [*définition de fonction;*] … }
[Begin
sequence_of_statements]
End [nom_package] ;

Initialisation d'un package

Note) Dans le corps du package, l'ordre de définition doit être tel que les éléments référencés par une autre élément doivent être définis avant lui.

```
Create  or replace Package pilote_work AS
        Procedure del_pilote(x_nopilot  pilote.nopilot%type);
        Function moy_h_vol (xcodetype avion.type%type) Return number;
        Procedure valid_pilote (x_pilote pilote.nopilot%type) ;  err_comm Exception ;
End pilote_work ;
/
Create or replace Package Body pilote_work AS
        Procedure del_pilote(x_nopilot  pilote.nopilot%type) Is
        Begin
              Delete from Pilote where nopilot = x_nopilot ;
        End del_pilote;
        Function moy_h_vol (x_codetype  avion.type%type)  RETURN number  IS
              nbhvol_avg NUMBER(8,2) := 0  /* sert à transmettre la valeur résultat */
        Begin
           Select Avg(nbhvol) Into nbhvol_avg From Avion Where type = x_codetype;
           Return (nbhvol_avg);
        End moy_h_vol ;
        Procedure valid_pilote (x_pilote pilote.nopilot%type)  Is
              dif  pilote.sal%type := 0 ;
        Begin
           Select sal – NVL(comm, 0)  Into dif From Pilote Where nopilot = x_nopilot;
           If dif < 0 Then  RAISE err_comm  End If;
              Exception When err_comm Then
                    RAISE_APPLICATION_ERROR(-20001, 'commission supérieure au salaire') ;
        End valid_pilote;
End pilote_work ;
/
```



•Spécification = Interface , Corps = Boite noire

• Seules les déclarations dans la spécification du package sont visibles et accessibles aux applications. L'implémentation détaillée dans le corps est cachée.

• Changement du corps possible sans avoir à recompiler les programmes appelants si l'interface n'est pas touchée.

Packages de curseurs

- On sépare la spécification d'un curseur de son corps (définition). On peut changer le corps du curseur sans changer sa spécification.
  - Spécifier le curseur et son attribut de retour (%Type, %Rowtype) dans la partie de spécification du package.

    CURSOR cursor_name [(parameter[, parameter]...)] RETURN return_type;
  - Définir le corps du curseur dans la partie Body.
  - Ex)

```
CREATE PACKAGE emp_actions AS
Type temp  IS record (n emp.ename%type, s emp.sal%type);
CURSOR c1 RETURN temp ; -- declare cursor specification
...
END emp_actions;
/
CREATE PACKAGE BODY emp_actions AS
CURSOR c1 RETURN temp IS   -- define cursor body
   SELECT ename,sal FROM emp WHERE sal > 3000;
...
END emp_actions;
```

- Référencer un curseur de package dans un bloc PL/SQL :

```
DECLARE
      emp_rec emp%ROWTYPE;
      ...
BEGIN
      ...
      OPEN emp_actions.c1;
      LOOP FETCH emp_ actions.c1 INTO emp_rec;
      EXIT WHEN emp_ actions.c1%NOTFOUND;
      ...
      END LOOP;
      CLOSE emp_actions.c1;
END;
```

Note) Lorsque l'on ouvre un curseur de package, il reste ouvert jusqu'à on le ferme ou on se déconnecte de la session Oracle.

2.2. Initialisation d'un package

- A la fin du package, on peut inclure un block spécifique qui n'est exécuté qu'une seule fois, au premier appel d'un composant du package dans une session.
- Ex) initialiser une variable globale qui contient le nombre moyen d'heure de vol de tous les avions de la base ;

```
Create Package pilote_work AS
G_moy_hvol numer := 0 ;
End pilote_work;
/

Create Package Body pilote_work AS
Function moy_h_vol (x_codetype  avion.type%type)  RETURN number  IS
…
End moy_h_vol ;
/* section d'initialisation */
Begin
     Select Avg (nbhvol) Into g_moy_hvol From Avion ;
End pilote_work ;
/
```

