Le Langage PL/SQL d'Oracle

Gabriel MOPOLO-MOKE

prof. PAST

UNSA

2018/2019

Plan Général

1. INTRODUCTION

SQL

- Est un langage ensembliste et nonprocédural

PL/SQL :

- Est un langage procédural qui intègre des ordres SQL de gestion de la base de données

- Instructions SQL intégrées dansPL/SQL :
  - SELECT
  - INSERT, UPDATE, DELETE
  - COMMIT, ROLLBACK, SAVEPOINT
  - TO_CHAR, TO_DATE, UPPER, ...

2. Structure d'un bloc PL/SQL

. Un bloc PL/SQL est divisé en 3 sections :

[DECLARE

Déclaration de variables, constantes,
exceptions, curseurs]

BEGIN [nom_du_bloc]

Instructions SQL et PL/SQL
(gestion des erreurs)

[EXCEPTION

Traitement des exceptions]

END [nom_du_bloc] ;

. Remarques :

- Les sections DECLARE et EXCEPTION sont facultatives.
- chaque instruction se termine par un « ; »
- Les commentaires :

  -- Sur une ligne
  Ou
  /* Sur plusieurs
  lignes */

2. Structure d'un bloc PL/SQL

. Exemple :

```
REM Code SQLPLUS
PROMPT nom du produit :
ACCEPT prod

REM Code PL/SQL
DECLARE
-- qte NUMBER(5) ;
qte stock.quantite%type;
BEGIN
SELECT quantite INTO qte
FROM STOCK
WHERE produit= '&prod';  -- &prod : transmission de variable sqlplus
IF qte > 0 THEN              -- on contrôle si le stock est positif
    UPDATE stock             -- modification du stock
    SET quantite = quantite -1
    WHERE produit = '&prod';
    INSERT INTO VENTE -- insertion d'une ligne dans la table Vente
    VALUES('&prod'|| 'VENDU', SYSDATE);
ELSE
    INSERT INTO commande  -- Passer commande s'il stock négatif
    VALUES('&prod'|| 'DEMANDE', SYSDATE);
END IF;
COMMIT;
END;
/
```

3. Les variables utilisées dans PL/SQL

. Les différents types de variables locales

- **Les variables locales se déclarent dans la partie DECLARE** du bloc PL/SQL.

- **Différents types de variables :**
  - Variables sur les types prédéfinis ORACLE
  - Variables de type BOOLEAN
  - Variables faisant référence au dictionnaire de données (tables et colonnes)
  - Variables de même types que d'autres variables PL/SQL
  - Variables sur les types Oracle définis par les utilisateurs
  - Variables sur les types définis dans PL/SQL

- **Nota**
  - Les variables peuvent être Initialisées
  - Les variables ont un **degré de visibilité** dans un blocPL/SQL

3. Les variables utilisées dans PL/SQL

. Les variables sur les types prédéfinis d'Oracle

- **Liste des types prédéfinis Oracle**

Type de données	Description	Longueur
char(taille)	taille fixe	1 à 255 bytes
varchar2(size)	taille variable	1 à 2000 bytes
number(p,s)	taille variable	21bytes max
date	taille fixe	7 bytes
LONG	taille Variable	2Go (préférer Clob)
BLOB	taille Variable	4Go à 128 tera bytes
CLOB	taille Variable	4Go à 128 tera bytes
BFILE		
RAW(size)	taille variable	jusqu'à 255 bytes
LONG RAW	taille variable	2Go (préférer Clob)
ROWID	Binaire	6 bytes
MLSLABEL	trusted Oracle	2 à 5 bytes
REF	taille fixe	42 octets

3. Les variables utilisées dans PL/SQL

. Les variables sur les types prédéfinis d'Oracle

- **Syntaxe :**

```
nom_var TYPE_PREDEFINI_ORACLE;
```

- **Exemple :**

```
DECLARE
   nom          VARCHAR2(20);
   prenom       VARCHAR2(15);
   age          NUMBER(3);
   salaire      NUMBER(7,2);
   dateNaiss    date;

BEGIN
...
END;
/
```

3. Les variables utilisées dans PL/SQL

. Les variables sur les types prédéfinis d'Oracle

- **Les différents types Oracle et leurs <u>équivalents ANSI SQL</u>**
  - CHAR(n) équivalent ANSI : CHARACTER(n), CHAR(n)
  - VARCHAR2(n) équivalent ANSI : CHARACTERVARYING(n),CHAR VARYING(n)
  - NCHAR(n) équivalent ANSI : NATIONALCHARACTER(n), NATIONAL CHAR(n)
  - NVARCHAR2(n) équivalent ANSI : NATIONALCHARACTER VARYING(n), NATIONAL CHARVARYING(n), NCHAR VARYING(n)
  - NUMBER(p,s) équivalent ANSI : NUMERIC(p,s), DECIMAL(p,s) (a)
  - NUMBER(38) équivalent ANSI : INTEGER, INT, SMALLINT
  - NUMBER équivalent ANSI : FLOAT (b), DOUBLEPRECISION (c), REAL (d)

3. Les variables utilisées dans PL/SQL

. Les variables de type BOOLEAN

- Deux valeurs possibles TRUE ou FALSE

- **Syntaxe :**

  nom_var **BOOLEAN**;

- **Exemple :**

```
DECLARE
   retour   BOOLEAN;
BEGIN
...
END;
/
```

3. Les variables utilisées dans PL/SQL

Les variables faisant référence au dictionnaire de données Oracle

- **Variable de même type que le type d'un attribut d'une table de la base de données Oracle**

  – **Syntaxe :**

  nom_var
  nomTable.nomColonne%TYPE;

  – **Exemple :**

```
DECLARE
nomPilote        pilote.plnom%TYPE;
BEGIN
...
END;
/
```

3. Les variables utilisées dans PL/SQL

Les variables faisant référence au dictionnaire de données Oracle

- **Variable de même type que le type d'une ligne d'une table de la base de données Oracle**

  – **Syntaxe :**

  nom_var **nomTable%ROWTYPE;**

  – **Exemple :**

```
DECLARE
   lignePilote   pilote%ROWTYPE;
BEGIN
...
END;
/
```

  – Remarque :

  - La structure d'une ligne contient autant de variables que de colonnes de la table. Ces variables portent le même nom que les colonnes de la table et sont de même type. Pour y accéder :

```
nom_var.nom_col1
nom_var.nom_col2
...
nom_var.nom_coln
```

3. Les variables utilisées dans PL/SQL

Les variables de même type qu'une autre variable

- **Variable héritant du type d'une autre variable PL/SQL**

- **Syntaxe :**

  nom_var2 **nom_var1%TYPE;**

- **Exemple :**

```
DECLARE
   ancien_sal      NUMBER(5);
   nouveau_sal     ancien_sal%TYPE; -- NUMBER(5);
BEGIN
...
END;
/
```

3. Les variables utilisées dans PL/SQL

Variables sur les types Oracle définis par les utilisateurs

- Depuis la version 8 d'Oracle, les utilisateurs peuvent définir leurs propres types

- **Syntaxe :**

  nom_var **nomTypeOracleDefiniParUtilisateur;**

- Exemple 1 : Création d'un type sous SQL par l'utilisateur

```
Sql> CREATE OR REPLACE TYPE adresse_t as object(
        numero    number(4),
        rue       varchar2(30),
        ville     varchar2(50)
    );
    /
```

- **Exemple 2 : déclaration d'une variable**

```
DECLARE
   adresse  ADRESSE_T;
BEGIN
...
END;
/
```

3. Les variables utilisées dans PL/SQL

Variables sur les types définis dans PL/SQL

- Il est possible dans un programme PL/SQL de définir des types de données. Ces types de données sont perdus à la fin de l'exécution du programme

- **Syntaxe :**

  nom_var **nomTypeDefiniDansPLSQL;**

- **Exemple : déclaration d'une variable**

```
DECLARE

    TYPE adresse_t IS RECORD(
        numero      number(4),
        rue         varchar2(30),
        ville       varchar2(50)
    );
    adresse         ADRESSE_T;
BEGIN
...
END;
/
```

3. Les variables utilisées dans PL/SQL

Initialisation des variables

- **Avec l'opérateur d'affectation**
  :=
  Ou l'ordre sql SELECT … INTO
  SELECT col1, col2, …coln INTO var1, var2,…varn

- **Exemple :**

```
DECLARE
        var1 VARCHAR2(10) := 'DUPONT';
        var2 NUMBER(5,2) := 100;
        var3 VARCHAR2(10);
        var4 DATE;

        BEGIN
        SELECT col1, col2
        INTO var3, var4     -- Var3 prend la valeur de col1
        FROM ... ;
        ...
END;
```

- **Remarque :**
  - L'ordre SELECT doit ramener une et une seule ligne, sinon erreur ou il faut définir un CURSEUR

3. Les variables utilisées dans PL/SQL

Visibilité des variables

- Une variable est visible dans le bloc où elle a été déclarée et dans les blocs imbriqués si elle n'a pas été redéfinie

- **Exemple**

```
DECLARE
        var1 NUMBER(3);          -- (1)
        var2 VARCHAR2(10);       -- (2)
BEGIN
        var1 :=…                 -- Var1 défini en (1)
        ...
        var2 :=…                 -- Var2 défini en (2)

        DECLARE                  -- bloc imbriqué
            var1 VARCHAR2(10);   -- (3)
            var3 DATE;           -- (4)
        BEGIN
            Var1:=…              -- var1 défini en (3)
            ...
            Var2:=…              -- var2 défini en (2)
            ...
            var3:=…              -- var3 défini en (4)
        END;
            var1 := …            -- var1 défini en (1)
END;
/
```

3. Les variables utilisées dans PL/SQL

Variables de l'environnement extérieur à PL/SQL

- Outre les variables locales vues précédemment, un bloc PL/SQL peut utiliser d'autres variables :
  - Les champs d'écrans FORMS. FORMS est le langage de développement d' écrans graphiques d' Oracle
  - Les variables définies en langage hôte (Pro*C/C++, Pro*Cobol, …) (préfixée de :)
  - Les variables définies dans SQL*Plus (préfixée de &)

4. Les traitements

IF : traitement conditionnel

- **Exécution d'un traitement en fonction d'une condition.**

  IF condition 1 THEN traitement 1;
  [ELSIF condition 2 THEN traitement 2;]
  …
  [ELSIF condition n-1 THEN traitement n-1;]
  [ELSE traitement n;]
  END IF;

- **Les opérateurs utilisés dans les conditions sont les mêmes que dans SQL :**

  =, <, >, >=, <=, !=, IS NULL, LIKE, IS NOT NULL, ...

- **Dès que l'une des conditions est vraie, le traitement qui suit le THEN est exécuté**

- **Si aucune condition n'est pas vraie, c'est le traitement qui suit le ELSE qui est exécuté.**

4. Les traitements

IF : traitement conditionnel

- **Exemple**

```
REM Code SQLPLUS

PROMPT nom du produit :
ACCEPT prod


DECLARE
qte NUMBER(5) ;

BEGIN
SELECT quantite INTO qte
FROM PILOTE
WHERE produit= '&prod';  -- &prod : transmission de variable sqlplus

IF qte > 0 THEN              -- on contrôle si le stock est positif
    UPDATE stock             -- modification du stock
    SET quantite = quantite -1
    WHERE produit = '&prod';
    INSERT INTO VENTE   -- insertion d'une ligne dans la table Vente
    VALUES('&prod'|| 'VENDU', SYSDATE);
ELSE
    INSERT INTO commande  -- Passer commande s'il stock négatif
    VALUES('&prod'|| 'DEMANDE', SYSDATE);
END IF;

COMMIT;

END;
/
```

4. Les traitements

IF : traitement conditionnel

- L'instruction IF-THEN-ELSIF permet d'effectuer des choix entre plusieurs alternatives

- **Exemple**

```
DECLARE
        sales NUMBER(8,2) := 20000;
        bonus NUMBER(6,2);
        emp_id NUMBER(6) := 120;

BEGIN
        IF sales > 50000 THEN
           bonus := 1500;
        ELSIF sales > 35000 THEN
           bonus := 500;
        ELSIF sales>20000 THEN
           bonus:=200;
        ELSE
           bonus := 100;
        END IF;

        UPDATE emp SET sal = sal + bonus
        WHERE empno = emp_id;
END;
/
```

4. Les traitements

CASE : traitement conditionnel

- L'instruction CASE permet d'effectuer des choix alternatives à l'image de IF-THEN-ELSIF-ELSE

- **Exemple**

```
DECLARE
job VARCHAR2(30);
BEGIN
job := 'PDG';
CASE
        WHEN job = 'Ingenieur' THEN
           DBMS_OUTPUT.PUT_LINE('Ingenieur d''études');
        WHEN job = 'PDG' THEN
           DBMS_OUTPUT.PUT_LINE('Président');
        WHEN job = 'Secretaire' THEN
           DBMS_OUTPUT.PUT_LINE('Assistante');
        WHEN job = 'Planton' THEN
           DBMS_OUTPUT.PUT_LINE('Homme à tout faire');
        WHEN job = 'Ouvrier' THEN
           DBMS_OUTPUT.PUT_LINE('Manoeuvre');
        ELSE DBMS_OUTPUT.PUT_LINE('Job inexistant');
END CASE;
END;
/
```

4. Les traitements

Les boucles LOOP : traitement répétitif

- L'instruction LOOP permet d'exécuter plusieurs fois une séquences d'instructions

- Il existe trois formes d'instructions LOOP

  – **La boucle de base LOOP** : boucle perpétuelle, la condition de sortie est exprimée à l'intérieur de la boucle. Le nombre de répétition n'est pas connu à l'avance

  – **La boucle WHILE-LOOP** : la condition de sortie est dans l'expression du WHILE. Le nombre de répétition n'est pas connu à l'avance

  – **La boucle FOR-LOOP** : le nombre de répétitions est connu à l'avance

4. Les traitements

Boucle de base LOOP : traitement répétitif

- **Boucle de base LOOP** : traitement répétitif Exécution d'un traitement plusieurs fois, le nombre n'étant pas connu mais dépendant d'une condition.

```
BEGIN
    [<<label>>]
    LOOP
            instructions;
    END LOOP [label];
END;
```

- **Pour sortir de la boucle, utiliser la clause :**
  EXIT [label] **WHEN** condition

- **Exemple :** insérer les 10 premiers nombres dans la table RESULT.

```
CREATE TABLE result (res       number);
DECLARE
   nb NUMBER := 1;
BEGIN
       LOOP
            INSERT INTO RESULT VALUES (nb);
            nb := nb+ 1;                -- incrémenter le compteur
            EXIT WHEN nb >10; -- Fin de boucle si nb>10
       END LOOP;
END ;
/
```

Boucle de base LOOP : traitement répétitif

- **Un « label »** peut être posé sur une boucle afin de contrôler la sortie surtout en cas de boucles imbriquées

- **Exemple**

```
DECLARE
        s PLS_INTEGER := 0;
        i PLS_INTEGER := 0;
        j PLS_INTEGER;
        BEGIN
        <<outer_loop>>  -- Le label doit précéder la boucle
        LOOP outer_loop
        i := i + 1;
        j := 0;
        <<inner_loop>>
        LOOP inner_loop
        j := j + 1;
        s := s + i * j; -- sum a bunch of products
        EXIT inner_loop WHEN (j > 5);
        EXIT outer_loop WHEN ((i * j) > 15);
        END LOOP inner_loop;
        END LOOP outer_loop;
        DBMS_OUTPUT.PUT_LINE('The sum of products equals: '
           || TO_CHAR(s));
END;
```

4. Les traitements

Boucle FOR : traitement répétitif

- Exécution d'un ensemble « d'instructions » un certain nombre de fois. **Ce nombre étant CONNU**.

```
BEGIN
      FOR indice IN [REVERSE] exp1 .. exp2
      LOOP
         instructions;
      END LOOP;
END;
```

- Remarques :
  - inutile de déclarer « indice »
  - « indice » varie de exp1 à exp2 de 1 en 1
  - si REVERSE est précisé, « indice » varie de exp2 à exp1 avec un pas de -1.

