Bases de Données

C4-PL/SQL(Fin)

Lina Soualmia

Université de Rouen
LITIS - Équipe TIBS-CISMeF
lina.soualmia@chu-rouen.fr

23 septembre 2015

Manipulation des exceptions en PL/SQL

- Le traitement des exceptions PL/SQL :
  - mécanisme pour manipuler les erreurs rencontrées lors de l'exécution
- Possibilité de continuer l'exécution :
  - si l'erreur n'est pas suffisamment importante pour produire la terminaison de la procédure
- Décision de continuer une procédure après erreur :
  - décision que le développeur doit faire en fonction des erreurs possibles

Types des exceptions

- Déclenchées implicitement
  - Exceptions Oracle prédéfinies
  - Exceptions Oracle non-prédéfinies
- Déclenchées explicitement
  - Exceptions définies par l'utilisateur

Capture des exceptions :

- Syntaxe

```
EXCEPTION
    WHEN exception_1 [OR exception_2...] THEN
        énoncé_1;
        énoncé_2;
    ...
    [WHEN exception_2 [OR exception_4...] THEN
        énoncé_3;
        énoncé_4;
    ...]
    [WHEN OTHERS THEN
        énoncé_5;
        énoncé_6;
    ...]
```


Capture des exceptions prédéfinies :

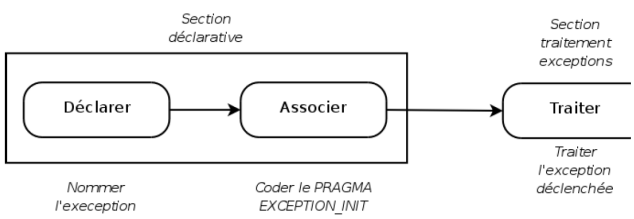
- Faire référence au nom dans la partie traitement des exceptions
- Quelques exceptions prédéfinies :
  - NO_DATA_FOUND
  - TOO_MANY_ROWS
  - INVALID_CURSOR
  - ZERO_DIVIDE
  - DUP_VAL_ON_INDEX

Exceptions prédéfinies

- Exemple

```
...
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        énoncé_1; énoncé_2;
        DBMS_OUTPUT.PUTLINE (TO_CHAR(etudno)||'Non valide');
    WHEN TOO_MANYROWS THEN
        énoncé_3; énoncé_4;
        DBMS_OUTPUT.PUT_LINE ('Données invalides');
    WHEN OTHERS THEN
        énoncé_5; énoncé_6;
        DBMS_OUTPUT.PUT_LINE('Autres erreurs');
```


Capture des exceptions définies par l'utilisateur

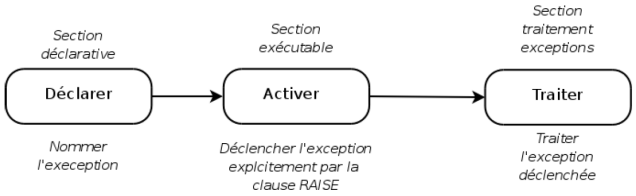


Capture des exceptions non-prédéfinies

- Exemple : Capture de l'erreur n°2291 (violation de la contrainte intégrité).

```
DECLARE
    cont_integrit_err EXCEPTION;
    PRAGMA EXCEPTION_INIT(cont_integrit_err, -2291);
    ...
BEGIN
    ...
    EXCEPTION
        WHEN contr_integrit_err THEN
        DBMS_OUTPUT.PUT_LINE ('violation de contrainte d''intégrité
        ...
END;
```

Capture des exceptions définies par l'utilisateur



Exceptions définies par l'utilisateur

```
DECLARE
    x   NUMBER := ...;
    x_trop_petit EXCEPTION;
    ...
BEGIN
    ...
    IF x < 5 THEN RAISE x_trop_petit;
    END IF ;
    ...
    EXCEPTION
        WHEN x_trop_petit THEN
        DBMS_OUTPUT.PUT_LINE('La valeur de x est trop petite!');
    ...
END;
```


Fonctions pour capturer les exceptions :

- SQLCODE :
  - Retourne la valeur numérique du code de l'erreur
- SQLERRM :
  - Retourne le message associé au numéro de l'erreur