Exemple d'un package

```
CREATE PACKAGE emp_actions AS
      /* Declare externally visible types, variable, cursor, exception. */
        commission Number := 0;
        TYPE EmpRecTyp IS RECORD (emp_id  emp.empno%type, salary emp.sal%type);
        CURSOR desc_salary RETURN EmpRecTyp;

        /* Declare externally callable subprograms. */
        FUNCTIONE hire_employee (ename VARCHAR2,job VARCHAR2,mgr REAL, sal
            REAL,comm REAL,deptno REAL) RETURN INT;

        PROCEDURE fire_employee (emp_id INT);
END emp_actions;
/

CREATE PACKAGE BODY emp_actions AS
        number_hired INT; -- visible only in this package

        /* Fully define cursor specified in package. */
        CURSOR desc_salary RETURN EmpRecTyp IS SELECT empno, sal
                                        FROM emp ORDER BY sal DESC;
```

```
/* Fully define subprograms specified in package. */
FUNCTION hire_employee (ename VARCHAR2,job VARCHAR2,mgr REAL,
                                    sal REAL, comm REAL,deptno REAL) RETURN INT IS
    new_empno INT;
BEGIN
    SELECT empno_seq.NEXTVAL INTO new_empno FROM dual;
    INSERT INTO emp VALUES (new_empno, ename, job,
                                                mgr, SYSDATE, sal, comm, deptno);
    number_hired := number_hired + 1;
    RETURN new_empno;
END hire_employee;

PROCEDURE fire_employee (emp_id INT) IS
BEGIN
    DELETE FROM emp WHERE empno = emp_id;
END fire_employee;

BEGIN -- initialization part starts here
    INSERT INTO emp_audit VALUES (SYSDATE, USER, 'EMP_ACTIONS');
    number_hired := 0;
END emp_actions;
/
```

2.3. Surcharge d'une procédure ou d'une fonction

- Dans un package, il est possible de définir plusieurs procédures ou fonctions avec le même nom mais avec des paramètres de type différentes => Surcharger
- Utile quand on veut un sous-programme avec des paramètres similaires mais qui ont de type différentes.

```
CREATE PACKAGE journal_entries AS
    PROCEDURE journalize (amount REAL, trans_date VARCHAR2);
    PROCEDURE journalize (amount REAL, trans_date INT);
END journal_entries;
CREATE PACKAGE BODY journal_entries AS
    PROCEDURE journalize (amount REAL, trans_date VARCHAR2) IS
    BEGIN
      INSERT INTO journal VALUES (amount, TO_DATE(trans_date, 'DD-
                                          MON-YYYY'));
    END journalize;
    PROCEDURE journalize (amount REAL, trans_date INT) IS
    BEGIN
      INSERT INTO journal VALUES (amount, TO_DATE(trans_date, 'J'));
    END journalize;
END journal_entries;
```

2.4. Un package sans corps

- Seuls les sous-programmes et curseurs ont une implémentation dans le corps du package. Donc, un package sans corps est possible.
- Ex)

```
CREATE PACKAGE trans_data AS  --  bodiless package
    TYPE TimeRec IS RECORD (minutes SMALLINT, hours SMALLINT);
    TYPE TransRec IS RECORD ( category VARCHAR2, account INT,
                              amount REAL, time_of TimeRec);
    minimum_balance CONSTANT REAL := 10.00;
    number_processed INT;
    insufficient_funds EXCEPTION;
END trans_data;
 /
```

- Ce genre de packages permet de définir des variables globales qui persistent durant une session.

3. Utilisation d'un package

- Référence à un élément d'un package
  - Possible de faire référence à tous les éléments déclarés dans la partie spécification d'un package.
  - Ex)
    - Execute emp_actions.commission := 500 ;
    - Execute emp_actions. fire_employee (100) ;

Persistance des données

- Tout élément (variable, constante ou curseur) de type privé et local est créé au moment de l'appel et supprimé à la fin de l'exécution de la procédure.