- Exemple : calcul de la factorielle 5

```
DECLARE
fact NUMBER := 1;
BEGIN
      FOR i IN 1 .. 5
      LOOP
         fact := fact *i ;
      END LOOP;
      DBMS_OUTPUT.PUT_LINE('Fact='||fact);
END;
/
```

4. Les traitements

Boucle WHILE : traitement répétitif

- Exécution d'un ensembles « d'instructions » tant que la « condition » reste vraie. Nombre d'exécution inconnu au départ

```
BEGIN
WHILE condition
LOOP
   instructions;
END LOOP;
END;
/
```

- **Exemple** : reste de la division de 5432 par 5

```
DECLARE
   reste NUMBER := 5432;
BEGIN
      WHILE reste >= 5
      LOOP
         reste := reste -5;
      END LOOP;
END;
/
```

5. Les curseurs en PL/SQL

Définitions

- **Un curseur permet d' accéder ligne par ligne à un ensemble de lignes renvoyées par une requête SQL**

- **Il existe 2 types de curseurs :**

  - **CURSEUR IMPLICITE :**

    - **curseur SQL généré et géré par le noyau pour chaque ordre SQL d'un bloc. Utile pour traiter des ordres SQL INSERT, UPDATE, DELETE ou ordre SELECT qui ramène UNE LIGNE au plus**

  - **CURSEUR EXPLICITE :**

    - **curseur SQL généré et géré par l'utilisateur pour traiter un ordre SELECT qui ramène plus d'une ligne.**

Curseur explicite

- **4 étapes pour gérer un curseur :**
  - Déclaration du curseur
  - Ouverture du curseur
  - Traitement des lignes
  - Fermeture du curseur

- **Déclaration du curseur**

  - Un curseur se déclare dans la section de déclaration d'un bloc. DECLARE pour les blocs anonymes. Cette déclration indique le nom du curseur et l'ordre SQL associé

  - **Syntaxe :**
    - **CURSOR nom_curseur IS ordre_select ;**

5. Les curseurs en PL/SQL

Curseur explicite

- **Exemple :**

```
DECLARE
CURSOR pl_nice IS  SELECT pl#, plnom
FROM pilote
WHERE adr=’Nice’;

BEGIN
...
END ;
```

5. Les curseurs en PL/SQL

Curseur explicite

- **Ouverture du curseur**
  - L'ouverture du curseur lance l'exécution de l'ordre SELECT associé au curseur.
  - Ouverture du curseur se fait dans la section **BEGIN** d'un bloc PL/SQL
  - **Syntaxe :**
    - OPEN nom_curseur ;
  - **Exemple :**

```
DECLARE

CURSOR pl_nice IS
SELECT pl#, plnom
FROM pilote
WHERE adr='Nice';

BEGIN
...
OPEN pl_nice;
...
END ;
```

5. Les curseurs en PL/SQL

Curseur explicite

• Extraction des lignes dans le curseur

• Syntaxe :

- FETCH nom_curseur INTO var1, var2, …, varN ;
- Var1=> col1 du Select …

• Exemple :

```
DECLARE
    CURSOR pl_nice IS  SELECT pl#, plnom, sal
    FROM pilote  WHERE adr=’Nice’;

    num pilote.pl#%TYPE;
    nom pilote.plnom%TYPE;
    salaire pilote.sal%TYPE;
    BEGIN
            OPEN pl_nice;
    LOOP
            FETCH pl_nice INTO num, nom,salaire;
            if salaire=500 then
                    salaire:=salaire+200;
                    UPDATE PILOTE SET SAL=SALAIRE
                    WHERE SAL=500 AND adr='Nice';
            end if;
            ...
            EXIT WHEN salaire > 10 000;
    END LOOP;
END ;
```

5. Les curseurs en PL/SQL

Curseur explicite

- **Fermeture du curseur**
  - Pour libérer la mémoire prise par le curseur, il faut lefermer dès qu'on n'en a plus besoin.

- **Syntaxe :**
  - CLOSE nom_curseur ;

- **Exemple :**

```
DECLARE
   CURSOR pl_nice IS  SELECT pl#, plnom, sal
   FROM pilote  WHERE adr='Nice';

   num pilote.pl#%TYPE;
   nom pilote.plnom%TYPE;
   salaire pilote.sal%TYPE;
   BEGIN
            OPEN pl_nice;
   LOOP
            FETCH pl_nice INTO num, nom,salaire;
            ...
            EXIT WHEN salaire > 10 000;
   END LOOP;
   CLOSE pl_nice;
END ;
```

5. Les curseurs en PL/SQL

Les attributs d'un curseur

- **Pour tout curseur (implite ou explicite) il existe des indicateurs sur leur état.**
  - %FOUND : des lignes ont étés traitées
  - %NOTFOUND : aucune ligne trouvée
  - %ISOPEN ouverture d'un curseur
  - %ROWCOUNTnombre de lignes déjà traitées

- **%FOUND**
  - Précéder l'indicateur de la particule : **SQL**
  - **curseur implicite : SQL%FOUND**
    - **vaut TRUE**
      - **si INSERT, UPDATE, DELETE traite au moins une ligne**
      - **si SELECT ... INTO ... ramène une et une seule ligne. Dans le cas contraire un curseur explicite est nécessaire**

```
DECLARE
dept_no NUMBER(4) := 270;
BEGIN
DELETE FROM dept WHERE deptno= dept_no;
IF SQL%FOUND THEN -- d
    dbms_output.putline('Suppression avec succès du département ' ||
    dept_no);
END IF;
END;
/
```

5. Les curseurs en PL/SQL

Les attributs d'un curseur

- **Curseur explicite : nom_curseur%FOUND**
  - **Vaut TRUE si le dernier FETCH a ramené une ligne.**

- **Exemple :**

```
DECLARE
    CURSOR pl_nice IS
    SELECT pl#, plnom, sal
    FROM pilote
    WHERE adr='Nice';
    num pilote.pl#%TYPE;
    nom pilote.plnom%TYPE;
    salaire pilote.sal%TYPE;

BEGIN
    OPEN pl_nice;
    FETCH pl_nice INTO num, nom,salaire;
    WHILE pl_nice%FOUND
    --si pl_nice%found=false alors fin de la boucle
    LOOP
    ...
    FETCH pl_nice INTO num, nom,salaire;

    END LOOP;
    CLOSE pl_nice;
END ;
/
```

5. Les curseurs en PL/SQL

Les attributs d'un curseur

- **SQL%NOTFOUND et nom_curseur%NOTFOUND**

  – Ces deux curseurs ont un comportement **inverse** à ceux de **SQL%FOUND et nom_curseur%FOUND**

5. Les curseurs en PL/SQL

Les attributs d'un curseur

- **%ISOPEN**
  - **Curseur implicite : SQL%ISOPEN**
    - Toujours à FALSE car ORACLE referme les curseurs après utilisation.
  - **Curseur explicite : nom_curseur%ISOPEN**
    - **Vaut TRUE si le curseur est ouvert.**

Exemple :

```
DECLARE
CURSOR pl_nice IS
SELECT pl#, plnom, sal FROM pilote WHERE
   adr='Nice';
num                pilote.pl#%TYPE;
nom                pilote.plnom%TYPE;
salaire     pilote.sal%TYPE;
BEGIN
IF NOT(pl_nice%ISOPEN) THEN
   OPEN pl_nice;
END IF;
...
END ;
```

Les attributs d'un curseur

- %ROWCOUNT
  - Curseur implicite : SQL%ROWCOUNT
    - X : INSERT, UPDATE, DELETE.
      - X=0 Si aucune ligne traitée
      - Si non X= nombre de lignes traités
    - 0 : SELECT ... INTO : ne ramène aucune ligne
    - 1 : SELECT ... INTO : ramène 1 ligne

    - Exemple

```
DECLARE
empno1 NUMBER(6) := 7369;
BEGIN
DELETE FROM scott.emp WHERE empno = empno1;
DBMS_OUTPUT.PUT_LINE('Employes supprimés : ' ||
   TO_CHAR(SQL%ROWCOUNT));
END;
/
```

Les attributs d'un curseur

- **%ROWCOUNT**
  - **Curseur explicite : nom_curseur%ROWCOUNT**
    - **Contient le nombre de lignes traitées par un FETCH**
    - **Exemple**

```
DECLARE
Cursor curs_emp is select * FROM scott.emp ;
Emp1 curs_emp%rowtype;
BEGIN
Open curs_emp ;
loop
Fetch curs_emp into emp1;
Exit when curs_emp%notfound;
End loop;
DBMS_OUTPUT.PUT_LINE(' Nombre d''employes lus
   après les fetchs: ' ||
   TO_CHAR(curs_emp%ROWCOUNT));
close curs_emp ;
END;
/
```

Les attributs d'un curseur

- **Tableau de synthèse**

		%FOUND	%ISOPEN	%NOTFOUND	%ROWCOUNT
OPEN	AV	exception	FALSE	exception	exception
	AP	NULL	TRUE	NULL	0
First FETCH					
	AV	NULL	TRUE	NULL	0
	AP	TRUE	TRUE	FALSE	1
Next FETCH(es)					
	AV	TRUE	TRUE	FALSE	1
	AP	TRUE	TRUE	FALSE	data dependent
Last FETCH					
	AV	TRUE	TRUE	FALSE	data dependent
	AP	FALSE	TRUE	TRUE	data dependent
CLOSE					
	AV	FALSE	TRUE	TRUE	data dependent
	AP	exception	FALSE	exception	exception

AV: AVANT **AP**: APRES

5. Les curseurs en PL/SQL

Simplification d'écriture

- **Déclaration de variables sur la structure d'un curseur**

  – Au lieu de déclarer autant de variables que d'attributs ramenés par le SELECT du curseur, on peut utiliser une structure.

  – **Syntaxe :**

  DECLARE
  CURSOR nom_curseur IS ordre_select;
  nom_structure nom_curseur%ROWTYPE;

  – **Pour renseigner la structure :**
    - FETCH nom_curseur INTO nom_structure;

  – **Pour accéder aux éléments de la structure :**
    - nom_structure.nom_colonne

5. Les curseurs en PL/SQL

Simplification d'écriture

- **Utilisation de l'instruction FOR nom_struct IN nom_curseur au lieu de Open-fetch-Close**

  – **Au lieu d'écrire :**

```
DECLARE
CURSOR nom_curseur IS SELECT ... ;
nom_struct nom_curseur%ROWTYPE;

BEGIN
OPEN nom_curseur;
LOOP
FETCH nom_curseur INTO nom_struct;
EXIT WHEN nom_curseur%NOTFOUND;
...
END LOOP;
CLOSE nom_curseur;
END;
```

  – **il suffit d'écrire :**

```
DECLARE
   CURSOR nom_curseur IS SELECT ... ;
   nom_struct nom_curseur%ROWTYPE;

BEGIN
   FOR nom_struct IN nom_curseur LOOP
...
   END LOOP;
END;
```

Notes : L'ouverture et la fermeture du curseur se fait automatiquement

5. Les curseurs en PL/SQL

Simplification d'écriture

- Utilisation de l'instruction **FOR nom_struct IN (select …)** au lieu de **Open-fetch-Close** ou **FOR** nom_struct **IN** nom_curseur

  – **Sans déclaration de curseur ni de variable**

```
FOR nom_struct IN (SELECT ...)
LOOP
...
END LOOP;
```

  – **Exemple**

```
DECLARE
   BEGIN
   FOR emp1 IN (select * from scott.emp)
   LOOP
   dbms_output.put_line('emp1.ename='||emp1.ename);
   END LOOP;
END;
/
```

5. Les curseurs en PL/SQL

Utilisation des variables REF CURSOR

- Les variables **REF CURSOR doivent être définis sur des types REF CURSOR**. Un type REF CURSOR doit avoir été créé

- Les variables REF CURSOR **sont comme des pointeurs sur un Result Set** (résultat d'une requête)

- Les variables REF CURSOR sont utiles **quand on désire traiter le résultat d'une requête dans différents sous-programmes** (fonctions, procédures, etc.) ou dans **différents langages** (PL/SQL->Java, PL/SQL->C/C++, etc.)

- Une variable REF CURSOR peut être défini sur un type REF CURSOR faiblement typé ou fortement typé

- La déclaration du type **REF CURSOR** peut se faire au **niveau d'un bloc PL/SQL** ou au niveau **d'un package** pour factoriser et éviter des répétitions de déclaration

5. Les curseurs en PL/SQL

Utilisation des variables REF CURSOR

- **Déclaration du type REF CURSOR dans un bloc PL/SQL**

  – **Exemple**

```
DECLARE
TYPE empcurtyp IS REF CURSOR RETURN
   employees%ROWTYPE; -- Fortement typé
TYPE genericcurtyp IS REF CURSOR; -- Faiblement type
cursor1 empcurtyp; -- déclare d'une variable cursor
cursor2 genericcurtyp; -- déclare d'une variable cursor
my_cursor SYS_REFCURSOR; -- Permet d'éviter de
                         -- déclarer de nouveaux types
TYPE deptcurtyp IS REF CURSOR RETURN
   scott.dept%ROWTYPE;
dept_cv deptcurtyp; -- déclare d'une variable cursor
Begin
…
End;
/
```

5. Les curseurs en PL/SQL

Utilisation des variables REF CURSOR

- **Déclaration du type REF CURSOR dans un Package PL/SQL**

  – L'instruction **OPEN-FOR**

    - Permet d'exécuter une requête associée à une variable de type REF CURSOR. A la fin de l'exécution, le curseur sera positionné avant la première ligne résultante

    - **Syntaxe**

      – OPEN {cursor_variable_name | :host_cursor_variable_name} FOR select_statement [using_clause]

      – Cursor_variable_name : nom d'une variable de type ref cursor
      – :host_cursor_variable_name et using_clause utile si SQL dynamique

  – **Exemple**

    Voir page précédente

5. Les curseurs en PL/SQL

Utilisation des variables REF CURSOR

- **Déclaration du type REF CURSOR dans un Package PL/SQL**

  – Permet d'éviter de redéclarer le type dans chaque bloc ou il est utile

  – **Exemple**

```
CREATE OR REPLACE PACKAGE PK_REFCURSOR IS
TYPE refCursorType IS REF CURSOR;
End;
/

CREATE OR REPLACE FUNCTION
getDepartementEmployes (deptno1 IN number)
RETURN PK_REFCURSOR.refCursorType IS
cursListEmployes   PK_REFCURSOR.refCursorType;
Begin
    OPEN cursListEmployes
    FOR select e.* from emp e, dept d
    where e.deptno=d.deptno and d.deptno=deptno1;
    return cursListEmployes ;
end;
/
```

5. Les curseurs en PL/SQL

Utilisation des variables REF CURSOR

- **Déclaration du type REF CURSOR dans un Package PL/SQL**
  - Permet d'éviter de redéclarer le type dans chaque bloc ou il est utile
  - **Exemple**

```
set serveroutput on
declare
cursListEmployes  PK_REFCURSOR.refCursorType;
unEmploye        emp%rowtype;
begin
cursListEmployes := getDepartementEmployes (10);
loop
    fetch cursListEmployes  into unEmploye;
    EXIT WHEN cursListEmployes%notfound;
    dbms_output.put_line('numero employé='||
    unEmploye.empno
    || ' Nom Employé='|| unEmploye.ename);
end loop;
end;
/
```

6. Gestion des erreurs en PL/SQL

Généralités

- On distingue **deux types d'erreurs** :

  - **Les erreurs ORACLE** : ce sont les erreurs qui violent les règles sémantiques du noyau Oracle ou des outils Oracle. Par exemple **Division par zéro** ou **violation des contraintes d'intégrité dans une table Oracle**

  - **Les erreurs utilisateur** : ce sont les erreurs qui violent les règles de gestions d'une application utilisateur. Par exemple **le montant du stock est trop bas pour accepter une commande**

- La tâche du développeur Oracle est d'écrire les programmes les plus sûrs possible. Il doit donc bien contrôler les erreurs et bien les gérer

- Il doit savoir anticiper sur les erreurs pour mieux les contrôler et les gérer

- La section EXCEPTION permet de gérer les erreurs survenues lors de l'exécution d'un bloc PL/SQL.

6. Gestion des erreurs en PL/SQL

Les erreurs Oracle

- **Les erreurs ORACLE** : ce sont les erreurs qui violent les règles sémantiques du noyau Oracle ou des outils Oracle. Par exemple **Division par zéro** ou **violation des contraintes d'intégrité dans une table Oracle**

- Certaines **erreurs sont détectés à la compiltation et d'autres à l'exécution** (Runtime errors). Celles qui nous intéressent lors de la gestion des erreurs sont celles détectées à l'exécution

- Le noyau Oracle est capable d'identifier des milliers d'erreurs. Les erreurs sont classés par catégories. Il existe par exemples les catégories suivantes :

  - ORA-nnnnn : Erreur de la catégorie ORA suivit d'un numéro sont des erreurs lié au noyau Oracle. Exemple:
    - 1) ORA-01403 est généré si une requête SQL ne renvoie aucune ligne.
    - 2) ORA-01017 est généré si un programme tente de se connecté et que cette dernière est réfusée
    - 3) ORA-1422 est généré si une requête ramène plus d'une ligne et qu'aucun curseur n'a été défini

6. Gestion des erreurs en PL/SQL

Les erreurs Oracle

- **Gestion des erreurs Oracle survenant à l'exécution**
  - **On distingue deux types d'erreurs :**
    - **Celles ayant des noms d'exceptions prédéfinies**
    - **Et celle n'ayant aucun nom d'excption prédéfini**
- **Gestion des erreurs ayant des noms d'exceptions prédéfinis**
  - Certaines erreurs ORACLE ont déjà un nom. Il est donc inutile de les déclarer à nouveau. On utilise leur nom dans la section EXCEPTION.

```
DECLARE
...
BEGIN

… --l'erreur Oracle est détectée par le système

EXCEPTION
WHEN nom_erreur THEN ...--traitement de l'erreur
END;
```

Les erreurs Oracle

- **Gestion des erreurs ayant des noms d'exceptions prédéfinis**
  - Exemple

```
DECLARE
    Emp1          emp%rowtype;
BEGIN

-- Si l'employé numéro 1111 n'existe pas le programme
-- continue dans la section Exception. Le traitement de
-- L'exception prédéfinie NO_DATA_FOUND est fait.

Select  * INTO emp1 FROM emp where empno=1111;

EXCEPTION
WHEN NO_DATA_FOUND THEN
    -- Le programme se poursuit dans cette section si
    -- L'exception NO_DATA_FOUND est automatiquement
    -- levée
    DBMS_OUTPUT.PUT_LINE('Aucune ligne
    sélectionnée');
--traitement de l'erreur
END;
/
```

6. Gestion des erreurs en PL/SQL

Les erreurs Oracle

- **Gestion des erreurs ayant des noms d'exceptions prédéfinis**
  - Tableau des exceptions prédéfinies

Exception	Erreur Oracle	SQLCODE	Condition de déclenchement
ACCESS_INTO_NULL	06530	-6530	Tentative d'assigner des valeurs à un objet non initialisé
CASE_NOT_FOUND	06592	-6592	Choix absent dans la clause WHEN-CASE. De plus il n'ya pas de ELSE
COLLECTION_IS_NULL	06531	-6531	Tentative d'appliquer des méthodes sur collection Nested table ou Varray non initialisée
CURSOR_ALREADY_OPEN	06511	-6511	Tentative d'ouvrir un curseur déjà ouvert
DUP_VAL_ON_INDEX	00001	-1	Tentative de dupliquer des valeurs dans une colonne ayant un index unique
INVALID_CURSOR	01001	-1001	Tentative d'effectuer une opération sur un curseur. Par exemple, fermer un curseur alors qu'il n'est pas ouvert
INVALID_NUMBER	01722	-1722	Echec de conversion d'une chaîne en un nombre
LOGIN_DENIED	01017	-1017	Echec de connexion. Mot de passe ou non d'utilisateur invalide
NO_DATA_FOUND	01403	+100	La requête Select INTO ne ramène aucune ligne.