Fonctions pour capturer les exceptions

```
...
v_code_erreur NUMBER;
v_message_erreur VARCHAR2(255);
BEGIN
    ...
    EXCEPTION
        ...
        WHEN OTHERS THEN
            ...
            v_code_erreur := SQLCODE;
            v_message_erreur := SQLERRM;
            INSERT INTO erreurs VALUES (v_code_erreur, v_message_erre
END;
```


Les sous-programmes

- Un sous programme est une séquence d'instructions PL/SQL qui possède un nom
- On distingue deux types de sous programmes :
  - Les procédures
  - Les fonctions

Les sous-programmes

- Une procédure :
  - sous-programme qui ne retourne des résultats que dans ses paramètres
- Une fonction :
  - sous-programme qui retourne des résultats dans :
    - le nom de la fonction
    - les paramètres de la fonction

Les procédures

- Syntaxe

```
DECLARE
    ...
    PROCEDURE <Nom_Proc>[(P1,...,Pn )] IS
        [Déclarations locales]
        BEGIN
            ...
        EXCEPTION
            ...
        END;
BEGIN
    /*Appel de la procédure*/
    ...
    EXCEPTION
    ...
END ;
```


Les procédures

- Syntaxe des paramètres : P1, ...Pn suivent la syntaxe

```
<Nom_Arg> [IN|OUT|IN OUT] <Type_Arg>
```

Où :

- IN : paramètre d'entrée
- OUT : paramètre de sortie
- IN OUT : paramètre d'entrée/sortie
- Par défaut le paramètre est IN

Les procédures

```
DECLARE
    PROCEDURE NouvSal (PNum IN Emp.EmpId%Type, PAug NUMBER )
    BEGIN
        SELECT Sal INTO VSal FROM Emp
        WHERE Emp_Id=PNum;
        UPDATE Emp SET Sal = VSal+PAug WHERE Emp_Id=PNum;
        COMMIT;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Employé inexistant');
    END;
BEGIN
    NouvSal (7550, 500);
    EXCEPTION
        WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE('Erreur');
END ;
```


Les procédures

```
DECLARE
    VErr NUMBER ;
    PROCEDURE NouvSal(PNum Emp.EmpId%TYPE, PAug NUMBER,PErr OUT NUMBER) IS VSal NUM
        BEGIN
            SELECT Sal INTO VSal FROM Emp WHERE EmpId=PNum;
            UPDATE Emp SET Sal = VSal+PAug WHERE EmpId=PNum;
            COMMIT;
            PErr :=0
        EXCEPTION
            WHEN NO_DATA_FOUND THEN PErr:=1;
        END;
```


Les procédures

```
BEGIN
    NouvSal (7550 ,500 ,VErr);
    IF VErr = 0 THEN
        DBMS_OUTPUT.PUT_LINE('Opération effectuée');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Employé inexistant');
    END IF;
EXCEPTION
    WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE('Erreur');
END;
/
```


Les fonctions

```
DECLARE
    [Déclarations globales]
    FUNCTION <Nom_fonc>[(P1,...,Pn )] RETURN Type IS
    [Déclarations locales]
        BEGIN
            ...
            RETURN valeur;
            EXCEPTION
            ...
        END;
BEGIN
    --Appel à la fonction
    ...
    EXCEPTION
    ...
END;
/
```


Les fonctions

```
DECLARE
    VNomComplet VARCHAR2(40);
    FUNCTION NomComplet (PNum Emp.EmpId%TYPE, PErr OUT NUMBER)
    RETURN VARCHAR2 IS
    VNom    Emp.Nom%Type;
    VPrenom     Emp.Prenom%Type ;
        BEGIN
            SELECT Nom, Prenom
            INTO VNom, VPrenom
            WHERE EmpId=PNum;
            PErr :=0;
            RETURN VNom||' '||VPrenom;
            EXCEPTION
                WHEN NO_DATA_FOUND THEN PErr :=1;
                RETURN Null;
        END ;
```


Les fonctions

```
BEGIN
    VNomComplet:= NomComplet(&Num, VErr);
    IF VErr = 0 THEN
        DBMS_OUTPUT.PUT_LINE('Le nom complet est : '||VNomComplet
    ELSE
        DBMS_OUTPUT.PUT_LINE('Employé inexistant');
    END IF;
    EXCEPTION
        WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE('Erreur ');
END ;
```