- Tout élément de type public (déclaré dans la spécification du package) et tout élément déclaré dans le corps mais sans être associée à une procédure ou fonction, sont créés lors du premier appel et reste accessible jusqu'à la fin de la session.

  → Persistance des données

- Note) Lorsqu'un package est simultanément utilisé par plusieurs sessions, chaque session utilise sa propre copie des variables et des curseurs.

Avantages des packages

- encapsulation de procédures, variables et types de données
- déclaration de procédures “public” et “private” :
- définition des variables persistantes :
  - Les variables et les curseurs publiques persistent pendant la durée d'une session . Elles peuvent être partagée (variables globales) par tous les autres programmes qui s'exécutent dans l'environnement.
- amélioration des performances : un sous-programme packagé est chargé en mémoire lors de son premier appel. Pour tous les autre appels, pas besoin d'accès aux disk (disk I/O)

4. Privilèges requis

1. Création

- Pour créer un objet procédural (procédure, fonction, package) dans son propre schéma, un utilisateur doit posséder le privilège :

  CREATE PROCEDURE

- Pour créer un objet procédural dans n'importe quel schéma, un utilisateur doit posséder le privilège :

  CREATE ANY PROCEDURE

2. Utilisation

- Tout utilisateur peut employer sans restriction les objets procéduraux dont il est le créateur.
- Pour utiliser un objet procédural appartenant à un autre utilisateur, il faut avoir reçu
  - soit le privilège EXCUTE sur l'objet
  - soit le privilège système EXECUTE ANY PROCEDURE

5. Package STANDARD

- PL/SQL fournit divers outils dans un package STANDARD
  - inclut des fonctions internes et des exceptions internes
  - Ex) Function ABS(n NUMBER) RETURN NUMBER
- Fonction internes dans le package STANDARD
  - Fonctions de chaînes : ASCII, CHR, LENGTH, LOWER, LPAD, LTRIM, SUBST, UPPER, ..
  - Fonctions numériques : ABS, MOD, SIN, TRUNC,…
  - Fonctions de conversion : CHARTOROWID, TO_CHAR, TO_DATE, TO_NUMBER, …
  - Fonctions de date : ADD_MONTHS, LAST_DAY, NEXT_DAY, …
  - Fonctions diverses : NVL, USER, …
- Select length(ename) from emp ;
- Select avg(sal) ..
- Pas besoin de faire STANDARD.ABS(n)

- Beaucoup de surcharge
  - To_char (a number)
  - To_char (a date, b format)
  - …

  - PL/SQL résout l'ambiguïté en comparant les nombres et les types de paramètres.

- Il est possible de définir un objet procédural utilisateur ayant le même nom qu'une fonction interne de STANDARD.

  → La définition locale prends le pas sur la définition interne de la fonction. On peut accéder à la fonction interne par :
    -

- Note) les procédures du package DBMS_OUTPUT
  - PUT : affiche le résultat sur une seul ligne
  - PUT_LINE

  Dbms_output.put_line(…)

Triggers (déclencheurs)

1. Structure d'un package
2. Développement d'un package
3. Utilisation d'un package
4. Privilèges requis pour la gestion et l'utilisation d'un package
5. Package STANDARD

Caractéristiques d'un déclencher

- Déclencher (trigger) : traitement déclenché par un événement.
- Déclencheur associé aux commandes
  - LMD (Insert, Delete, Update)
  - Depuis Oracle 8i
    - LDD (Create, Alter, Drop)
    - Opérations de la base (Servererror, logon, logoff, startup, shutdown)

- Caractéristiques d'un déclencheur
  - Un programme PL/SQL stocké
  - Associé à une et une seul table
  - ORACLE exécute le déclencheur automatiquement quand une opération SQL affecte la table.
  - Actif ou inactif

Description d'un déclencheur

- **Modèle : Événement-Condition- Action**

  – *Événement* est une action de mise à jour sur la table, INSERT, UPDATE ou DELETE (LMD sur une table).

  – *Condition* permet de déclencher l'action du trigger dans le cas où elle est vérifiée.