6. Gestion des erreurs en PL/SQL

Les erreurs Oracle

- **Gestion des erreurs ayant des noms d'exceptions prédéfinis**
  - Tableau des exceptions prédéfinies

Exception	Erreur Oracle	SQLCODE	Condition de déclenchement
NOT_LOGGED_ON	01012	-1012	Tentative d'effectuer une action sur la base de données sans être connecté
PROGRAM_ERROR	06501	-6501	Problème interne PL/SQL
ROWTYPE_MISMATCH	06504	-6504	La variable ref curseur hôte et la variable ref curseur PL/SQL en jeu dans une affectation sont incompatibles
SELF_IS_NULL	30625	-30625	Tentative d'appel d'une méthode membre d'une instance non initialisé
STORAGE_ERROR	06500	-6500	Le code PL/SQL s'exécute OUT OF MEMORY ou dans une mémoire corrompue
SUBSCRIPT_BEYOND_COUNT	06533	-6533	Out of index dans une collection nested table ou varray. Index plus grand
SUBSCRIPT_OUTSIDE_LIMIT	06532	-6532	Out of index dans une collection nested table ou varray. Index hors limite
SYS_INVALID_ROWID	01410	-1410	Echec de conversion d'une chaîne de caractères en rowid universal. Car la chaîne ne représente pas rowid
TIMEOUT_ON_RESOURCE	00051	-51	Timeout suite à l'attente d'une ressource par Oracle
TOO_MANY_ROWS	01422	-1422	La requête Select Into renvoie plus d'une ligne. Alors qu'aucun curseur n'est associé

6. Gestion des erreurs en PL/SQL

Les erreurs Oracle

- **Gestion des erreurs ayant des noms d'exceptions prédéfinis**
  - Tableau des exceptions prédéfinies

Exception	Erreur Oracle	SQLCODE	Condition de déclenchement
VALUE_ERROR	06502	-6502	Erreur de calcul, de conversion, de troncature ou de contrainte de taille
ZERO_DIVIDE	1476	-1476	Tentative de division pas zéro

6. Gestion des erreurs en PL/SQL

Les erreurs Oracle

- **Gestion des erreurs à l'exécution d'Oracle N'AYANT PAS des noms d'exceptions prédéfinis**

  – Seule les erreurs à l'exécution courante on un nom d'exception prédéfini

  – Les autres erreurs Oracle survenant à l'exécution n'ont pas de nom par défaut

  – Le développeur peut leur donner un nom d'exception afin de les traiter grâce à la routine PRAGMA EXCEPTION_INIT(nomException, code_erreur)

  – Une autre solutions peut être de les traiter en utisant la clause OTHERS (traitement alternatif pour toutes les exceptions non traitées individuellement)

6. Gestion des erreurs en PL/SQL

Les erreurs Oracle

- **Gestion des erreurs à l'exécution d'Oracle N'AYANT PAS des noms d'exceptions prédéfinis**

– Syntaxe

```
DECLARE
   nom_erreur EXCEPTION; --Déclaration d'un nom à d'exception
   PRAGMA EXCEPTION_INIT(nom_erreur,code_erreur)
   --on associe le nom de l'erreur à un code erreur
   ...
BEGIN
   ... --l'erreur Oracle est détectée par le système
EXCEPTION
   WHEN nom_erreur THEN ...--traitement de l'erreur
   WHEN OTHERS THEN …
      -- Traitement de tout autre erreur non traitée dans les
      -- clauses WHEN précédentes
END;
```

Remarque :

– on sort du bloc après le traitement de l'erreur.

6. Gestion des erreurs en PL/SQL

Les erreurs Oracle

- **Gestion des erreurs à l'exécution d'Oracle N'AYANT PAS des noms d'exceptions prédéfinis**

  – **Exemple**

```
-- Transaction 1
DECLARE
deadlock_detected EXCEPTION;
PRAGMA EXCEPTION_INIT(deadlock_detected, -60);
BEGIN
--T1 verrouille EMP
UPDATE EMP SET SAL=3000
WHERE ENAME='KING';
-- T1 Verrouille DEPT et se
-- met en attente
UPDATE DEPT SET
LOC='Nice'
WHERE deptno=10;
EXCEPTION
WHEN deadlock_detected THEN
DBMS_OUTPUT.PUT_LINE('Dead lock détecté');
ROLLBACK;
END;
/
```

```
-- Transaction 2
DECLARE
deadlock_detected EXCEPTION;
PRAGMA EXCEPTION_INIT(deadlock_detected, -60);
BEGIN
-- T2 Verrouille DEPT
UPDATE DEPT SET
LOC='Marseille'
WHERE deptno=10;

--T2 verrouille EMP et met en attente
-- =>DEAD LOCK erreur
-- ORA-00060
UPDATE EMP SET SAL=3000
WHERE ENAME='KING';
EXCEPTION
WHEN deadlock_detected THEN
DBMS_OUTPUT.PUT_LINE('Dead lock détecté');
ROLLBACK;
END;
/
```

6. Gestion des erreurs en PL/SQL

Les erreurs Oracle

- Les fonctions SQLCODE et SQLERRM

  – SQLCODE renvoie le code de l'erreur courante (numérique). Si aucune exception n'est survenue SQLCODE vaut toujours 0

  – SQLERRM[(code_erreur)] renvoie le libellé de l'erreur courante ou le libellé de l'erreur dont le numéro est passé en paramètre.

6. Gestion des erreurs en PL/SQL

Les erreurs Oracle

• Les fonctions SQLCODE et SQLERRM

– Exemple

```
DECLARE
    Emp1            emp%rowtype;
BEGIN

-- Si l'employé numéro 1111 n'existe pas le programme
-- continue dans la section Exception. Le traitement de
-- L'exception prédéfinie NO_DATA_FOUND est fait.

Select  * INTO emp1 FROM emp where empno=1111;
DBMS_OUTPUT.PUT_LINE('NomPilote ='|| emp1.ename);

EXCEPTION
WHEN NO_DATA_FOUND THEN
    -- Le programme se poursuit dans cette section si
    -- L'exception NO_DATA_FOUND est automatiquement
    -- levée
    DBMS_OUTPUT.PUT_LINE('Aucune ligne sélectionnée');
    DBMS_OUTPUT.PUT_LINE('Code erreur=' || sqlcode || '
    Message Oracle = '||sqlerrm(sqlcode));

--traitement de l'erreur
END;
/
```

6. Gestion des erreurs en PL/SQL

Les erreurs utilisateurs

- **Ce** sont les erreurs qui violent les règles de gestions d'une application utilisateur. Par exemple **le montant du stock est trop bas pour accepter une commande**

- **Le développeur peut créer ces exceptions (nomException EXCEPTION), il peut les lever (RAISE) et les gérer (clause EXCEPTION)**

- **Syntaxe**

```
DECLARE
nom_erreur EXCEPTION; --on donne un nomd'exception
...
BEGIN
        IF condition …THEN
            RAISE nom_erreur; --on lève l'exception
        END IF;
        ...
        EXCEPTION
        WHEN nom_erreur THEN ... --traitement de l'exception
END;
```

Remarque : on sort du bloc après le traitement de l'erreur.

6. Gestion des erreurs en PL/SQL

Les erreurs utilisateurs

- **Exemple**
  - **Créer un nouveau Departement dans la base de données. Si la localité vaut NULL, créer, lever et traiter une exception appelé BAD_DEPARTEMENT_LOC**
  - **Lui affecter une localité par défaut : Lyon**

```
DECLARE
BAD_DEPT_LOC EXCEPTION;
    --on donne un nomd'exception
     dept1       DEPT%ROWTYPE;
BEGIN
       dept1.deptno:=50;
       dept1.dname:='Achat';
       dept1.loc:=null;
       IF dept1.loc IS NULL THEN
                RAISE BAD_DEPT_LOC; --on lève l'exception
        END IF;
        INSERT INTO DEPT Values(dept1.deptno, dept1.dname, dept1.loc);

        EXCEPTION
        WHEN BAD_DEPT_LOC THEN
        Dept1.loc:='Lyon';
        INSERT INTO DEPT Values(dept1.deptno, dept1.dname, dept1.loc);
        DBMS_OUTPUT.PUT_LINE('La localité du département ne doit pas être null :
        La valeur par défaut Lyon a été affecté');
        --WHEN NO_DATA_FOUND THEN

        --WHEN OTHERS THEN

         --traitement de l'exception

END;
/
```

Remarque : on sort du bloc après le traitement de l'erreur.

6. Gestion des erreurs en PL/SQL

Les erreurs utilisateurs

- **Définition de ses propres messages d'erreurs**

  – La procédure RAISE_APPLICATION_ERROR permet au développeur de définir ses propres messages d'erreur de type ORA-errorNumber à l'image des erreurs à l'exécution d'Oracle.

  – Ainsi ses erreurs seront remontées à l'application. On évite ainsi d'avoir des exceptions non traitées

  – Syntaxe

    •Raise_application_error(errorNumber, message[, {TRUE | FALSE}])

    •ErrorNumber : doit être compris entre -20000 et -20999

    •Message : message de 2048 octets maximum

    •True : erreur est empilée. FALSE elle se substitue aux erreurs précédentes

6. Gestion des erreurs en PL/SQL

Les erreurs utilisateurs

- **Définition de ses propres messages d'erreurs**

  – Exemple

```
DECLARE
dept1        DEPT%ROWTYPE;
BEGIN
      dept1.deptno:=60;
      dept1.dname:='Sport';
      dept1.loc:=null;
      IF dept1.loc IS NULL THEN
            RAISE_APPLICATION_ERROR(-20000, 'La localité du
      département ne doit pas être null : La valeur par défaut Lyon a
      été affecté'); --on lève l'exception avec son propore message et code
      d'erreur
      END IF;
      INSERT INTO DEPT Values(dept1.deptno, dept1.dname, dept1.loc);
      EXCEPTION
      WHEN OTHERS THEN
      Dept1.loc:='Lyon';
      INSERT INTO DEPT Values(dept1.deptno, dept1.dname, dept1.loc);
      DBMS_OUTPUT.PUT_LINE('Code erreur : '|| sqlcode || '
      Message :'|| sqlerrm(sqlcode));
END;
/
```

6. Gestion des erreurs en PL/SQL

Les erreurs utilisateurs

- **Définition de ses propres messages d'erreurs et les associer A UN NOM D'EXCEPTION**

– Exemple

```
DECLARE
dept1          DEPT%ROWTYPE;
BAD_DEPT_LOC EXCEPTION; --Déclaration d'un nom à d'exception
PRAGMA EXCEPTION_INIT(BAD_DEPT_LOC, -20000) ;

BEGIN
      dept1.deptno:=60;
      dept1.dname:='Sport';
      dept1.loc:=null;
      IF dept1.loc IS NULL THEN
            RAISE_APPLICATION_ERROR(-20000, 'La localité du
       département ne doit pas être null : La valeur par défaut Lyon a été affecté');
       --on lève l'exception avec son propore message et code d'erreur
       -- RAISE BAD_DEPT_LOC;
       END IF;
       INSERT INTO DEPT Values(dept1.deptno, dept1.dname, dept1.loc);
       EXCEPTION
       WHEN BAD_DEPT_LOC THEN
            Dept1.loc:='Lyon';
            INSERT INTO DEPT Values(dept1.deptno, dept1.dname,   dept1.loc);
            DBMS_OUTPUT.PUT_LINE('Code erreur : '|| sqlcode || '
            Message :'|| sqlerrm(sqlcode));
END;
/
```

6. Gestion des erreurs en PL/SQL

- **Les fonctions suivantes permettent d'intégrer des traces dans le code. Elles sont similaires aux sorties textes dans la console JAVA**
  - **dbms_output.put_line(*expression*)**
  - **dbms_output.put(*expression*)**
  - **dbms_output.new_line**

  - Nota : *Expression* est une chaîne de caractères

- **Attention**
  - Afin d'activer les sorties à l'écrans, le code SQLPLUS suivant doit au moins une fois pendant une session précéder le code PL/SQL

    SET SERVEROUTPUT ON

- **Pour afficher les erreurs en cas de codage des packages ou fonctions**
  - Show error

7. Exercices : PLSQL

Exercice 1

Ecrire un programme PL/SQL qui permet d'afficher le numéro, le nom, la date de naissance, l'adresse, le salaire et le téléphone d'un pilote connaissant son numéro.

Si pilote n'existe pas afficher un message d'erreur. Pilote inexistant

Saisissez en interactif le numéro de pilote pour lequel on souhaite afficher les informations.

Tester le programme avec les numéros suivants : 1 puis 100

Exercice 2

Ecrire un programme PL/SQL qui permet d'insérer un nouveau pilote dans la base.

insert into pilote values(22, nomPilote , to_date('04-08-1966', 'DD-MM-YYYY') , 'Nice', null, 23000.6);

Si le pilote n'est pas créé, afficher un message d'erreur.

Tester le programme avec les noms suivants :

nomPilote= 'Bill' puis nomPilote= 'Barak'

Exercice 3

Ecrire un programme PL/SQL qui permet d'affecter le salaire de 23000 à tous les pilotes qui habitent dans une ville donnée.

Si aucun pilote n'y habite, afficher un message d'erreur. Aucun pilote dans cette ville

Tester le programme avec les numéros suivants :

adr= 'Nice' puis adr= 'Louvain'

7. Exercices : PLSQL

Exercice 4

Ecrire un programme PL/SQL qui permet d'afficher le numéro, le nom, la date de naissance, l'adresse, le salaire et le téléphone des pilotes habitant une ville donnée.

S'il n'ya aucun pilote dans la dite ville, afficher un message d'erreur. Pas de Pilote dans cette ville.