Le paramètre `IN OUT`

- Paramètre jouant le rôle des deux paramètres `IN` et `OUT`
- Obligatoire de le spécifier

```
SQL> Create or replace procedure affnom(v_nom IN OUT varchar2) IS
  2  BEGIN
  3     v_nom := UPPER (v_nom);
  4  END affnom;
  5  /
Procédure créée.
```


Appel de *affnom* sous SQL*Plus

- Définition d'une variable liée
- Initialisation de la variable

```
SQL> var name varchar2(10);
SQL> begin :name := 'mark' ; end;
  /
Procédure PL/SQL terminée avec succès
SQL> print name
NAME
--------------------------------------------------
mark
```

Appel de *affnom* sous SQL*Plus

- Exécution de la procédure avec un paramètre IN OUT
- Affichage la nouvelle valeur de la variable

```
SQL> execute affnom(:name);
Procédure PL/SQL terminée avec succès.
SQL> print name
NAME
____________________________________________

MARK
```


Passage de paramètres

Il y a plusieurs façons de faire un passage de paramètres :

- Appel de la procédure en spécifiant les paramètres
- Appel de la procédure sans paramètre si ce dernier est un paramètre d'entrée initialisé
- Appel de la procédure en changeant la position des paramètres (il faut spécifier le nom du paramètre)

Passage de paramètres

Exemple :

```
CREATE OR REPLACE PROCEDURE renseign_etud
    (v_nom IN etud.nom%type default 'inconnu' ,v_adr IN etud.adr%type default 'inco
IS
BEGIN
    INSERT INTO etud
    VALUES (etud_etudid.nextval, v_nom, v_adr);
END;
```


Passage de paramètres

```
SQL> begin
  2  renseign_etud('mark', 'paris');
  3  renseign_etud;
  4  renseign_etud(v_adr  => 'lyon');
  5  end;
  6  /

Procédure PL/SQL terminée avec succès.
SQL> select * from etud;

    ETUDID NOM        ADR
__________ _________ _________
         6 mark      paris
         7 inconnu   inconnu
         8 inconnu   lyon
```


Les procédures et les fonctions stockées

- Sont des blocs PL/SQL qui possèdent un nom
- Consistent à ranger le bloc PL/SQL compilé dans la base de données (CREATE)
- Elles peuvent être réutilisées sans être recompilées (EXECUTE)
- Elles peuvent être appelées de n'importe bloc PL/SQL
- Peuvent être regroupées dans un package

Les procédures stockées

```
CREATE [OR REPLACE] PROCEDURE <Nom_Proc>[(P1,...,Pn )]
    [Déclaration des variables locales]
BEGIN
...
EXCEPTION
...
END;
/
```

- Procedure Created / Procédure Créée :
  - La procédure est correcte
- Ou Procedure Created with compilation errors/ Procédure créée avec erreurs de complitation :
  - Corriger les erreurs avec SHOW ERRORS;

Les procédures stockées

```
CREATE PROCEDURE AjoutProd (PrefPro Prod.RefPro%TYPE,.
    PPriUni Prod.PriUni%TYPE,PErr OUT Number) IS
BEGIN
    INSERT INTO Prod VALUES(PrefPro,...,PPriUni);
    COMMIT;
    PErr :=0;
    EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN PErr:=1;
END;
/
```


Appel des procédures stockées

La procédure stockée est appelée par les applications soit :

- En utilisant son nom dans un bloc PL/SQL (autre procédure)
- Par execute dans SQL*Plus

Appel des procédures stockées :

- Dans un bloc PL/SQL :

```
DECLARE
BEGIN
    <Nom_Procedure>[<P1> ,..., <Pn>] ;
END;
```

- Sous SQL*PLUS :

```
EXECUTE <Nom_Procedure>[<P1> ,..., <Pn>] ;
```


Appel des procédures stockées :

```
ACCEPT VRefPro PROMPT 'Référence produit: '
ACCEPT VPriUni PROMPT 'Prix unitaire: '
DECLARE
    VErr NUMBER;
BEGIN
    AjoutProd(&VRefPro,...,&VPriUni,VErr) ;
    IF VErr=0 THEN
        DBMS_OUTPUT.PUT_LINE('Opération effectuée');
    ELSE DBMS_OUTPUT.PUT_LINE('Erreur');
    END IF ;
END ;
/
```