  – *Action* permet de réaliser une ou plusieurs opérations sur la base . Un bloc PL/SQL qui peut faire appel à des sous-programmes stockés en PL/SQL ou en Java.

Exemple : Application bancaire

"Tout compte passant en dessous d'un certain seuil doit faire l'objet d'une notification à la personne titulaire du compte"

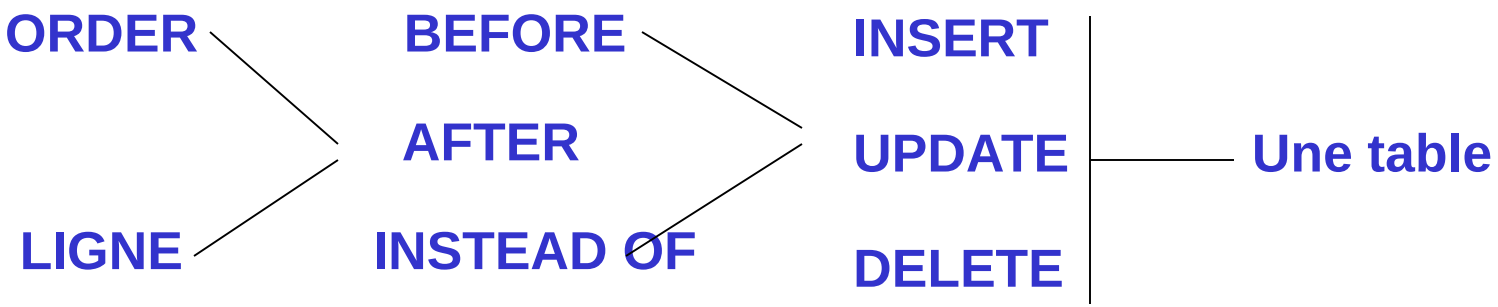
ÉVENEMENT Update CompteB

CONDITION

Propriétaire(Personne,CompteB) AND (Reste (CompteB, Solde) < Seuil (CompteB))

ACTION Notifier(Personne, CompteB)

Structure d'un déclencheur



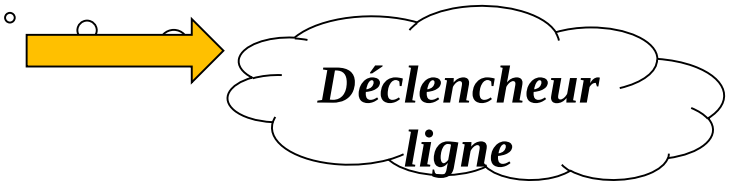
- BEFORE (AFTER) : L'exécution du traitement associé au déclencheur peut se dérouler avant (après) la prise en compte de l'évènement.
- INSTEAD OF s'exécute "à la place" de l'action qui les a déclenchés.

- Le niveau d'exécution d'un déclencheur
  - Déclencheur par ordre : s'exécute une seule fois pour l'ensemble des lignes concernées par l'événement.
  - Déclencheur ligne : s'exécute une seule fois pour chaque ligne.

Création d'un déclencheur

- Déclencheur créé sur un événement LMD :

```
Create [or replace] trigger [utilisateur.]nom_déclencheur
    { before | after | instead of }
    { delete | insert | update [of colonne [, colonne] … ] }
      [ or  { delete | insert | update [of colonne [, colonne] … ] … }
    On [utilisateur].{ Table |  Vue }
    [ Referencing { [OLD [AS] ancien | NEW [AS] nouveau ] } ]
    For each row
    Bloc_PL/SQL
```



Un seul déclencheur sur plusieurs événements

- Un même déclencheur peut répondre à plusieurs événements.

```
CREATE TRIGGER ..
BEFORE INSERT OR UPDATE Of Salaire OR DELETE
   ON employé
BEGIN
    IF INSERTING THEN .. END IF;
    IF DELETING THEN .. END IF;
    IF UPDATING (Salaire) THEN  .. END IF;
```

Déclencheur par ordre

Ex1) Limiter le droit de supprimer un pilote à l'utilisateur Dupont.

```
Create or replace Trigger del_pilote
Before delete On Pilote
Begin
      If USER != ‘DUPONT’ then
         Raise_application_error (-20001, ‘utilisateur non autorisé’);
      End if;
End ;
/
```