4.1 Définir un curseur. Contrôler le curseur avec OPEN-FETCH-CLOSE

utiliser la boucle perpétuelle LOOP … END LOOP. Comment sortir de la boucle

utiliser la boucle WHILE condition LOOP … END LOOP. Quelle est la condition de sortie dans cette boucle

4.2 Définir un curseur. Controler le curseur avec la boucle FOR element IN …

Tester le programme avec les villes de villes de : Paris puis Lilles

8. L'option procédurale

PLAN

8.1 Généralités

- **L'option procédurale se compose de :** procédures, fonctions, packages et triggers
- **permet à Oracle de stocker des traitements** (*code PL*SQL uniquement*) au niveau du moteur base de données
- **Elle est maintenant *intégré* dans le noyau Oracle**
- **C/S 2ème Génération :** Jusqu'à 40 % des traitements transposables du ***coté serveur***
- **Activation de l'option procédurale**
  - Exécuter le script *catproc.sql* étant SYS
    sql>@@$ORACLE_HOME/rdbms/admin/catproc

8.2 Procédures et fonctions

- **Définition**

  - Unité de traitement PL/SQL pouvant contenir des Ordres SQL, des variables, des constantes, des curseurs et des exceptions

- **Une fonction à la différence d'une procédure rend une valeur**

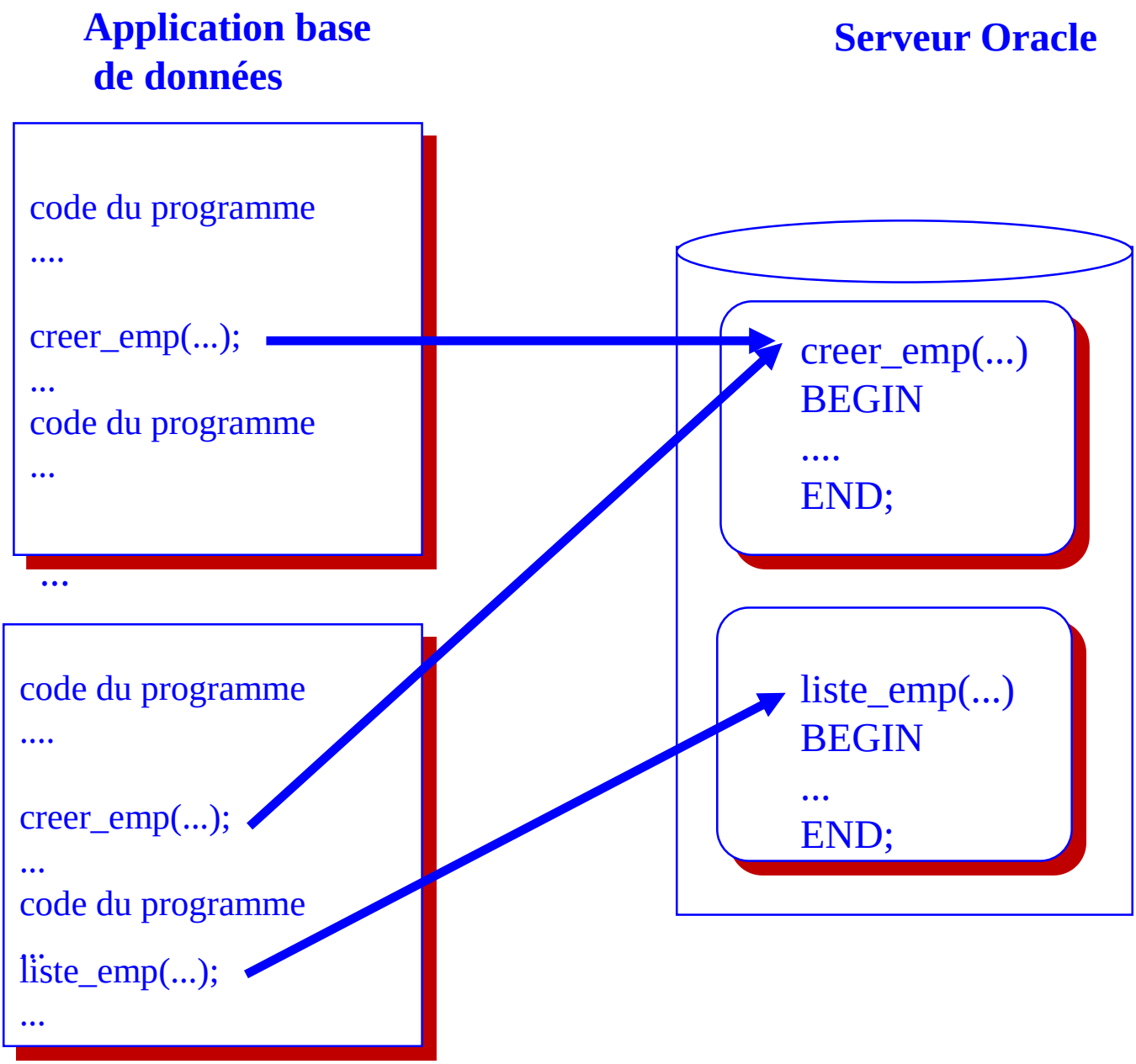
- **une procédure ou une fonction peut être :**

  - exécutée en interactif sous SQL*PLUS ou SQL*DBA
  - appelée depuis une application
  - appelée depuis une autre procédure ou fonction

- **une procédure ou une fonction comporte deux parties créées en même temps :**
  - la partie spécification (l'entête)
  - la partie implémentation (le corps)

8.2 Procédures et fonctions

Appel de procédures et fonctions depuis un applicatif



8.2 Procédures et fonctions

Avantages des procédures et fonctions stockées dans la base

- *plus de sécurité* : il n'est plus utile de donner des droits aux utilisateurs sur les objets de base manipulés par les procédures (tables, vues, ...)

- *plus de performance* :
  - échanges réduits entre l'application et la base
  - une compilation pour plusieurs exécutions
  - moins d'E/S, procédure déjà dans la zone des requêtes

- *Economie d'espace mémoire* : une seule copie en SGA mais des exécutions par plusieurs applications possibles

- *une plus grande productivité* :
  - évolutivité : la modification du corps d'une procédure n'implique pas celle des applications qui l'appelle
  - centralisation des modifications

8.2 Procédures et fonctions

Création de procédures et de fonctions

- privilège requis : CREATE PROCEDURE ou CREATE ANY PROCEDURE

Syntaxe de création d'une procédure

```
CREATE [OR REPLACE] PROCEDURE [schema.]fonction
      [ (<liste d'arguments>)]
    [{invoker_rights_clause | DETERMINISTIC| parallel_enable_clause} ]
  {aggreate_pipelined_type |[PIPELINED]{is | as}
     {plsql_function_body | call_spec}
  }
```

8.2 Procédures et fonctions

Création de procédures et de fonctions

- privilège requis : CREATE PROCEDURE ou CREATE ANY PROCEDURE

Syntaxe de création d'une fonction

CREATE [OR REPLACE] FUNCTION [schema.]fonction
[(<liste d'arguments>)] **RETURN type**
[{invoker_rights_clause | DETERMINISTIC| parallel_enable_clause}]
{aggreate_pipelined_type |[PIPELINED]{is | as}
{plsql_function_body | call_spec}

}

Liste d'arguments ::=
nom argument [{IN | OUT | IN OUT}[NOCOPY]] type
[, nom argument [{IN | OUT | IN OUT}[NOCOPY]] type];

invoker_rights_clause::=AUTHD {CURRENT_USER | DEFINER}

parallel_enable_clause ::=
[(PARTITION argument BY {ANY|{HASH |
RANGE}(column[,column]…)}[streaming_clause]

streaming_clause::={ORDER |CLUSTER} expr BY (column[,column]…)

call_spec :: LANGUAGE {java_declaration | c_declaration}

java_declaration ::= JAVA NAME string

c_declaration ::= C [NAME name] LIBRARY lib_name

[AGENT IN (argument[,argument]…)]

[WITH CONTEXT] [PARAMETERS (parameter[,parameter]…)]

8.2 Procédures et fonctions

Création de procédures et de fonctions

Syntaxe de création d'une fonction

Plsql_function_body::=

```
BEGIN  -- bloc PLSQL
   -- corps de la fonction
END;
```

Paramètre	Description
OR REPLACE	Recréer la procédure ou la fonction
IN	Paramètre en entrée
OUT	Paramètre en sortie
IN OUT	Paramètre en entrée sortie
NOCOPY	Passage rapide de paramètre en OUT ou IN OUT
RETURN type	Type de retour s'il s'agit d'une fonction
Invoker_righths_clause	Permet de spécifier avec quelles droit exécuter la procédure. Ceux du propriétaire ou ceux de l'exécuteur
AUTHID clause	Spécifier CURRENT_USER pour exécuter avec les privilèges de l'utilisateur courant ou DEFINER pour ceux du propriétaire
DETERMINISTIC clause	La fonction doit renvoyer une même résultat si elle est appelée avec les valeurs de paramètres
Parallel_enable_clause	PARALLEL_ENABLE permet une exécution parallèle de la fonction dans une requête parallèle
PIPELINED clause	Indique à oracle de renvoyer le résultat d'une table fonction itérativement
AGGREGATE USING clause	Spécifier cette clause pour identifier cette fonction comme étant une fonction agrégat. Fonction de groupe. Peut apparaître derrière HAVING
IS | AS clause	Permet ensuite de définir un body
Pl/sql_subprogram_body	Corps du programme PL/SQL
Call_spec	Permet de mapper une fonction java ou C
AS EXTERNAML	Alternative pour déclarer des méthodes C

8.2 Procédures et fonctions

Création de procédures et de fonctions (suite)

Exemple de création de procédure

```
CREATE OR REPLACE PROCEDURE pnouveau_sal (empid IN
NUMBER, taux IN NUMBER) IS
NO_ROW_EXECPT  EXCEPTION;
BEGIN
   UPDATE emp  SET sal = sal* (1 + taux)  WHERE empno = empid ;
      IF SQL%ROWCOUNT = 0 THEN
   RAISE  NO_ROW_EXECPT;
   END IF;
   EXCEPTION
      WHEN  NO_ROW_EXECPT THEN
      DBMS_OUTPUT.PUT_LINE('Employe inexistant');
      RAISE_APPLICATION_ERROR(-20012, 'Employe Nr. ' ||
   to_char(empid) || 'Inexistant');
END pnouveau_sal;
/
```

Exemple de création d'une fonction

```
CREATE OR REPLACE FUNCTION emp_info(empid IN number) return
emp%ROWTYPE AS
    emprow emp%rowtype ;
 BEGIN
   SELECT empno, ename, job, mgr, hiredate, sal, comm, deptno
       INTO emprow FROM emp
         WHERE empno = empid;
   RETURN (emprow);
END;
/
```

Note : sous sqlplus ou sql la dernière ligne de la procédure doit être /

8.2 Procédures et fonctions

Modification et ré-compilation de procédures et fonctions

- **Ré-compilation** :En cas d'évolution des objets du schéma (tables, ...) manipulés dans une procédure ou une fonction. En de code invalide, oracle tente une ré-compilation à l'exécution

- **privilège requis** : ALTER PROCEDURE ou ALTER ANY PROCEDURE

- **Syntaxe**

ALTER { FUNCTION | PROCEDURE }
[schema.]nom COMPILE
[compiler_parameters_clause]
[,compiler_parameters_clause] [REUSE SETTINGS];

compiler_parameters_clause ::
parameter_name=parameter_value

Exemple

ALTER FUNCTION emp_info COMPILE;

8.2 Procédures et fonctions

Modification et récompilation de procédures et fonctions

Mots clés ou paramètres	Description
nom	nom de la procédure ou fonction
schema	propriétaire de l'objet
COMPILE	récompiler l'objet en cas d'évolution des objets référencés
compiler_parameters_clause	Permet de passer des paramètres de compilation au compilateur PL/SQL. Les paramètres peuvent être : PLSQL_OPTIMIZE_LEVEL, PLSQL_CODE_TYPE, PLSQL_DEBUG, PLSQL_WARNINGS, PLSQL_CCFLAGS, et NLS_LENGTH_SEMANTICS. Ces paramètres être modifiés pour une session ALTER SESSION ou une instance ALTER SYSTEM
DEBUG	suggère au compilateur de générer le code pour le déboggage. Idem PLSQL_DEBUG=TRUE dans la clause compiler_parameters_clause
REUSE SETTINGS	Permet à Oracle de conserver les paramètres de compilation

8.2 Procédures et fonctions

Suppression d'une procédure ou d'une fonction

- privilège requis DROP PROCEDURE ou DROP ANY PROCEDURE

- **Syntaxe**

  DROP FUNCTION | PROCEDURE
  [schema.]nomproc;

- **Exemple**

  DROP PROCEDURE tintin.nouveau_sal ;

8.3 Packages

Généralités

- **Définition**
  Un package Oracle est à l'image du package ADA une unité de traitement PL/SQL nommée, regroupant des procédures, des fonctions avec des curseurs et des variables qu'elles utilisent ensemble

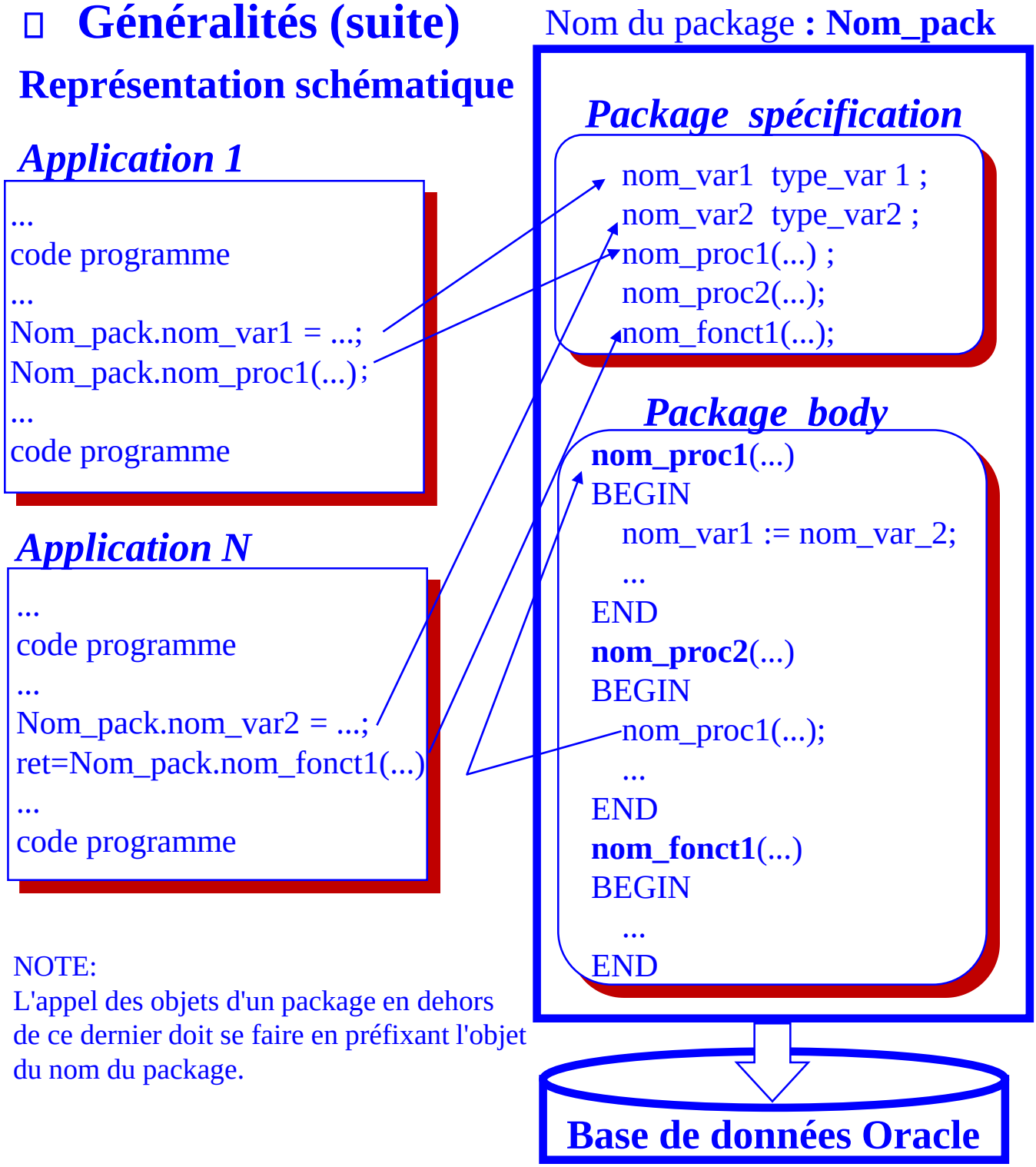
- **Un package comporte deux parties** : la partie spécification (PACKAGE SPECIFICATION) et la partie implémentation (PACKAGE BODY)

- Les composants d'un package peuvent être **publics** ou **privés** : Les composants publics sont déclarés au niveau de la partie spécification et les composants privés au niveau la partie implémentation

8.3 Packages

Généralités (suite)

Représentation schématique



NOTE:
L'appel des objets d'un package en dehors de ce dernier doit se faire en préfixant l'objet du nom du package.

8.3 Packages

Création d'un Package

- La création d'un package consiste à créer la spécification puis le corps du package

Syntaxe de création de la partie spécification

CREATE [OR REPLACE] PACKAGE [schéma.]nom_package
[invoker_rights_clause] {IS | AS} spéficification PL/SQL

invoker_rights_clause::= AUTHID {CURRENT_USER | DEFINER}

spécification PL/SQL ::=
déclaration de variable |
déclaration d'enregistrement |
déclaration d'execption |
déclaration de table PL/SQL |
déclaration de fonction |
déclaration de procédure ...

Mots clés ou paramètres Description

schéma Nom du schéma auquel le package appartient
nom_package Nom du package dans le schéma
Invoker_righths_clausePermet de spécifier avec quelles droit exécuter la procédure. Ceux du propriétaire ou ceux de l'exécuteur
AUTHID clauseSpécifier CURRENT_USER pour exécuter avec les privilèges de l'utilisateur courant ou DEFINER pour ceux du propriétaire

spécification PL/SQL déclaration de variables, fonctions, procédures, ... globales

8.3 Packages

Création d'un package (suite)

Syntaxe de création de la partie implémentation

CREATE [OR REPLACE]
  PACKAGE BODY [schéma.]nom_package
    {IS | AS} corps PL/SQL

corps PL/SQL ::=
        déclaration de variable |
        déclaration d'enregistrement |
        déclaration d'execption |
        déclaration de table PL/SQL |
        corps de fonction |
        corps de curseur |
        corps de procédure ...