Les fonctions stockées

```
CREATE [OR REPLACE] FUNCTION <Nom_Fonction>[(P1,...,Pn)]
RETURN Type IS [--Déclaration de variables locales]
BEGIN
    --Instruction SQL et PL/SQL
    RETURN(Valeur)
EXCEPTION
    --Traitement des exceptions
END;
/
```

- `Function created/Fonction créée.` :
  - La fonction est correcte
- `Function created with compilation errors/Fonction créée avec erreurs de compilation` :
  - Corriger les erreurs $\Longrightarrow$ `SHOW ERRORS;`


```
CREATE [OR REPLACE] FUNCTION NbEmp(PNumDep Emp.DeptId%Type, PErr OUT NUMBE
RETURN NUMBER IS VNb NUMBER(4);
BEGIN
    SELECT Count(*)
    INTO VNb
    FROM Emp
    WHERE DeptId=PNumDep;
    PErr:=0
    RETURN VNb ;
    Exception
        When No Data Found Then
            PErr:=1;
            Return Null;
END;
/
```


Appel des fonctions stockées

La fonction stockée est appelée par les applications soit :

- Dans une expression dans un bloc PL/SQL
- Par la commande `EXECUTE` (dans SQL*PLUS)

- Dans un bloc PL/SQL :

```
DECLARE
BEGIN
    <var> := <Nom_fonction>[<P1>,...,<Pn>]
END;
```

- Sous SQL*PLUS :

```
EXECUTE :<var> := <Nom_fonction> [<P1>,...,<Pn>];
```


Fonctions stockées

```
ACCEPT VDep PROMPT 'Numéro département :'
DECLARE
    VErr Number ;
    VNb Number(4);
BEGIN
    VNb:=NbEmp(&VDep, VErr);
    IF VErr=0 THEN
        DBMS_Output.Put_Line('Le nombre d''employés est'||VNb);
    ELSE
        DBMS_Output.Put_Line('Erreur');
    END IF;
END;
/
```


Appel des fonctions stockées

Exemple :

```
SQL> VARIABLE VNb
SQL> EXECUTE :VNb:=NbEmp(&VDep, VErr);
Procédure PL/SQL terminée avec succès.
SQL> PRINT VNb

       VNB
----------
       300
```

Suppression des procédures et des fonctions stockées

- Comme tout objet manipulé par Oracle, les procédures et les fonctions peuvent être supprimées si nécessaire

```
DROP PROCEDURE nom_procedure ;
DROP FUNCTION nom_fonction ;
```



```
SQL> DROP PROCEDURE AjoutProd;
Procedure dropped.

SQL> DROP FUNCTION NbEmp;
Function dropped.
```


Compilation

- Oracle recompile automatiquement un sous-programme quand la structure d'un objet dont il dépend a été modifiée
  - Pour une compilation manuelle :
    - `ALTER PROCEDURE | FUNCTION nom COMPILE;`
  - Affichage des erreurs de compilation sous SQL*Plus :
    - `SHOW ERRORS;`

Quelques commandes utiles :

```
SELECT ObjectName, ObjectType FROM obj$;

DESC NomProcedure;

DESC NomFonction;
```


Paquetage (Package)

- C'est un regroupement de variables, curseurs, fonctions, procédures, .. PL/SQL qui fournissent un ensemble cohérent de services
- Distinction entre ce qui est accessible depuis l'extérieur et ce qui n'est accessible qu'à l'intérieur du paquetage → encapsulation
- Structure :
  - Section de spécification :
    - déclarations des variables, curseurs, sous-programmes accessibles depuis l'extérieur
  - Section d'implémentation :
    - code des sous-programmes accessibles depuis l'extérieur + sous-programmes accessibles en interne (privés)

Développement des packages

- Ne peut pas être appelé, ni paramétré, ni imbriqué
- Permet à Oracle de lire plusieurs objets à la fois en mémoire
- Sauvegarder l'énoncé de `CREATE PACKAGE` dans deux fichiers différents (ancienne/dernière version) pour faciliter d'éventuelles modifications
- Le corps du package ne peut pas être compilé s'il n'est pas déclaré (spécifié)