EX2) SQL> delete from pilote where no<10 ;

- Déclencheur par ordre :
- Déclencheur par ligne :

Déclencheur ligne

- Un déclencher linge est
  - exécuté pour chacune des lignes concernées par l'exécution de l'événement.
  - créé à l'aide de la clause *for each row* dans la comme create trigger.
- Dans la syntaxe
  - Condition : ne contient pas de requête SQL. Utilisé pour restreindre l'action du déclencheur à certaines lignes de la table.
  - Referencing : changer l'indicatif de OLD ou NEW.
- On peut référencer à la valeur d'une colonne avant (après) modification avec OLD (NEW), dans la clause When ou dans le corps du traitement.

Ordre SQL	OLD	NEW
INSERT	NULL	Valeur créée
DELETE	Valeur avant suppression	NULL
UPDATE	Valeur avant modification	Valeur après modification

Emp	
No	sal
11	1200
12	2100

```
Create or replace trigger emp_sal
before update of sal on emp
For each row
Begin
    Dbms_output.put_line( :old.no || ‘ancien salaire  : ‘ || :old.sal ||  ‘nouveau salaire :  ‘ || :new.sal);

  :new.no := :old.no + 10 ;

End;
/
```

SQL) update emp set sal = sal + 100 ;

Exemple : déclencheur ligne

```
CREATE TRIGGER vérification_salaire
Before Insert Or Update of Sal, job On emp
For each row When (new.job != 'PRESIDENT')  -- attention !! Non :new dans WHEN
Declare
      Sal_min Number; Sal_max Number;
Begin
      Select lsal, hsal into sal_min, sal_max from SalIntervalles
                                              where Job = :new.job ;
      if (:new.sal < sal_min OR :new.sal > sal_max) Then
          Raise_application_error (-20230, 'salaire hors norme');
      Elseif (:new.sal < :old.sal) Then
          Raise_application_error (-20231, 'salaire négative');
      Elseif (:new.sal  > :old.sal * 1.1 ) Then
          Raise_application_error (-20232, 'augmentation supérieur à 10 %');
      End if;
      Exception WHEN no_data_found  THEN
          Raise_application_error (-20233, 'Invalide Job ');
End ;
```

Exercices

1) Soit deux tables

-Audit_pilote(date, user, opération, old_nopilote, old_nom, old_adresse, old_sal)

note) opération : 'delete' ou 'update'

-pilote (nopilote, nom, adresse, sal)

Créer un déclencheur qui garde dans la table Audit_pilote un historique des lignes manipulées (delete ou update) à chaque modification de la table, en sauvegardant les valeurs des lignes avant le modification.

2) Créer un déclencheur qui interdit toute modification (insert, delete, update) d'une ligne de la table pilote.

Déclencheur Instead Of

- Utile pour modifier des vues qui ne peuvent pas l'être directement par insert, update ou delete. Utilisé seulement sur les vues.

Ex) Jointure de deux tables

- Emp_tab (Empno,Ename,Job,Hiredate,Sal...)
- Dept_tab (Deptno, Loc, #Mgr_no)

- Vue Dept_Manager ;

```
Create view Dept_Manager as SELECT deptno , mgr_no, ename
                    FROM Emp_tab , Dept_tab  WHERE empno = mgr_no;
```

Ex) Insert into Dept_Manager values (10, 20, 'Tom');