Mot clé ou paramètre	Description
schéma	Nom du schéma auquel le package appartient
nom_package	Nom du package dans le schéma
corps PL/SQL	déclaration de variables, ..., corps fonctions, des procédures, ...

Note : Les noms utilisés dans la partie spécification doivent être les mêmes que dans la partie implémentation.

8.3 Packages

Création d'un package (suite)

Exemples de création de la partie spécification

```
CREATE OR REPLACE        PACKAGE tintin.gestion_employes
IS
   PROCEDURE pnouveau_sal (empid IN NUMBER, taux IN NUMBER);
   FUNCTION pemp_info(empid IN number) RETURN emp%ROWTYPE;

END gestion_employes ;
/
```

8.3 Packages

□ Création d'un package (suite)

Exemple de création de procédure

```
CREATE OR REPLACE
        PACKAGE BODY tintin.gestion_employes IS

PROCEDURE pnouveau_sal (empid IN NUMBER, taux IN NUMBER) IS
   NO_ROW_EXECPT                    EXCEPTION;

BEGIN
        UPDATE emp  SET sal = sal* (1 + taux)  WHERE empno = empid ;

   IF SQL%ROWCOUNT = 0 THEN

        RAISE  NO_ROW_EXECPT;

   END IF;

   EXCEPTION
              WHEN  NO_ROW_EXECPT THEN
                 DBMS_OUTPUT.PUT_LINE('Employe inexistant');
                    RAISE_APPLICATION_ERROR(-20012, 'Employe Nr. ' ||
                           to_char(empid) || 'Inexistant');
 END pnouveau_sal;

   FUNCTION pemp_info(empid IN number)  RETURN emp%ROWTYPE AS
         emprow emp%rowtype ;
        BEGIN
           SELECT empno, ename, job, mgr, hiredate, sal, comm, deptno
              INTO emprow FROM emp
                 WHERE empno = empid;
           RETURN (emprow);
        EXCEPTION
        WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Employe inexistant');
        RETURN NULL;
   END pemp_info   ;

END gestion_employes;

/
```

8.3 Packages

Avantages des packages

- Encapsulation et modularité : le package se comporte comme une boîte noire
- Possibilité de cacher des informations
- Séparation de la spécification des implémentations permet d'augmenter la productivité des développeurs
- + tous les avantages énumérés au niveau des fonctions et procédures (performance, partage, récompilation intelligente, ...)

8.3 Packages

Exécution d'une procédure

- **Etapes**
  - 1. Vérifier les droits de l'utilisateur (privilège objet EXECUTE)
  - 2. Vérifier la validité de la procédure
    - un objet référencé a - t - il changé (table, vue, procédure , ...) ?
    - un privilège sur le package a - t - il été retiré ?
    - un privilège sur un objet référencé a - t - il été révoqué ?
  - 3. La procédure à exécuter réside - t -elle déjà en SGA? sinon la charger et récompiler
  - 4. Exécuter la procédure.
- **Exemple en interactif**

```
sql > EXECUTE
            gestion_employes.pemp_info(7856);
sqlplus>EXECUTE nouveau_sal(7856, 0.5);
```

8.3 Packages

Informations sur les packages et les fonctions

Listes des vues utiles

. ALL_ERRORS, USER_ERRORS, DBA_ERRORS
. ALL_SOURCE, USER_SOURCE , DBA_SOURCE
. USER_OBJECT_SIZE, DBA_OBJECT_SIZE

Exemple 1 : Vue USER_SOURCE

Visualisation du code source et du nombre de lignes de la procédure de l'utilisateur courant

```
sql> SELECT line, text FROM user_source
      WHERE name = 'EMP_INFO';
```

8.3 Packages

Informations sur les packages et les fonctions (suite)

Exemple 2 : Vue USER_ERRORS

Visualisation des erreurs de compilation de l'utilisateur courant

```
sql>  SELECT name, type, line, position, text
      FROM user_errors
      WHERE name = 'EMP_INFO';
```

Exemple 3 : vue USER_OBJECT_SIZE

Espace consommé par la procédure EMP_INFO

```
sql>  SELECT name, sorce_size+parsed_size+ code_size +
          error_size "Taille totale" FROM user_object_size
        WHERE name = 'EMP_INFO';
```

8.3 Les packages

Packages fournis par Oracle et utilisables à la place de certains ordres SQL

Package dbms_session : gestion des infos. de session

Procédures : close_database_link, reset_package, set_label, set_nls, set_mls_label_format, set_role, set_sql_trace, unique_session_id, is_role_enabled, set_close_cached_open_cursors, free_unused_user_memory

Package dbms_ddl : Analyse et compilation des objets du schéma

Procédures : alter_compile, analyze_object

Package dbms_transaction : Gestion des transactions

Procédures : advise_commit, advise_rollback, advise_nothing, commit, commit_comment, commit_force, read_only, read_write, rollback, rollback_force, rollback_savepoint, savepoint, use_rollback_segment, purge_mixed, begin_discrete_transaction, local_transaction_id, step_id

Package dbms_utility : Ensemble de fonctions utilitaires

Procédures : compile_schema, analyze_schema, format_error_stack, format_call_stack, is_parallel_server, get_time, name_resolve

8.3 Les packages

Packages additionnels fournis par Oracle

Nom du package	Description	Procédures
dbms_alert	Gestion asynchrone des alertes des events de la BD	register, remove, signal, waitany, waitone, set_defaults
dbms_describe	permet de décrire les arguments des procédures stockées	describe_procedure
dbms_job	permet de gérer la soumission des job	submit, remove, change, what, next_date, sys as sysdba, broken, run
dbms_lock	Permet d'utiliser les mécanismes de verrouillage	allocate_unique, request, convert, release, sleep
dbms_output	permet d'envoyer des informations sur la sortie standard depuis une proécdure stockée	get_line, get_lines, new_line, put_line, …
dbms_pipe	permet la communication inter-session dans une même instance	create_pipe, pack_message, send_message, receive_message, next_item_type, unpack_message remove_pipe, purge, reset_buffer, unique_session_name
dbms_shared_pool	permet de conserver des objets dans shared pool area	sizes, keep, unkeep, aborted_request_threshold
dbms_application_info	permet de gérer les infos d'une application à des fins de performance ou d'audit	set_module, set_action, set_client_info, read_module read_client_info
dbms_system	permet d'activer des utilitaires systèmes tel tracer des requêtes	set_sql_trace_in_session

8.3 Les packages

Packages additionnels fournis par Oracle (suite)

Nom du package	Description	Procédures
dbms_space	fourni les infos sur l'espace des segments non accessibles via les vues	
dbms_sql	permet d'écrire des procédures stockées et des blocs PL/SQL anonymes en utilisant le SQL dynamique	open_cursor, parse, execute, bind_variable, define_column, define_column_long, is_open, execute_and_fetch, fetch_rows column_value, variable_value, column_value_long, last_error_position, close_cursor, last_row_count, last_row_id, last_sql_function_code
dbms_refresh	permet de gérer des groupes de snapshot	(voir dbmssnap.sql)
dbms_snapshot	permet de rafraichir des snapshot non membre d'un groupe	(voir dbmssnap.sql)
dbms_defer, dbms_defer_sys, dbms_defer_query	permet construire et administrer des appels de procédures distantes	(voir dbmdefr.sql)
dbms_repcat	permet d'utilier les mécanismes de réplication statique	(voir dbmsrepc.sql)
dbms_repcat_auth , dbms_repcat_ admin	permet de créer des utilisateurs ayant le privilège de répliquer	(voir dbmsrepc.sql)

NOTA : Voir le manuel Oracle << PL/SQL Packages and Types Reference >> pour la liste complètes des packages prédéfinis.

8.4 Les triggers Base de données

Généralités

- Un trigger BD est une action qui se déclenche avant ou après un événement.

- Les triggers BD Oracle sont des blocs PL/SQL stockés dans la base pouvant se déclencher automatiquement avant ou après :
  - ***L'exécution d'un Ordre SQL*** du Langage de Manipulation de Données ***LMD*** (insertion, modification ou suppression) sur une table ou sur une vue,
  - ***L'exécution D'un ordre SQL*** du Langage de Définition de Données ***LDD*** (CREATE …, ALTER …, DROP, …)
  - ***L'apparition d'un événement Bases de Données*** (connexion, déconnexion, erreurs, arrêt, démarrage d'une base, …)

- Il existe donc trois classes de triggers :
  - Des triggers LMD
  - Des triggers LDD
  - Des triggers liés à des événements Bases de Données(BD)

8.4 Les triggers Base de données

Les triggers LMD

- Ce sont des triggers associés aux ordres SQL (INSERT, UPDATE, DELETE) du langage de manipulation de données (LMD)

- La fréquence de déclenchement du trigger :
  - pour chaque ligne traitée (FOR EACH ROW)
  - pour toutes les lignes (statement level)

- Le moment de son déclenchement (avant le début des traitements PRE-CONDITION ou à la fin des traitements POST-CONDITION)
  - BEFORE
  - AFTER
  - INSTEAD OF

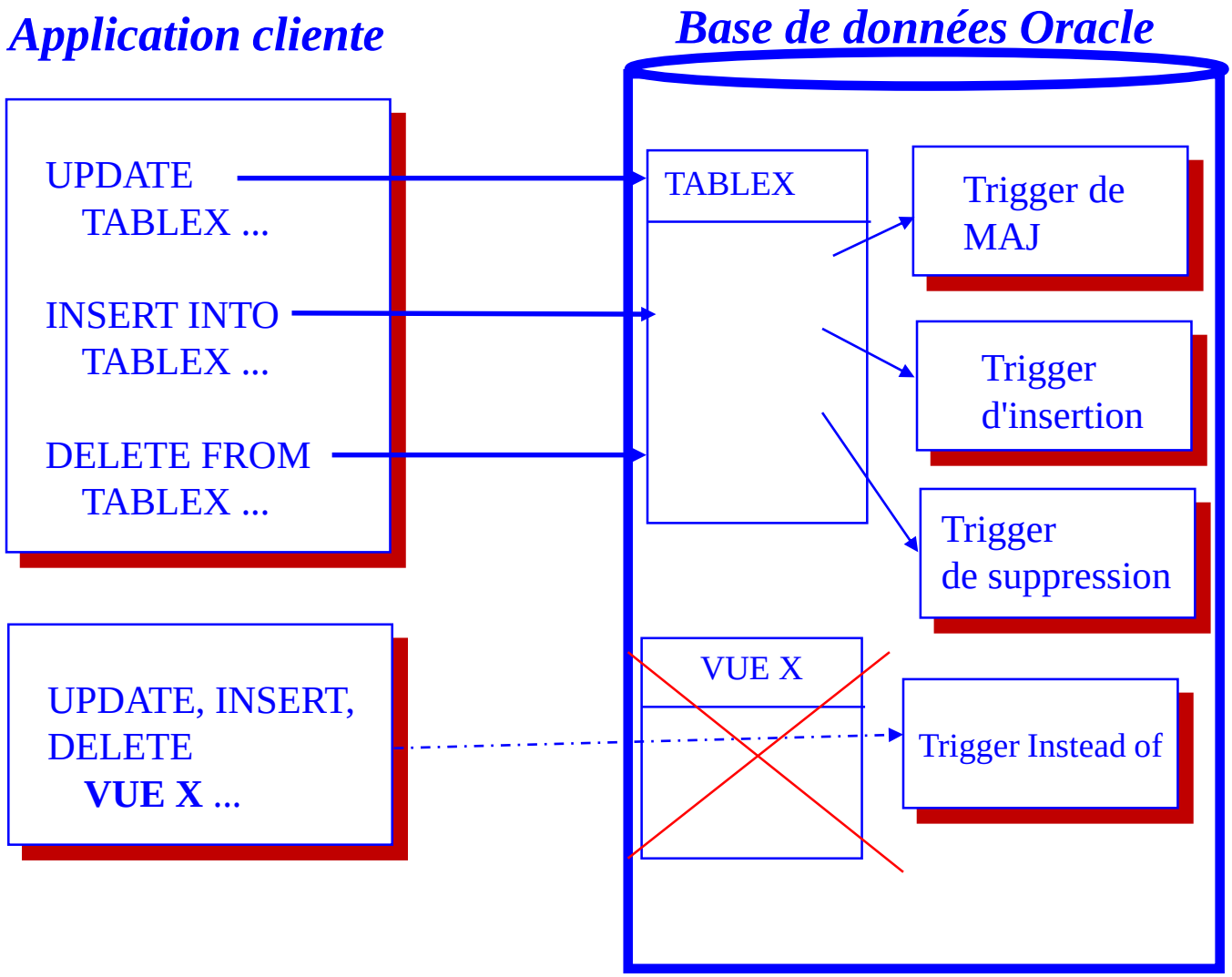
- Ils sont souvent associés aux tables et accessoirement aux VUES pour uniquement les triggers INSTEAD OF

- Un trigger **INSTEAD OF** se déclenche sur une **VUE** et remplace l'action prévue dans la requête de la vue

8.4 Les triggers Base de données

Les triggers LMD

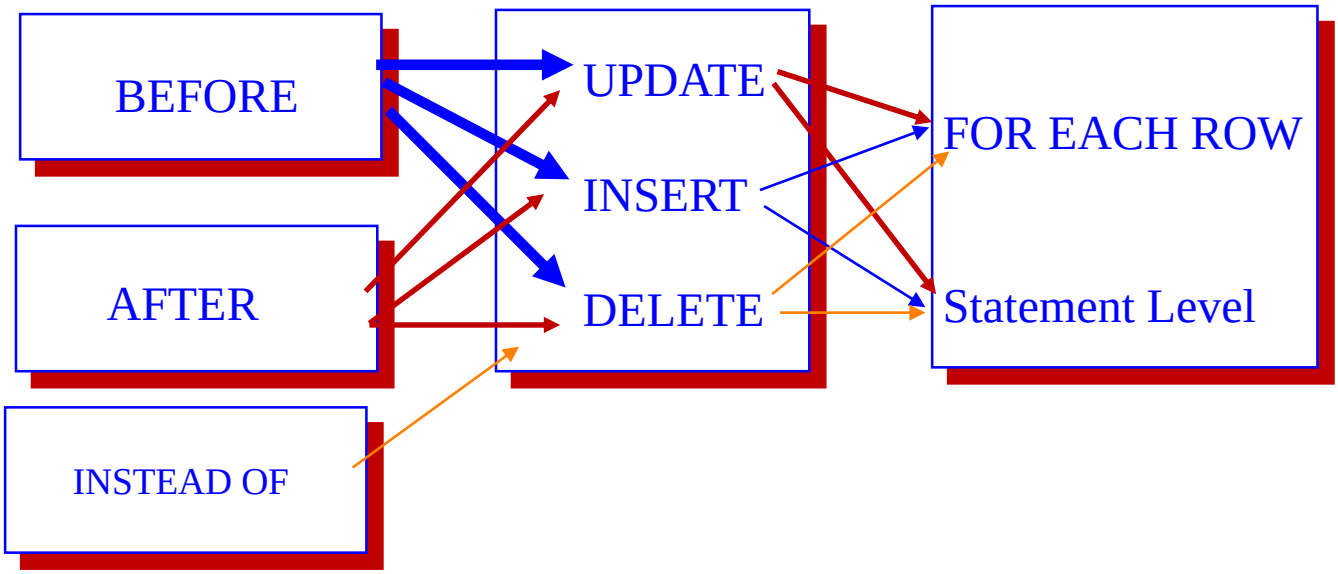
- Relation entre le déclenchement du trigger et l'ordre sql lancé depuis l'application cliente



8.4 Les triggers Base de données

Les triggers LMD

- Résumé du moment du déclenchement d'un trigger LMD



Note :

1) Un déclenchement conditionnel(clause WHEN) est possible. La clause WHEN ne peut être posée qu'en évaluation ligne par ligne

2) Un trigger INSTEAD OF est toujours de niveau ligne FOR EACH ROW

8.4 Les triggers Base de données

Les triggers LMD

- Deux blocs implicites **:NEW et :OLD** servent à accéder respectivement aux informations de la nouvelle ligne et de l'ancienne ligne de la table ou est posé le trigger

	:NEW	:OLD
INSERT	**:new**	**-**
UPDATE	**:new**	**:old**
DELETE	**-**	**:old**

Dans le corps du trigger il est possible de manipuler l'état AVANT et l'état NOUVEAU d'une ligne. Valable uniquement pour les trigger FOR EACH ROW

Exemple 1 : action possible dans un trigger ON DELETE sur la table DEPT. Il faut mettre à NULL la clé étrangère dans la table EMP si la clé primaire correspondant est supprimé ou modifiée dans La table DEPT

UPDATE emp SET deptno=null
WHERE emp.deptno=**:OLD.DEPTNO;**

8.4 Les triggers Base de données

Les triggers LMD

- Certain triggers peuvent être **multi-action (INSERT, UPDATE ou DELETE),** cela évite de créer un trigger pour chaque type d'action

- Si l'on souhaite dans le corps du trigger associer des instructions à un type d'action particulier, oracle propose trois mots clés : **INSERTING, UPDATING et DELETING**