Spécification des packages

La spécification contient la déclaration des curseurs, variables, types, procédures, fonctions et exceptions.

```
CREATE [OR REPLACE] PACKAGE <Nom_Package>
IS [Déclaration des variables et types]
[Déclaration des curseurs]
[Déclaration des procédures et fonctions]
[Déclaration des exceptions]
END[<Nom_Package>];
/
```



```
CREATE OR REPLACE PACKAGE PackProd
IS CURSOR IS SELECT RefPro, DesPro FROM Produit;
PROCEDURE AjoutProd (PRefPro Prod.RefPro%Type,...,PErr OUT NUMBER)
PROCEDURE ModifProd (PRefPro Prod.RefPro%Type,...,PErr OUT NUMBER)
PROCEDURE SuppProd (PrefPro Prod.RefPro%Type,...,PErr OUT NUMBER)
PROCEDURE AffProd;
END PackProd;
/
```

Le corps du package

- On implémente les procédures et fonctions déclarées dans la spécification

```
CREATE [OR REPLACE] PACKAGE BODY <Nom_Package>
IS [Implémentation procédures|fonctions]
END [<Nom_Package>];
/
```



```
CREATE OR REPLACE PACKAGE BODY PackProd IS
    PROCEDURE AjoutProd(PRefPro Prod.RefPro%Type,...,PErr OUT NUM
    IS BEGIN
        INSERT INTO VALUES (PrefPro,...,PPriUni);
        COMMMIT;
        PErr:=0;
        EXCEPTION
            WHEN Dup_Val_On_Index THEN PErr:=1;
            WHEN OTHERS THEN PErr:=1;
    END;
    PROCEDURE ModifProd(PrefPro Prod.RefPro%Type,..,PErr OUT NUMB
    IS BEGIN
    ...
END PackProd;
/
```


Appel des procédures et fonctions

Les procédures et les fonctions définies dans un package sont appelées de la façon suivante :

```
<NomPackage>.<NomProcedure >[(Paramètres)];

Var := <NomPackage>.<NomFonction>[(Paramètres)];
```



```
ACCEPT VRef PROMPT 'Référence produit :';
ACCEPT VPri PROMPT 'Prix : ';
DECLARE
    VErr NUMBER;
BEGIN
    PackProd.ModifProd(&VRef,...,&VPri,VErr);
    IF VErr=0 THEN
        DBMS_Output.Put_Line ('Traitement effectué');
    ELSE
        DBMS_Output.Put_Line ('Erreur');
    END IF;
End;
/
```


Manipulation d'un paquetage

- Re-compilation d'un paquetage :
  - Utiliser `CREATE OR REPLACE PACKAGE` et terminer par '/' une des sections (sous SQL*Plus)
- La modification d'une des sections entraîne la re-compilation automatique de l'autre section
- Affichage des erreurs de compilation avec SQL*Plus :
  - `SHOW ERRORS`

Création le corps du package suivant en mode interactif :

```
SQL> create or replace package body pack1 is
  2 function double_x(x number) return number is
  3 begin
  4 return(2*x);
  5 end;
  6 end;
  7 /
```

Corps de package créé avec des erreurs de compilation.

Pour afficher les erreurs on utilise la commande `SHOW ERRORS`

```
SQL> show errors
Erreurs pour PACKAGE BODY PACK1 :

LINE/COL ERROR
-------- -----------------------------------------------------------------
0/0      PL/SQL: Compilation unit analysis terminated
1/14     PLS-00201: l'identificateur 'PACK1' doit être déclaré
1/14     PLS-00304: impossible de compiler le corps de 'PACK1' sa
         spécification
```


Suppression d'un paquetage

- `DROP BODY nomPaquetage;`
- `DROP nomPaquetage;`

Triggers

- Un trigger est un programme PL/SQL qui s'exécute avant ou après une opération LMD (Insert, Update, Delete)
- Contrairement aux procédures, un trigger est déclenché automatiquement suite a un ordre LMD
- Un déclencheur n'est pas appelé explicitement par une application

Evènement-Condition-Action

- Un trigger est activé par un évènement
  - Insertion, suppression ou modification sur une table
- Si le trigger est activé, une condition est évaluée
  - Prédicat qui doit retourner vrai
- Si la condition est vraie, l'action est exécutée
  - Insertion, suppression ou modification de la base de données

Structure d'un Trigger

1. Description de l'événement déclencheur
2. Éventuelle condition supplémentaire à satisfaire pour déclenchement
3. Description du traitement à réaliser après déclenchement

Composants du trigger - A quel moment se déclenche le trigger ?