deptno	mgr_no	ename
10	20	Tom

Si mgr_no (=emp_no) n'existe pas dans la table Emp_Tab → ajouter le.

```
CREATE OR REPLACE TRIGGER Dept_Manager_insert
INSTEAD OF INSERT ON Dept_Manager
REFERENCING NEW AS n -- new Dept_Manager information
FOR EACH ROW
DECLARE
rowcnt number;    existedept Exception;
BEGIN
      SELECT COUNT(*) INTO rowcnt FROM Dept_tab WHERE deptno = :n.deptno;
      IF rowcnt = 0 THEN
          INSERT INTO Dept_tab (deptno) VALUES(:n.deptno);
      ELSE Raise existedept ;
      END IF ;

      SELECT COUNT(*) INTO rowcnt FROM Emp_tab WHERE empno = :n.mgr_no;
      IF rowcnt = 0 THEN
          INSERT INTO Emp_tab (empno,ename) VALUES (:n.mgr_no, :n.ename);
      END IF;

      UPDATE Dept_tab Set mgr_no = :n.mgr_no ;

      Exception When existedept Then
          dbms_output.put_line(' dept ' ||:n.deptno || 'existe déjà' );
END;
```

Exercices

Emp (nom, age, #loge_nom) ;

Logement(loge_nom, manager, adresse) ;

Create view Emp_Loge_Manager as Select nom, e.loge_nom, l.manager
from emp e, logement l where emp.loge_nom = l.loge_nom ;

- Créer un déclencher qui réalise la mis à jour de cette vue comme par exemples :

  Update emp_log_manager set nom = xxx where … ;
  Update emp_log_manager set loge_nom = xxx where … ;
  Update emp_log_manager set manger = xxx where … ;

Gestion des déclencheurs

- Le code source du déclencheur est stocké dans la base, pas le code exécutable => recompiler le source à chaque appel.

- Privilèges pour créer un déclencheur :
  - CREATE TRIGGER
  - ALTER TABLE sur la table concernée

- Informations sur les déclencheurs
  - Les vues User_Triggers, All_Triggers, DBA_Triggers

- Suppression d'un déclencheur
  - DROP TRIGGER nom_déclencheur

- Activation et désactivation d'un déclencheur
  - Désactivation [Activation]
    - ALTER TRIGGER nom_déclencheur DISABLE [ENABLE] ;
    - ALTER TABLE nom_table DISABLE [ENABLE] ALL TRIGGERS ;

- Note) un déclencheur ne peut pas consulter les données d'une table (Select ..) concernée par l'événement du déclencheur.

Ex) Create trigger verif_hbhvol After update on avion for each row

Begin

Select min(nbvol) into … from avion ; …

Les déclencheurs sont fréquemment utilisés pour :

- générer automatiquement des valeurs dérivées des colonnes ;
- prévenir les transactions non valides ;
- renforcer l'autorisation complexe de sécurité ;
- compléter l'intégrité référentielle associé à la déclaration des tables ;

  Ex) tables avion, vol, affectation

  => un déclencher peut être défini pour vérifier, lors de chaque affectation d'un avion à un vol, que l'avion n'est pas déjà requis par une autre affectation pendant la durée du vol.
- implémenter les règles de gestion complexes ;
- maintenir les répliquants synchrones des tables ;
- collecter des statistiques sur l'accès aux tables ;
- permettre de construire des vues complexes qui son modifiables.

Exemple de plusieurs lanages de programmations des SGBD

ORACLE (PL/SQL)

```
CREATE OR REPLACE PROCEDURE raise_sal (sal IN
FLOAT) I

N Integer;

BEGIN

SELECT count(*) into from emp where salaire < sal;

If N <5 then
      update emp set salaire = salaire * 1.1 where salaire < sal
;
       commit;
End if;
END;
/
```

MySQL :

```
Delimiter |
create or replace procedure  raise_sal(IN sal float)
begin
declare  Name varchar(20);
Declare  N INT  ;
Set i=0;

select  count(*) into i from emp where  salaire < sal;

If  N <5 5 then
      update emp set salaire = salaire * 1.1 where  salaire < sal;
          commit;
end if;
End
|
```

PostgreSQL (PL/pgSQL)

```
CREATE OR REPLACE FUNCTION raise sal (sal  integer)  RETURNS void AS $$
DECLARE
N  integer;
BEGIN
SELECT count(*) into from emp where salaire < sal;

If N <5 then
      update emp set salaire = salaire * 1.1 where salaire < sal ;
       commit;
End if;
END; postgreerer
$$ LANGUAGE plpgsql ;
```