- Il est dès lors possible dans le corps du trigger d'écrire

```
IF INSERTING THEN
   -- quelques actions à faire lors de l'insert
END IF;
IF INSERTING OR DELETING THEN
   -- quelques actions à faire lors de l'insert ou delete
END IF;
IF INSERTING OR UPDATING OR DELETING THEN
   -- quelques actions à faire lors de l'insert update
   -- ou delete
END IF;
```

8.4 Les triggers Base de données

Création d'un trigger

. Privilège requis : CREATE TRIGGER

Syntaxe de création d'un trigger

CREATE [OR REPLACE] TRIGGER [schema.] trigger
{ BEFORE | AFTER | INSTEAD OF }
{lmd_event_clause
| {ldd_event OR ldd_event OR …
| db_event OR db_event OR …
}ON {[schema.]SCHEMA | DATABASE}
}
[WHEN (condition)]{plsql_block | call_procedure_statement};

db_event ::={SERVERERROR | LOGON | LOGOFF | STARTUP |
SHUTDOWN | SUSPEND | DB_ROLE_CHANGE}

ldd_event ::={ALTER | ANALYZE | ASSOCIATE STATISTICS | AUDIT | COMMENT | CREATE | DISASSOCIATE STATISTICS | DROP | GRANT | NOAUDIT | RENAME | REVOKE | TRUNCATE | DDL }

lmd_event_clause ::=

{ DELETE | INSERT | UPDATE [OF column [, column] ...] }
[OR { DELETE | INSERT | UPDATE [OF column [, column] ...] }] ...
ON{ [schema.] table
| [NESTED TABLE nested_table_column OF][schema.]vue
}

[[referencing_clause] [FOR EACH ROW]]

8.4 Les triggers Base de données

Création d'un trigger

. Syntaxe de création d'un trigger

```
referencing_clause::=
        REFERENCING  {      OLD [ AS ] old
                        | [ NEW [ AS ] new
                    | PARENT AS parent
                        }
                       [{     OLD [ AS ] old
    | [ NEW [ AS ] new
    | PARENT [AS] parent
                }] …
```

Mots clés ou paramètres	Description
OR REPLACE	supprime le contenu du trigger et le recréé
schéma	Nom du propriétaire
trigger	Nom du trigger
BEFORE	déclencher d'abord le trigger puis exécuter l'ordre
AFTER	exécuter d'abord l'ordre puis déclencher le trigger
INSTEAD OF	Le trigger s'exécute à la place de l'évènement. Trigger valide uniquement sur les vues
lmd_event_clause	Un des évènements du langage de manipulation de données(LMD): INSERT, UPDATE, DELETE

8.4 Les triggers Base de données

Création d'un trigger

. Mots clés ou paramètres	Description
ldd_event	Un des événements du langage de définition de données: ALTER, ANALYZE, ASSOCIATE STATISTIC,| AUDIT, COMMENT, CREATE, DISASSOCIATE STATISTICS, DROP, GRANT, NOAUDIT, RENAME, REVOKE, TRUNCATE, DDL. Le mot clé DDL signifie tous les événement DDL
db_event	Un des événements Bases de Données (BD). SERVERERROR (AFTER) se déclare en cas d'erreur serveur (sauf si ora-01403, ora-01422, ora-01423, ora-01034, ora-0430) (AFTER), LOGON (AFTER) événement levé en cas de connexion d'un utilisateur, LOGOFF (BEFORE) idem pour la déconnexion; STARTUP (AFTER) se déclenche à l'ouverture de la base, SHUTDOWN (BEFORE) se déclenche avant la fermeture de la base, SUSPEND (AFTER) se déclenche à suspension de la base, DB_ROLE_CHANGE (AFTER) se déclenche en configuration data guard lors du changement du role. STARTUP et SHUTDOWN ne s'appliquent que ON DATABASE
Delete, update, insert	déclencher le trigger respectivement à la suppression, modification ou insertion.

8.4 Les triggers Base de données

Création d'un trigger

. Mots clés ou paramètres	Description
ON [user.]SCHEMA	Evénement LDD ou BD concerne le schéma de user ou le schéma de l'utilisateur courant si user n'apparaît pas
ON DATABASE	Evénement LDD ou BD concernant la base de données
on schema.table	table concernée
on schema.vue	vue concernée
FOR EACH ROW	déclenche le trigger pour chaque ligne traitée
WHEN	déclenche pour les lignes ayant satisfait la condition
referencing_clause	Permet de donner un nouveau nom aux blocs standard NEW, OLD et PARENT permettant de manipuler la courante, la ligne ancienne ou la ligne parente
PL/SQL BLOCK	bloc de code PL/SQL pour le trigger
call_procedure_statement	

8.4 Les triggers Base de données

Création d'un trigger (suite)

Limite des triggers

- **1 trigger LMD par Type par Table** (12 possibilités).
  Exemple (idem pour INSERT et DELETE):
  BEFORE UPDATE FOR EACH ROW
  BEFORE UPDATE --statement_level
  AFTER UPDATE FOR EACH ROW
  AFTER UPDATE -- statement_level

- **pas de commande DDL** dans le corps du trigger (pas de CREATE TABLE ...)

- **pas de gestion de transaction** (COMMIT, ROLLBACK, SAVEPOINT) dans le corps du trigger même pas à travers une procédure appelée

- **pas de déclaration de variables de type LONG ou LONG ROW**

- **Ordre d'évaluation des lignes pas garantie** (ne pas faire dépendre le déclenchement d'un trigger de l'ordre

Création d'un trigger BD(suite)

Exemples 1 , trigger LMD :

Création d'un trigger qui permet d'assurer la contrainte d'intégrité de mise à jour et de suppression d'une clé dans la table maître et d'assigner la clé étrangère à NULL dans la table ayant la clé étrangère.

<UPDATE - DELETE - SET NULL>

```
CREATE TRIGGER updateset
   AFTER DELETE OR UPDATE OF deptno ON dept
   FOR EACH ROW
   -- Avant de supprimer une ligne dans la table dept ou modifier la clé dans
   -- cette table. Mettre les clés étrangères à NULL
   BEGIN
      IF UPDATING AND :OLD.deptno != :NEW.deptno
         OR DELETING THEN
            UPDATE emp SET emp.deptno = NULL
              WHERE emp.deptno = :old.deptno ;
      END IF ;
   END;
   /
```

8.4 Les triggers Base de données

Création d'un trigger BD(suite)

Exemples 2, trigger LMD :

Création d'un trigger qui permet (lors d'une suppression, modification ou insertion) de mettre à jour automatiquement le montant de la commande pour chaque ligne de commande enregistrée.

```
CREATE TRIGGER modif_commande
   AFTER DELETE OR UPDATE OR INSERT ON item FOR EACH ROW
   DECLARE


BEGIN
   IF DELETING OR UPDATING OR INSERTING THEN
      UPDATE ord SET ord.total = ord.total + :NEW.itemtot - :OLD.itemtot
      WHERE ord.ordid =
         DECODE(:NEW.ordid, NULL, :OLD.ordid, :NEW.ordid);

   END IF;

END;

/
```

8.4 Les triggers Base de données

Exécution d'un trigger BD

Etapes

1. Le trigger doit être armé

2. le code du trigger sera **récompilé à sa première exécution** (code absent de la SGA) ou si modification des objets référencés dans le code du trigger (*note*: récompilation inutile depuis la 7.3)

3. Exécuter le trigger

Notes

a) un trigger doit avoir **moins de 60 lignes**

b) **utiliser les procédures pour étendre la taille** d'un trigger

c) pour un trigger ligne avec la clause WHEN, ce dernier ne sera exécuté que si la clause **WHEN** est vérifée

d) en cas d'utilisation de trigger pour alimenter une table distante, le **commit à deux phases** est assuré

e) si N trigger du même type (INSERT, UPDATE, DELETE) existe, Oracle les **exécute** tous **sans un ordre** particulier

f) un trigger ne peut lire les données impropres

g) un trigger **ne peut violer les contraintes** d'intégrité d'une table

8.4 Les triggers Base de données

Modification d'un trigger BD

- Modifier un trigger consiste à l'**activer**, **désactiver** ou **récompiler** (privilège requis : ALTER TRIGGER)

- **Syntaxe**
  ALTER TRIGGER [schema.]trigger
  {ENABLE | DISABLE | COMPILE}

- **Exemple**

  sql> ALTER TRIGGER updateset COMPILE ;
  sql> ALTER TRIGGER updateset DISABLE ;

Suppression d'un trigger

- **Privilège requis** : CREATE TRIGGER ou DROP ANY TRIGGER

- **Syntaxe**: DROP TRIGGER [schema.]trigger ;

- **Exemple** : DROP TRIGGER updateset;

8.4 Les triggers Base de données

Domaines d'applications

- fournir des **mécanismes sophistiqués d'audit**
- renforcer les **contraintes d'intégrité** non supportées par Oracle en natif
- mettre en oeuvre **des règles complexes de gestion** par exemple : changer la catégorie d'un employé si son salaire change
- **renforcer la sécurité** : interdire par exemple la modification des salaires les jours fériés et les week-end
- **assurer la réplication synchrone** à distance de tables
- **propager des actions sur d'autres tables** en fonction des événements survenus

8.4 Les triggers Base de données

Domaines d'applications (suite)

Exemples 1 : Utilisation de trigger pour l'audit

Création d'un trigger qui permet (lors d'une suppression, modification ou insertion) d'insérer dans une table journal, l'état avant et après d'une ligne de la table EMP.

```
CREATE TRIGGER audit_employe
   AFTER INSERT OR DELETE OR UPDATE ON emp FOR EACH ROW
   BEGIN
      -- La raison du déclenchement du trigger doit être passée en paramètre
      -- à la fonction ci-dessous.
      -- AUDITPACKAGE.SET_REASON(texte_de_la_raison_d'audit)


      IF auditpackage.reason IS NULL THEN
         raise_application_error(-20201, 'Une raison doit être donnée'
         || 'grâce à AUDITPACKAGE.SET_REASON(
               texte_de_la_raison_d'audit)');
      END IF ;

      --  Si la condition précédente est remplie alors ...
      INSERT INTO audit_employe VALUES
         (:old.EMPNO, :old.ENAME, :old.JOB, :old.MGR, :old.HIREDATE,
            :old.SAL, :old.COMM, :old.DEPTNO, :new.EMPNO,
            :new.ENAME, :new.JOB, :new.MGR, :new.HIREDATE,
         :new.SAL, :new.COMM, :new.DEPTNO,
            auditpackage.reasion, user, sysdate);

/
```

8.4 Les triggers Base de données

Triggers et tables mutantes

- Une table MUTANTE est une table en cours de modification via un ordre SQL INSERT, DELETE ou UPDATE

- Lors de la mise à jour d'une ligne il n'est pas possible pour la transaction en cours ou pour d'autres transactions d'accéder même avec un SELECT au données en cours de modification. ORACLE garantit la lecture cohérente lors de l'exécution d'un instruction

- L'erreur ORA-04091 est déclenchée si la lecture cohérente n'est pas garantie pour l'instruction en cours

- Documentation ayant servie pour traiter les tables mutantes
  - Article de Pomalais http://sgbd.developpez.com/oracle/ora-04091/
  - Documentation Oracle 10G
    - Sql reference manual
    - Application Developpers Guide fundamental

8.4 Les triggers Base de données

Triggers et tables mutantes

- L'erreur ORA-04091 se produit dans chacun des cas suivants :
  - Si un trigger de niveau ligne (qu'il soit BEFORE ou AFTER) tente d'accéder, même par un SELECT, à une table mutante.
  - Si un trigger de niveau instruction résultant d'une contrainte DELETE CASCADE tente d'accéder, même par un SELECT, à une table mutante.
  - Jusqu'en version Oracle 8.0.x, lire ou modifier par un trigger une clé primaire, unique ou étrangère d'une table contraignante était interdit et provoquait l'erreur ORA-04094. La notion de table contraignante a disparu depuis Oracle 8i (ce qui n'est pas sans danger dans certains cas).

Triggers et tables mutantes

- En revanche, il n'y a pas d'erreur ORA-04091 dans les cas suivants :
  - L'instruction DML déclenchante est un INSERT INTO … VALUES(…) avec valeurs littérales "en dur (donc forcément une seule ligne insérée, contrairement à un INSERT/SELECT qui pourrait traiter plusieurs lignes d'un coup)
  - Le trigger est de niveau instruction (AFTER ou BEFORE) et son exécution n'est pas due à une contrainte DELETE CASCADE.
  - Le trigger est de type INSTEAD OF.

8.4 Les triggers Base de données

Triggers et tables mutantes

- Les raisons d'être de l'erreur ORA-04091 sont :
  - **Entre deux sessions** : Oracle garantit que l'une ne peut voir les modifications faites par l'autre avant validation (Commit)
  - **Au sein d'une même transaction** : Oracle garantit, pour une instruction unitaire, qu'on ne puisse accéder aux données, tant que l'instruction unitaire n'est pas terminée. Malheureusement c'est ce qu'on tente de faire avec un trigger BEFORE ou AFTER de niveau ligne

Triggers et tables mutantes

- Exemple de trigger sur une table mutante
  - Considérant l'application de réservation de place d'avion. Nous souhaitons ne plus accepter de réservation s'il n'ya plus de place

```
drop table reservation;
drop table passagers;
drop table vol cascade constraints;

CREATE TABLE PASSAGERS(
   pas# number(8) constraint pk_psg_pas#  PRIMARY KEY ,
   NOM VARCHAR2 (40) constraint nl_pas_nom not null,
   PRENOM VARCHAR2 (40)
);

CREATE TABLE VOL(
   vOL# number(6) constraint pk_vol_vol#  PRIMARY KEY ,
   VilleDepart VARCHAR2 (40) constraint nl_vol_VilleDepart not null,
   VilleArrivee VARCHAR2 (40) constraint nl_vol_VilleArrivee not null,
   heureDepart varchar2(5)
         constraint chk_vol_hd check(heureDepart like '__:__'),
   heureArrivee varchar2(5)
         constraint chk_vol_ha check(heureArrivee like '__:__'),
   dateVol  date constraint nl_vol_dateVol not null,
   TOTPLACE number(4)constraint nl_vol_nbplace not null,
-- nombre total de places
   constraint chk_vol_ha_sup_hd check(heureArrivee>heureDepart)
);
```

8.4 Les triggers Base de données

Triggers et tables mutantes

- Exemple de trigger sur une table mutante
  - Considérant l'application de réservation de place d'avion. Nous souhaitons ne plus accepter de réservation s'il n'ya plus de place

```
CREATE TABLE RESERVATION(
   RESA# number(8) constraint pk_resa_resa# PRIMARY KEY ,
   PAS# number(6)  constraint pk_resa_pas#
REFERENCES PASSaGERS(pas#),
   VOL# number(6)  constraint pk_resa_vol# REFERENCES VOL(vol#),
   DATERESA DATE   constraint nl_resa_dateVol not null
);
-- Créations de passagers
INSERT INTO PASSAGERS
VALUES (1, 'Erzulie', 'Maria');

INSERT INTO PASSAGERS
VALUES (2, 'Ibolélé', 'Teranova');

INSERT INTO PASSAGERS
VALUES (3, 'Agoué', 'Aroyo');

INSERT INTO VOL
VALUES (300, 'Nice', 'Paris', '12:00', '13:20',
to_date('11-12-2008','DD-MM-YYYY'), 300);

INSERT INTO VOL
VALUES (310, 'Paris', 'Lyon', '11:00', '12:00',
to_date('11-12-2008','DD-MM-YYYY'), 200);
COMMIT ;
```

8.4 Les triggers Base de données

Triggers et tables mutantes

- Exemple de trigger sur une table mutante
  - Ce trigger vérifie que le nombre de place déjà réservées pour un VOL ne dépasse pas le nombre de places totales prévues pour ce VOL dans la table VOL

```
CREATE OR REPLACE TRIGGER TR_RESERVATION
 BEFORE INSERT ON RESERVATION FOR EACH
ROW
DECLARE
   TOTAL_RESA number ; -- nombre de réservations déjà
faites
   MAXPLACE number ; -- nombre de places total
BEGIN
   -- Tentative d'accès à une table mutante
   SELECT COUNT (*) INTO TOTAL_RESA
   FROM RESERVATION
   WHERE vol#=:NEW.vol#;

   SELECT TOTPLACE INTO MAXPLACE FROM VOL
   WHERE vol#=:NEW.vol#;
   IF  MAXPLACE - TOTAL_RESA < 0 THEN
      DBMS_OUTPUT.PUT_LINE('oops : Vol plein !!!');
   END IF ;
END ;
/
```

Triggers et tables mutantes

- Exemple de trigger sur une table mutante
  - Génération de l'erreur ORA-04091 en cas de tentative d'insertion de plusieurs lignes VIA INSERT INTO SELECT dans la table RESERVATION

```
-- Le passager NR 1 ERZULIE  souhaite effectuer une
-- réservation sur le vol nr 300 :
INSERT INTO RESERVATION
select 1, 1, 300, sysdate
from dual;
ERREUR à la ligne 1 :
ORA-04091: la table
TESTTRIGGER.RESERVATION est en mutation ; le
déclencheur ou
la fonction ne peut la voir
ORA-06512: à
"TESTTRIGGER.TR_RESERVATION", ligne 5
ORA-04088: erreur lors d'exécution du déclencheur
'TESTTRIGGER.TR_RESERVATION'
```