- `BEFORE` :
  - le code dans le corps du triggers s'exécute avant les évènements de déclenchement LMD/LDD
- `AFTER` :
  - le code dans le corps du triggers s'exécute après les évènements de déclenchement LMD/LDD
- Instructions LMD : `INSERT, UPDATE, DELETE` et la combinaison des ces opérations
- Instructions LDD : `CREATE, ALTER, DROP`

Composants du trigger - A quel moment se déclenche le trigger ?

- Démarrage ou arrêt de la base
- Connexion ou déconnexion d'utilisateur
- Erreur d'exécution
- Usage fréquent : implémenter les règles de gestion non exprimables par les contraintes au niveau des tables

Le corps du trigger est défini par un bloc PL/SQL anonyme :

```
CREATE [OR REPLACE] TRIGGER <Nom_Trigger>
[BEFORE |AFTER] <Opération_LMD> ON <Nom_Table>
[FOR EACH ROW]
[WHEN <Condition>]
DECLARE
BEGIN
EXCEPTION
END;
/
```


`FOR EACH ROW`

- Avec `FOR EACH ROW` :
  - 1 exécution par ligne concernée par l'instruction LMD (`row trigger`)
- Sans `FOR EACH ROW` :
  - 1 exécution par instruction LMD (`statement trigger`)

Exemple : Création d'un trigger qui remplit la table statistique (Nom Table ⟶ Nb Insert) lors d'une insertion dans la table facture

```
CREATE OR REPLACE TRIGGER StatFacture
AFTER INSERT ON Facture
FOR EACH ROW
DECLARE
    VNbInsert NUMBER;
BEGIN
    SELECT NbInsert
    INTO VNbInsert
    FROM Statistique
    WHERE Nom_Table='Facture';
    UPDATE Statistique
    SET NbInsert=VNbInsert+1
    WHERE Nom_Table='Facture';
```

Les attributs :Old et :New

Ces deux attributs permettent de gérer l'ancienne et la nouvelle valeurs manipulées

- Insert (...) ... ⟶ New
- Delete ...
- Where (...) ... ⟶ Old
- Update ...
- Set (...) ... ⟶ New
- Where(...) ... ⟶ Old

Changement des noms par défaut : REFERENCING

- :OLD désigne un enregistrement à effacer (déclencheur sur DELETE, UPDATE) : REFERENCING OLD AS nomAncien
- :NEW désigne un enregistrement à insérer (déclencheur sur INSERT, UPDATE) : REFERENCING NEW AS nomNouveau
- :PARENT pour des nested tables : REFERENCING PARENT AS nomParent

Exemple :

Etudiant (MatrEtu, Nom,..., CodCla)

Classe(CodCla, NbrEtu)

Trigger mettant à jour la table classe suite à une insertion d'un nouvel étudiant

```
CREATE OR REPLACE TRIGGER MajNbEtud
AFTER INSERT ON Etudiant
FOR EACH ROW
BEGIN
    UPDATE Classe
    SET NbrEtud=NbrEtud+1
    WHERE CodCla =:New.CodCla;
END;
/
```


Base Exemple :

Tables de la base (la clé primaire est soulignée) :

immeuble (Adr, NbEtg, DateConstr, NomGerant)

appart (Adr, Num, Type, Superficie, Etg, NbOccup)

personne (Nom, Age, CodeProf)

occupant (Adr, NumApp, NomOccup, DateArrivee, DateDepart)

propriete (Adr, NomProprietaire, QuotePart)

Exemples de contraintes à satisfaire :

- Intégrité référentielle (clé étrangère ← clé primaire) :
  - lors de la création de la table propriete : CONSTRAINT PropPers FOREIGN KEY (NomProprietaire) REFERENCES personne(Nom)
- Condition entre colonnes :
  - lors de la création de la table occupant : CONSTRAINT dates CHECK (DateArrivee < DateDepart)
- Règles de gestion :
  - somme quotes-parts pour une propriété = 100 ;
  - date construction immeuble < dates arrivée de tous ses occupants... → déclencheurs

Déclencheur sur INSERT

Pour un nouvel occupant, vérifier si occupant.DateArrivee > immeuble.DateConstr (FOR EACH ROW est nécessaire pour avoir accès à :NEW, l'enregistrement ajouté) :

```
CREATE TRIGGER TriggerVerificationDates
BEFORE INSERT ON occupant
FOR EACH ROW
DECLARE Imm immeuble%ROWTYPE;
BEGIN
    SELECT * INTO Imm
    FROM immeuble
    WHERE immeuble.Adr=:NEW.Adr;
    IF :NEW.DateArrivee < Imm.DateConstr THEN
        RAISE_APPLICATION_ERROR(-20100, :NEW.Nom||'arrivé avant construction immeuble'||Imm
    END IF;
END;
```

Évènement déclencheur : INSERT INTO occupant ...VALUES ...;

Déclencheur sur INSERT (2)

Si chaque nouvel immeuble doit avoir au moins un appartement, insérer un appartement après la création de l'immeuble (FOR EACH ROW est nécessaire pour avoir accès à :NEW, l'enregistrement ajouté) :

```
CREATE TRIGGER TriggerAppartInitial
AFTER INSERT ON immeuble
FOR EACH ROW
BEGIN
    INSERT INTO appart (Adr, Num, NbOccup)
    VALUES (:NEW.Adr,1,0);
END;
```

Événement déclencheur : INSERT INTO immeuble VALUES ...;

Déclencheur sur DELETE

Au départ d'un occupant, décrémente appart.NbOccup après effacement de l'occupant (FOR EACH ROW est nécessaire car la suppression peut concerner plusieurs occupants, ainsi que pour avoir accès à :OLD, l'enregistrement éliminé) :

```
CREATE TRIGGER TriggerDiminutionNombreOccupants
AFTER DELETE ON occupant
FOR EACH ROW
BEGIN
    UPDATE appart
    SET NbOccup=NbOccup - 1
    WHERE appart.Adr = :OLD.Adr
    AND appart.Num = :OLD.NumApp;
END;
```

Événement déclencheur : DELETE FROM occupant WHERE ...;

Déclencheur sur UPDATE

En cas de modification d'un occupant, met à jour les valeurs de appart.NbOccup pour 2 les appartements éventuellement concernés (utilise à la fois :OLD et :NEW) :

```
CREATE TRIGGER TriggerMAJNombreOccupants
AFTER UPDATE ON occupant
FOR EACH ROW
BEGIN
    IF :OLD.Adr<>:NEW.Adr OR :OLD.NumApp<>:NEW.NumApp THEN
        UPDATE appart
        SET NbOccup = NbOccup - 1
        WHERE appart.Adr = :OLD.Adr
        AND appart.Num = :OLD.NumApp;
        UPDATE appart
        SET NbOccup = NbOccup + 1
        WHERE appart.Adr = :NEW.Adr
        AND appart.Num = :NEW.NumApp;
    END IF;
END;
```


Les prédicats inserting, updating et deleting

- Inserting :
  - True : Le trigger est déclenché suite a une insertion
  - False : Sinon
- Updating :
  - True : le trigger est déclenché suite à une mise à jour
  - False : sinon
- Deleting :
  - True : le trigger est déclenché suite à une suppression
  - False : sinon


```
CREATE OR REPLACE TRIGGER MajNbEtud
AFTER INSERT OR DELETE ON Etudiant
FOR EACH ROW
BEGIN
    IF Inserting THEN
        UPDATE Classe
        SET NbrEtud = NbrEtud+1
        WHERE CodCla =:New.CodCla;
    End IF;
    IF Deleting THEN
        UPDATE Classe
        SET NbrEtud = NbrEtud-1
        WHERE CodCla=:Old.CodCla;
    END IF;
END;
/
```


Déclencheur sur conditions multiples

Un seul déclencheur pour INSERT, DELETE, UPDATE qui met à jour les valeurs de appart.NbOccup pour le(s) appartement(s) concerné(s) :

```
CREATE TRIGGER TriggerCompletMAJNombreOccupants
AFTER INSERT OR DELETE OR UPDATE ON occupant
FOR EACH ROW
BEGIN
    IF (INSERTING) THEN
    ...
    ELSIF (DELETING) THEN
    ...
    ELSIF (UPDATING) THEN
    ...
    END IF;
END;
```