8.4 Les triggers Base de données

Triggers et tables mutantes

- Comment contourner le problème d'accès à une table mutante dans un trigger. Plusieurs approches sont possibles :
  - Approche par modification de la structure de données
  - Approche par utilisation de deux triggers avec tables temporaires
  - Approche par Utilisation d'un trigger INSTEAD OF
  - Approche par ignorance de l'erreur ORA-04091

8.4 Les triggers Base de données

Triggers et tables mutantes

• Approche par modification de la structure de données

– La démarche consiste à ajouter des colonnes supplémentaires afin d'éviter la consultation de la table MUTANTE

Exemple : Ajout de la colonne NBPLACELIBRE dans la table VOL

```
CREATE TABLE VOL(
   vOL# number(6) constraint pk_vol_vol#  PRIMARY KEY ,
   VilleDepart VARCHAR2 (40) constraint nl_vol_VilleDepart
   not null,
   VilleArrivee VARCHAR2 (40) constraint nl_vol_VilleArrivee
   not null,
   heureDepart varchar2(5)
   constraint chk_vol_hd check(heureDepart like '__:__'),
   heureArrivee varchar2(5)
   constraint chk_vol_ha check(heureArrivee like '__:__'),
   dateVol date constraint nl_vol_dateVol not null,
   TOTPLACE number(4)constraint nl_vol_nbplace not null,
-- nombre total de places
NBPLACELIBRE number(4)constraint nl_vol_nbplacelibre not
   null,
   constraint chk_vol_ha_sup_hd
   check(heureArrivee>heureDepart)
);
```

8.4 Les triggers Base de données

Triggers et tables mutantes

- **Approche par modification de la structure de données**
  - La démarche consiste à ajouter des colonnes supplémentaires afin d'éviter la consultation de la table MUTANTE

Exemple : Ajout de la colonne NBPLACELIBRE dans la table VOL

```
CREATE OR REPLACE TRIGGER TR_RESERVATION
 BEFORE INSERT ON RESERVATION FOR EACH ROW
DECLARE
   placeLibre number ; -- nombre places libres

BEGIN
   -- Contournement de la table mutante
   SELECT NBPLACELIBRE INTO placeLibre
   FROM VOL
   WHERE vol#=:NEW.vol#;
DBMS_OUTPUT.PUT_LINE('place libre : '||placeLibre);
-- Génération d'une erreur s'il n'ya plus de places
IF  placeLibre <=0 THEN
   raise_application_error(-20000, 'oops : Vol plein !!!');
ELSE
   UPDATE VOL SET NBPLACELIBRE=NBPLACELIBRE-1
   WHERE vol#=:NEW.vol#;
END IF ;

END ;
/
```

8.4 Les triggers Base de données

Triggers et tables mutantes

• Approche par modification de la structure de données

– La démarche consiste à ajouter des colonnes supplémentaires afin d'éviter la consultation de la table MUTANTE

Exemple : Ajout de la colonne NBPLACELIBRE dans la table VOL

```
-- Le passager NR 1 ERZULIE  souhaite effectuer une
-- réservation sur le vol nr 300 :
INSERT INTO RESERVATION
select 1, 1, 300, sysdate
from dual;
1 ligne créée.

-- Le passager NR 1 ERZULIE  souhaite effectuer une
-- réservation sur le vol nr 310 ou il n'ya plus de place :
INSERT INTO RESERVATION
select 2, 1, 310, sysdate
from dual;
 *
ERREUR à la ligne 1 :
ORA-20000: oops : Vol plein !!!
ORA-06512: à "UTRIGGER.TR_RESERVATION", ligne 12
ORA-04088: erreur lors d'exécution du déclencheur
   'UTRIGGER.TR_RESERVATION'
```

8.4 Les triggers Base de données

Triggers et tables mutantes

- Approche par utilisation de deux triggers avec table temporaire

  – Il s'agit d'utiliser un trigger niveau Ligne pour effectuer les actions de mise à jour et un trigger niveau Instruction pour l'accès à la table mutante

  – La solution s'appuie sur une table temporaire

8.4 Les triggers Base de données

Triggers et tables mutantes

• Approche par utilisation de deux triggers table temporaire

```
-- connect utrigger/utrigger@orcl
drop table reservation;
drop table passagers;
drop table vol cascade constraints;

CREATE TABLE PASSAGERS(
   pas# number(8) constraint pk_psg_pas#  PRIMARY KEY ,
   NOM VARCHAR2 (40) constraint nl_pas_nom not null,
   PRENOM VARCHAR2 (40)
);

CREATE TABLE VOL(
   vOL# number(6) constraint pk_vol_vol#  PRIMARY KEY ,
   VilleDepart VARCHAR2 (40) constraint nl_vol_VilleDepart not null,
   VilleArrivee VARCHAR2 (40) constraint nl_vol_VilleArrivee not null,
   heureDepart varchar2(5)
    constraint chk_vol_hd check(heureDepart like '__:__'),
   heureArrivee varchar2(5)
    constraint chk_vol_ha check(heureArrivee like '__:__'),
   dateVol    date constraint nl_vol_dateVol not null,
   TOTPLACE number(4)constraint nl_vol_nbplace not null,
   constraint chk_vol_ha_sup_hd check(heureArrivee>heureDepart)
);
```

8.4 Les triggers Base de données

Triggers et tables mutantes

- **Approche par utilisation de deux triggers table temporaire**

```
CREATE TABLE RESERVATION(
   RESA# number(8) constraint pk_resa_resa# PRIMARY KEY ,
   PAS# number(6)  constraint pk_resa_pas#
REFERENCES PASSaGERS(pas#),
   VOL# number(6)  constraint pk_resa_vol# REFERENCES VOL(vol#),
   DATERESA DATE   constraint nl_resa_dateVol not null
);
-- Créations de passagers
INSERT INTO PASSAGERS
VALUES (1, 'Erzulie', 'Maria');

INSERT INTO PASSAGERS
VALUES (2, 'Ibolélé', 'Teranova');

INSERT INTO PASSAGERS
VALUES (3, 'Agoué', 'Aroyo');

-- vol avec 10 places libres
INSERT INTO VOL
VALUES (300, 'Nice', 'Paris', '12:00', '13:20',
to_date('11-12-2008','DD-MM-YYYY'), 300);

-- vol avec 0 places libres
INSERT INTO VOL
VALUES (310, 'Paris', 'Lyon', '11:00', '12:00',
to_date('11-12-2008','DD-MM-YYYY'), 200);
COMMIT ;
```

8.4 Les triggers Base de données

Triggers et tables mutantes

- **Approche par utilisation de deux triggers table temporaire**

```
-- Création d'une table temporaire vide de même structure que
   INSCRIPTION
CREATE GLOBAL TEMPORARY TABLE
TEMP_RESERVATION AS SELECT * FROM RESERVATION
WHERE resa# is null;
-- creation d'un premier trigger de niveau ligne
-- qui stocke les lignes à insérer dans une table
-- temporaire
CREATE OR REPLACE TRIGGER TR_RESERVATION1
BEFORE INSERT ON RESERVATION FOR EACH ROW
BEGIN
   INSERT INTO TEMP_RESERVATION(RESA#, PAS#,
   VOL#, DATERESA)
VALUES (:NEW.RESA#, :NEW.PAS#, :new.vol#,
   :NEW.DATERESA);
END ;
/
```

8.4 Les triggers Base de données

Triggers et tables mutantes

- **Approche par utilisation de deux triggers table temporaire**

```
-- création du trigger niveau instruction
CREATE OR REPLACE TRIGGER TR_RESERVATION2
 AFTER INSERT ON RESERVATION
DECLARE
   placeOccupe number ; -- nombre places réservées
   NbMaxPlace number;-- nb max places
BEGIN
   -- Contournement de la table mutante
   FOR LIGNE IN (SELECT * FROM TEMP_RESERVATION ORDER
    BY DATERESA) LOOP
      SELECT COUNT (*) INTO placeOccupe FROM RESERVATION
      WHERE VOL#=LIGNE.VOL#;
      SELECT TOTPLACE INTO NbMaxPlace FROM VOL
      WHERE VOL#=LIGNE.VOL#;
      IF NbMaxPlace - placeOccupe < 0 THEN
         Raise_application_error(-20001,'Réservation impossible pour le vol
    ' || LIGNE.vol# || ' et le passager ' || LIGNE.PAS#);
         -- on supprime les réservations excédentaires
         DELETE FROM RESERVATION WHERE VOL#=LIGNE.VOL#
               AND RESA#=LIGNE.RESA#;
      END IF ;
   END LOOP ;
   -- tout à la fin, on remet à zéro la table temporaire. A noter qu'un
    TRUNCATE n'est pas possible ici, car il débuterait une nouvelle
    transaction
   DELETE FROM TEMP_RESERVATION;

END ;
/
```

8.4 Les triggers Base de données

Triggers et tables mutantes

- Approche par utilisation de deux triggers table temporaire

```
-- Le passager NR 1 ERZULIE  souhaite effectuer une
-- réservation sur le vol nr 300 :
INSERT INTO RESERVATION
select 1, 1, 300, sysdate
from dual;
1 ligne créée.
```

```
-- Le passager NR 1 ERZULIE  souhaite effectuer une
-- réservation sur le vol nr 300 :
INSERT INTO RESERVATION
select 2, 1, 310, sysdate
from dual;
```

8.4 Les triggers Base de données

Triggers et tables mutantes

- Approche par Utilisation d'un trigger INSTEAD OF
  - Un trigger Instead of n'est à poser que sur une vue
  - Il inhibe le déclenchement de l'erreur ORA-04091
  - L'accès directe à la table sur laquelle est posée le trigger provoque l'erreur ora-04091
  - Il est important de contrôler l'utilisation de se trigger

8.4 Les triggers Base de données

Triggers et tables mutantes

- **Approche par Utilisation d'un trigger INSTEAD OF**

```
-- connect utrigger/utrigger@orcl
drop table reservation;
drop table passagers;
drop table vol cascade constraints;

CREATE TABLE PASSAGERS(
   pas# number(8) constraint pk_psg_pas#  PRIMARY KEY ,
   NOM VARCHAR2 (40) constraint nl_pas_nom not null,
   PRENOM VARCHAR2 (40)
);

CREATE TABLE VOL(
   vOL# number(6) constraint pk_vol_vol#  PRIMARY KEY ,
   VilleDepart VARCHAR2 (40) constraint nl_vol_VilleDepart
    not null,
   VilleArrivee VARCHAR2 (40) constraint nl_vol_VilleArrivee
    not null,
   heureDepart varchar2(5)
    constraint chk_vol_hd check(heureDepart like '__:__'),
   heureArrivee varchar2(5)
    constraint chk_vol_ha check(heureArrivee like '__:__'),
   dateVol date constraint nl_vol_dateVol not null,
   TOTPLACE number(4)constraint nl_vol_nbplace not null,
   constraint chk_vol_ha_sup_hd
    check(heureArrivee>heureDepart)
);
```

8.4 Les triggers Base de données

Triggers et tables mutantes

- **Approche par Utilisation d'un trigger INSTEAD OF**

```
CREATE TABLE RESERVATION(
   RESA# number(8) constraint pk_resa_resa# PRIMARY
   KEY ,
   PAS# number(6)  constraint pk_resa_pas#
REFERENCES PASSaGERS(pas#),
   VOL# number(6)  constraint pk_resa_vol#
   REFERENCES VOL(vol#),
   DATERESA DATE   constraint nl_resa_dateVol not
   null
);
-- Créations de passagers
INSERT INTO PASSAGERS
VALUES (1, 'Erzulie', 'Maria');

INSERT INTO PASSAGERS
VALUES (2, 'Ibolélé', 'Teranova');

INSERT INTO PASSAGERS
VALUES (3, 'Agoué', 'Aroyo');
```

8.4 Les triggers Base de données

Triggers et tables mutantes

- **Approche par Utilisation d'un trigger INSTEAD OF**

```
-- vol avec 10 places libres
INSERT INTO VOL
VALUES (300, 'Nice', 'Paris', '12:00', '13:20',
to_date('11-12-2008','DD-MM-YYYY'), 300);

-- vol avec 0 places libres
INSERT INTO VOL
VALUES (310, 'Paris', 'Lyon', '11:00', '12:00',
to_date('11-12-2008','DD-MM-YYYY'), 200);
COMMIT ;

INSERT INTO VOL
VALUES (320, 'Paris', 'Lyon', '11:00', '12:00',
to_date('11-12-2008','DD-MM-YYYY'), 1);
COMMIT
```

8.4 Les triggers Base de données

Triggers et tables mutantes

• Approche par Utilisation d'un trigger INSTEAD OF

```
-- Création d'une vue sur la table RESERVATION pour le support des déclencheurs
    INSTEAD OF
CREATE OR REPLACE VIEW V_RESERVATION AS SELECT * FROM
    RESERVATION;


CREATE OR REPLACE TRIGGER TRIG_V_RESERVATION INSTEAD
OF INSERT ON V_RESERVATION FOR EACH ROW
DECLARE
   NB_RESERVE INTEGER ; -- nombre de réservations déjà faites
   NB_MAXPLACE INTEGER ; -- nombre de places total

BEGIN
   SELECT COUNT (*) INTO NB_RESERVE FROM V_RESERVATION
   WHERE PAS#=:NEW.pas#
   AND VOL#=:NEW.VOL#;
   SELECT TOTPLACE INTO NB_MAXPLACE FROM vol
   WHERE VOL#=:NEW.VOL#;
   IF NB_MAXPLACE - NB_RESERVE < 1 THEN
      RAISE_APPLICATION_ERROR(-20002, 'Désolé, voyage complet');

   ELSE
      -- dans un déclencheur INSTEAD OF, l'instruction DML sous-jacente ne s'exécute
     pas. On traite donc l'insertion manuellement
      INSERT INTO RESERVATION(RESA#, PAS#, VOL#, DATERESA)
     VALUES (:NEW.RESA#, :NEW.PAS#, :NEW.VOL#, :NEW.DATERESA);
   END IF ;
END ;
/
```

8.4 Les triggers Base de données

Triggers et tables mutantes

- Approche par Utilisation d'un trigger INSTEAD OF

```
set serveroutput on

INSERT INTO V_RESERVATION
select 2, 1, 320, sysdate
from dual;

INSERT INTO V_RESERVATION
select 1, 2, 320, sysdate
from dual;
            *
ERREUR à la ligne 1 :
ORA-20002: Désolé, voyage complet
ORA-06512: à "UTRIGGER.TRIG_V_RESERVATION",
   ligne 12
ORA-04088: erreur lors d'exécution du déclencheur
   'UTRIGGER.TRIG_V_RESERVATION'
```

8.4 Les triggers Base de données

Triggers et tables mutantes

- Approche par ignorance de l'erreur ORA-04091
  - L'erreur ORA-04091 est un signal pour indiquer qu'il y'a risque d'incohérence de données au niveau d'une instruction
  - Si l'on est sûr que l'alerte est sans conséquence on peut l'ignorer
  - Il faut pour cela lever une exception au niveau du trigger

8.4 Les triggers Base de données

Triggers et tables mutantes

- Approche par ignorance de l'erreur ORA-04091

```
-- connect utrigger/utrigger@orcl
drop table reservation;
drop table passagers;
drop table vol cascade constraints;

CREATE TABLE PASSAGERS(
   pas# number(8) constraint pk_psg_pas#  PRIMARY KEY ,
   NOM VARCHAR2 (40) constraint nl_pas_nom not null,
   PRENOM VARCHAR2 (40)
);

CREATE TABLE VOL(
   vOL# number(6) constraint pk_vol_vol#  PRIMARY KEY ,
   VilleDepart VARCHAR2 (40) constraint nl_vol_VilleDepart not
    null,
   VilleArrivee VARCHAR2 (40) constraint nl_vol_VilleArrivee not
    null,
   heureDepart varchar2(5)
    constraint chk_vol_hd check(heureDepart like '__:__'),
   heureArrivee varchar2(5)
    constraint chk_vol_ha check(heureArrivee like '__:__'),
   dateVol          date constraint nl_vol_dateVol not null,
   TOTPLACE number(4)constraint nl_vol_nbplace not null,
   constraint chk_vol_ha_sup_hd check(heureArrivee>heureDepart)
);
```

8.4 Les triggers Base de données

Triggers et tables mutantes

- Approche par ignorance de l'erreur ORA-04091

```
CREATE TABLE RESERVATION(
   RESA# number(8) constraint pk_resa_resa# PRIMARY
   KEY ,
   PAS# number(6)  constraint pk_resa_pas#
REFERENCES PASSaGERS(pas#),
   VOL# number(6)  constraint pk_resa_vol#
   REFERENCES VOL(vol#),
   DATERESA DATE   constraint nl_resa_dateVol not
   null
);
-- Créations de passagers
INSERT INTO PASSAGERS
VALUES (1, 'Erzulie', 'Maria');

INSERT INTO PASSAGERS
VALUES (2, 'Ibolélé', 'Teranova');

INSERT INTO PASSAGERS
VALUES (3, 'Agoué', 'Aroyo');
```