Exemple d'évènement déclencheur : INSERT INTO occupant

Syntaxe pour déclenchement sur instruction LDD :

```
CREATE [OR REPLACE] TRIGGER nomDeclencheur
BEFORE|AFTER action [OR action ...] ON
{[nomSchema.]SCHEMA | DATABASE}
{[DECLARE ...] BEGIN ... [EXCEPTION ...] END;|CALL nomSousProgramme(listeParame
```

SCHEMA : déclencheur valable pour schéma courant

Quelques actions : CREATE, RENAME, ALTER, DROP sur un objet du dictionnaire

GRANT, REVOKE privilège(s) à un utilisateur

Manipulation d'un déclencheur

Tout déclencheur est actif dès sa compilation

- Re-compilation d'un déclencheur après modification :
  - ALTER TRIGGER nomDeclencheur COMPILE;
- Désactivation de déclencheurs :
  - ALTER TRIGGER nomDeclencheur DISABLE;
  - ALTER TABLE nomTable DISABLE ALL TRIGGERS;
- Réactivation de déclencheurs :
  - ALTER TRIGGER nomDeclencheur ENABLE;
  - ALTER TABLE nomTable ENABLE ALL TRIGGERS;
- Suppression d'un déclencheur :
  - DROP TRIGGER nomDeclencheur;

Séquences : Auto-incrémentation d'une colonne (évite les doublons)

Définition :

```
CREATE SEQUENCE sequenceName
[INCREMENT BY #] [START WITH #]
[MAXVALUE #|NOMAXVALUE]
[MINVALUE #|NOMINVALUE]
[CYCLE|NOCYCLE]
```

- Suppression DROP SEQUENCE sequenceName
- Pseudo-colonne CURRVAL : Valeur courante de la séquence
- Pseudo-colonne NEXTVAL : Incrémentation de la séquence et retourne la nouvelle valeur


```
DROP SEQUENCE SeqAnnotation;
CREATE SEQUENCE SeqAnnotation
    START WITH 1
    INCREMENT BY 1 NO CYCLE
    MAXVALUE  30000
INSERT INTO prof(profnum, profnom, profprenom)
VALUES (SEQ ANNOTATION.NEXTVAL, 'Dupond', 'Miche l')

SELECT SeqAnnotation.CURRVAL FROM dual;
```

SQL dynamique

- Construction dans un programme d'une requête SQL avant de l'exécuter
- Possibilité de création d'un code générique et réutilisable (sinon simple paramétrage de valeurs de remplacement de la clause where)
- `EXECUTE IMMEDIATE chaîne;` chaîne est une commande SQL donnée entre '...'

- Exemple 1 :

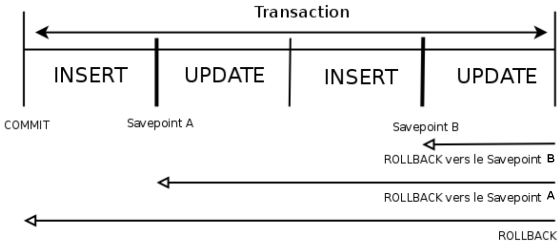
```
BEGIN
    EXECUTE IMMEDIATE 'CREATE TABLE test (col:NUMBER)';
END;
```

- Exemple 2 :

```
DECLARE
    Wreq varchar2(4000);
BEGIN
    Wreq:='CREATE TABLE test(col1:NUMBER)';
    EXECUTE IMMEDIATE Wreq;
END;
```


Contrôle de transactions

- Commandes `COMMIT` et `ROLLBACK`
- Lancer une transaction avec la première commande du LMD à la suite d'un `COMMIT` ou un `ROLLBACK`
- Utiliser le `COMMIT` ou le `ROLLBACK` de SQL pour terminer une transaction



Contrôle de transactions

Déterminer le traitement des transactions pour le bloc PL/SQL suivant :

```
BEGIN
    INSERT INTO temp(numcol1, numcol2, charcol) VALUES (1,1,'ROW 1')
    SAVEPOINT a;
    INSERT INTO temp(numcol1, numcol2, charcol) VALUES (2,2,'ROW 2')
    SAVEPOINT b;
    INSERT INTO temp(numcol1, numcol2, charcol) VALUES (3,3,'ROW 3')
    SAVEPOINT c;
    ROLLBACK TO SAVEPOINT b;
    COMMIT;
END;
```