Triggers et tables mutantes

- Approche par ignorance de l'erreur ORA-04091

```
-- vol avec 10 places libres
INSERT INTO VOL
VALUES (300, 'Nice', 'Paris', '12:00', '13:20',
to_date('11-12-2008','DD-MM-YYYY'), 300);

-- vol avec 0 places libres
INSERT INTO VOL
VALUES (310, 'Paris', 'Lyon', '11:00', '12:00',
to_date('11-12-2008','DD-MM-YYYY'), 200);
COMMIT ;

INSERT INTO VOL
VALUES (320, 'Paris', 'Lyon', '11:00', '12:00',
to_date('11-12-2008','DD-MM-YYYY'), 1);
COMMIT ;
```

8.4 Les triggers Base de données

Triggers et tables mutantes

- **Approche par ignorance de l'erreur ORA-04091**

```
-- Trigger avec utilisation d'une exception
CREATE OR REPLACE TRIGGER TR_RESERVATION BEFORE
INSERT ON RESERVATION FOR EACH ROW
DECLARE
   NB_RESERVE INTEGER ; -- nombre de réservations déjà faites
   NB_MAXPLACE INTEGER ; -- nombre de places total
   TABLE_MUTANTE EXCEPTION;
   PRAGMA EXCEPTION_INIT(TABLE_MUTANTE, -4091);

BEGIN
   DBMS_OUTPUT.PUT_LINE('Tr1');

   SELECT COUNT (*) INTO NB_RESERVE FROM
    RESERVATION
   WHERE PAS#=:NEW.pas#
   AND VOL#=:NEW.VOL#;
   DBMS_OUTPUT.PUT_LINE('Tr2');

   SELECT TOTPLACE INTO NB_MAXPLACE FROM vol
   WHERE VOL#=:NEW.VOL#;
   DBMS_OUTPUT.PUT_LINE('Tr3');
```

8.4 Les triggers Base de données

Triggers et tables mutantes

- **Approche par ignorance de l'erreur ORA-04091**

```
IF NB_MAXPLACE - NB_RESERVE < 1 THEN
      -- RAISE_APPLICATION_ERROR(-20002, 'Désolé, voyage
    complet');
       DBMS_OUTPUT.PUT_LINE('Désolé, voyage complet');
    END IF ;
    DBMS_OUTPUT.PUT_LINE('Tr4');

    EXCEPTION
    WHEN TABLE_MUTANTE THEN
      DBMS_OUTPUT.PUT_LINE('Fausse alerte');
END ;
/

set serveroutput on
INSERT INTO RESERVATION
select 2, 1, 320, sysdate
from dual;

INSERT INTO RESERVATION
select 1, 2, 320, sysdate
from dual;
```

NOTE : Les insertions sont faites mais avec quel sens !!! Puisque les contrôles préconisés dans le trigger sont contournés

8.4 Les triggers Base de données

Visualisation des informations sur les triggers

- **Les vues contenant les informations sur les triggers**

. USER_TRIGGERS, ALL_TRIGGERS, DBA_TRIGGERS, USER_TRIGGER_COLS, ALL_TRIGGER_COLS, DBA_TRIGGER_COLS

Exemple

```
sql>SELECT trigger_type, triggering_event, table_name
 FROM user_triggers WHERE trigger_name = 'UPDATESET';
```

TRIGGER_TYPE	TRIGGERING_EVENT	TABLE_NAME
AFTER EACH ROW	UPDATE OR DELETE	DEPT

```
sql>SELECT trigger_body
    FROM user_triggers  WHERE trigger_name = 'UPDATESET';
```

TRIGGER_BODY

```
BEGIN
   IF UPDATING AND :OLD.deptno != :NEW.deptno
      OR DELETING THEN
         UPDATE ......
END;
/
```

9. Exercices sur les procedures stockées et les triggers

Option procédurale : package

8.1 Créer le package suivant et son body

Exemples de création de la partie spécification

```
CREATE OR REPLACE       PACKAGE gestion_employes  IS
            PROCEDURE pnouveau_sal (empid IN NUMBER, taux IN NUMBER);
            FUNCTION pemp_info(empid IN number) RETURN emp%ROWTYPE;

END gestion_employes ;
/
CREATE OR REPLACE PACKAGE BODY gestion_employes IS
PROCEDURE pnouveau_sal (empid IN NUMBER, taux IN NUMBER) IS
NO_ROW_EXECPT EXCEPTION;
BEGIN
            UPDATE emp  SET sal = sal* (1 + taux)  WHERE empno = empid ;
    IF SQL%ROWCOUNT = 0 THEN
    RAISE  NO_ROW_EXECPT;
    END IF;
    EXCEPTION
                WHEN  NO_ROW_EXECPT THEN
                DBMS_OUTPUT.PUT_LINE('Employe inexistant');
                RAISE_APPLICATION_ERROR(-20012, 'Employe Nr. ' ||
            to_char(empid) || 'Inexistant');
END pnouveau_sal;

          FUNCTION pemp_info(empid IN number)
            RETURN emp%ROWTYPE AS
             emprow emp%rowtype ;
            BEGIN
                SELECT empno, ename, job, mgr, hiredate, sal, comm, deptno
                 INTO emprow FROM emp
                WHERE empno = empid;
        RETURN (emprow);
        EXCEPTION
        WHEN NO_DATA_FOUND THEN
         DBMS_OUTPUT.PUT_LINE('Employe inexistant');
        return null;
    END pemp_info    ;
END gestion_employes;
/
```

9. Exercices sur les procedures stockées et les triggers

Option procédurale : écrire les procédures stockées suivantes

/*
1. Exercice procédure stockée 1

Ecrire une function stockée PL/SQL qui permet
d'afficher le numéro, le nom, la date de naissance,
l'adresse, le salaire et le téléphone.

Elle reçoit en paramètre un numéro de Pilote.

Elle renvoie la structure d'un pilote.

Voici le prototype de la fonction :
- Function getPiloteById(piloteId
IN number) return pilote%rowtype ;

Si le pilote n'existe lever une erreur avec
le message suivant : Le Pilote nr. X est inexistant

Tester la procédure stockée avec
les numéros suivants : 1 puis 100

*/

9. Exercices sur les procedures stockées et les triggers

Option procédurale : écrire les procédures stockées suivantes

/*
2. Exercice procédure stockée 2

Ecrire une procédure stockée PL/SQL qui permet
d'insérer un nouveau pilote dans la base.
La fonction reçoit en paramètre la structure
d'un pilote (pilote%rowtype) et ne renvoie rien.

Voici le prototype de la fonction :
- Procedure insertPilote(lignePilote
  IN PILOTE%ROWTYPE);

Vous devez effectuer un maximum de contrôle
avant d'insérer le pilote dans la base :
1) Le numéro, le nom, l'adresse et le salaire
   du pilote sont obligatoires. Pas de valeur nulle.
   Lever une exception en cas de violation de cette contrainte.
2) Le salaire doit toujours être inférieur à 70000. Lever une exception en cas de
   violation de cette erreur.
3) Lever une exception en cas de duplication du numéro ou du nom du pilote
4) Lever une exception pour toutes autres erreurs non identifiées

Tester le programme avec un pilote de nom 'Bill'.
Provoquer la levée de chacune des erreurs définies plus haut.
*/

9. Exercices sur les procedures stockées et les triggers

Option procédurale : écrire les procédures stockées suivantes

/*
3. Exercice procédure stockée 3

Ecrire une procédure stockée PL/SQL qui permet de
modifier le salaire d'un pilote connaissant son numéro.

Voici le prototype de la fonction :
- Procedure updateSalairePilote(piloteId IN number,
nouveauSal IN number);

Si le pilote n'existe pas, lever une erreur avec
le message : Le pilote nr. X n'existe pas.

Tester le programme avec les numéros suivants :
Pl#= 1 puis pl#= 200
*/

4. Exercice procédure stockée 4

Ecrire une procédure stockée qui permet de renvoyer
la référence vers un curseur contenant le numéro,
le nom, la date de naissance, l'adresse, le salaire
et le téléphone des pilotes habitant une adresse donnée.

9. Exercices procedures stockées et triggers

Option procédurale : écrire les procédures stockées suivantes

Créer pour cela un package contenant un type REF CURSOR générique :

```
CREATE OR REPLACE PACKAGE Pk_refCursType AS
TYPE REFCURSTYPE  IS REF CURSOR;
End;
/
```

Voici le prototype de la fonction :

- function getPiloteByAdr(adresse IN VARCHAR2)
  return Pk_refCursType.REFCURSTYPE ;

Tester le programme avec les villes de villes de : Paris puis Lilles.
S'il n'ya aucun pilote dans la dite ville, afficher un message d'erreur.
Pas de Pilote dans cette ville.

5. Exercice procédure stockée 5

Regrouper les fonctions et les procédures définies dans
les exercices de 1 à 4 dans un package que vous appellerez PK_PILOTE.

Tester les fonctions comme indiquées dans les exercices de 1 à 4.

9. Exercices procedures stockées et triggers

Option procédurale : triggers

Option procédurale : Utilisation de trigger pour l'audit ou d'historique

Ecrire un trigger qui mémorise dans une table d'historique HistPilote
les mises à effectuées sur la table PILOTE.

La table HistPilote a la même structure que la table PILOTE.

```
create table histPilote as  select * from pilote where pl#<0;
```

Ajouter dans cette table les TROIS colonnes
(username : nom utilisateur, majDate : date mise à jour, action: insert, update ou delete)
COMME suit :

```
ALTER TABLE histPilote Add  username varchar2(30) ;
ALTER TABLE histPilote Add  majDate date ;
ALTER TABLE histPilote
    Add  ACTION VARCHAR2(20)
    CONSTRAINT CHK_histpiloteaction
    check(action in ('INSERT', 'UPDATE', 'DELETE')) ;

CREATE or replace TRIGGER audit_PILOTE
AFTER INSERT OR DELETE OR UPDATE ON pilote FOR EACH ROW
    declare
    ACTION   VARCHAR2(20);
    BEGIN
    IF INSERTING THEN
        INSERT INTO histPilote VALUES
        (:NEW.pl#, :NEW.plnom, :NEW.dnaiss, :NEW.adr, :NEW.tel,
        :NEW.sal, :NEW.age, user, sysdate, 'INSERT');
    END IF;
    IF DELETING THEN
        INSERT INTO histPilote VALUES
        (:old.pl#, :old.plnom, :old.dnaiss, :old.adr, :old.tel,
        :old.sal, :old.age, user, sysdate, 'DELETE');
    END IF;
    IF UPDATING THEN
        INSERT INTO histPilote VALUES
        (:old.pl#, :old.plnom, :old.dnaiss, :old.adr, :old.tel,
        :old.sal, :old.age, user, sysdate, 'UPDATE');
    END IF;
end;
/
```

Effectuer des mises à jour dans la table PILOTE. Vérifier l'effet dans la table histPilote

```
insert into  pilote values(40, 'Louis', TO_date('04-11-1966','DD-MM-YYYY' ), 'Paris', NULL, 21000, 45);
```

9. Exercices sur les procedures stockées et les triggers

Option procédurale : triggers

Option procédurale : Utilisation de trigger pour assurer une règle de gestion.

Ecrire un trigger qui permet en cas d'insertion ou de modification de la ville de depart et/ ou de la ville d'arrivée d'un VOL de mettre la transaction en échec si la ville de depart et la ville d'arrivée sont identitiques.

10.Annexes

Annexe 10.1 application de référence

```
ALTER SESSION SET NLS_DATE_FORMAT='DD-MON-YYYY';
ALTER SESSION SET NLS_DATE_LANGUAGE='AMERICAN';

 REM Creation de la base de donnees aerienne

drop table pilote CASCADE CONSTRAINTS;

create table pilote(
      pl#       number(4)          constraint pk_pilote
            primary key,
      plnom    varchar2(12)      constraint
            nl_pilote_plnom not null
            constraint
            uk_pilote_plnom  unique,
      dnaiss   date        constraint
            nl_pilote_dnaiss
            not null,
      adr       varchar2(20)    default 'Paris',
      tel varchar2(12),
      sal number(7,2)    constraint nl_pilote_sal
             not null constraint
            chk_pilote_sal
            check (sal < 70000.0)
);

drop table avion CASCADE CONSTRAINTS ;

create table avion(
      av#       number(4)          constraint pk_avion  primary key,
      avtype  varchar2(10)       constraint nl_avion_avtype    not null
        CONSTRAINT chk_avion_type
        CHECK (avtype in
('A300','A310','A320','B707','B727','CONCORDE','CARAVELLE')),
      cap   number(4)   CONSTRAINT nl_avion_cap not null,
      locvarchar2(20)  CONSTRAINT nl_avion_loc not null,
      remarq       long
);
```

10.Annexes

Annexe 10.1 application de référence

```
drop table vol CASCADE CONSTRAINTS ;

create table vol(
      vol#     number(4)          CONSTRAINT pk_vol primary key,
      pilote#  number(4)          CONSTRAINT nl_vol_pilote# not null
        CONSTRAINT vol_fk_pilote REFERENCES PILOTE(PL#)
        ON DELETE CASCADE,
      avion#       number(4)     CONSTRAINT nl_vol_avion# not null,
      vd    varchar2(20),
      va    varchar2(20),
      hd    number(4)   CONSTRAINT nl_vol_hd not null,
      ha    number(4) CONSTRAINT nl_vol_ha not null,
      dat   date,
    CONSTRAINT vol_chk_ha_hd CHECK (ha>hd),
    constraint fk_vol_avion# FOREIGN KEY (avion#)
    REFERENCES AVION(AV#)
);

REM insertion des valeurs dans les tables
insert into  pilote values(1, 'Miranda', '16-AUG-1952','Sophia-Antipolis', '93548254', 18009.0);
insert into  pilote values(2, 'St-exupery', '16-OCT-1932', 'Lyon', '91548254', 12300.0);
insert into  pilote values(3, 'Armstrong ', '11-MAR-1930', 'Wapakoneta','96548254', 24500.0);
insert into  pilote values(4, 'Tintin', '01-AUG-1929', 'Bruxelles','93548254', 21100.0);
insert into  pilote values(5, 'Gagarine', '12-AUG-1934', 'Klouchino','93548454', 22100.0);
insert into  pilote values(6, 'Baudry', '31-AUG-1959', 'Toulouse','93548444', 21000.0);
insert into  pilote values(8, 'Bush', '28-FEB-1924', 'Milton','44556254', 22000.0);
insert into  pilote values(9, 'Ruskoi', '16-AUG-1930', 'Moscou','73548254', 22000.0);
insert into  pilote values(10, 'Math', '12-AUG-1938', 'Paris', '23548254', 15000.0);
insert into  pilote values(11, 'Yen', '19-SEP-1942', 'Munich','13548254', 29000.0);
insert into  pilote values(12, 'Icare', '17-DEC-1962', 'Ithaques','73548211', 17000.6);
insert into  pilote values(13, 'Mopolo', '04-NOV-1955', 'Nice','93958211', 17000.6);
insert into  pilote values(14, 'Chretien', '04-NOV-1945', ','73223322', 15000.6);
insert into  pilote values(15, 'Vernes', '04-NOV-1935', 'Paris', ',17000.6);
insert into  pilote values(16, 'Tournesol', '04-AUG-1929', 'Bruxelles',', 15000.6);
insert into  pilote values(17, 'Concorde', '04-AUG-1966', 'Paris', ',21000.6);
insert into  pilote values(18, 'Foudil', '04-AUG-1966', 'Paris', ',21000.6);
insert into  pilote values(19, 'Foudelle', '04-AUG-1966', 'Paris', ',21000.6);
insert into  pilote values(20, 'Zembla', '04-AUG-1966', 'Paris', ',21000.6);
```

10.Annexes

Annexe 10.1 application de référence

```
REM Insertion des avions
insert into  avion values(1, 'A300', 300, 'Nice', 'En service');
insert into  avion values(2, 'A300', 300, 'Nice', 'En service');
insert into  avion values(3, 'A320', 320, 'Paris', 'En service');
insert into  avion values(4, 'A300', 300, 'Paris', 'En service');
insert into  avion values(5, 'CONCORDE', 300, 'Nice', 'En service');
insert into  avion values(6, 'B707', 400, 'Paris', 'En panne');
insert into  avion values(7, 'CARAVELLE', 300, 'Paris', 'En service');
insert into  avion values(8, 'B727', 250, 'Toulouse', 'En service');
insert into  avion values(9, 'CONCORDE', 350, 'Toulouse', 'En service');
insert into  avion values(10, 'A300', 400, 'Paris', 'En service');
insert into  avion values(11, 'A300', 400, 'Paris', 'En service');
insert into  avion values(12, 'A300', 400, 'Paris', 'En service');

REM Insertion des vols
insert into  vol values(100, 1,1,'Nice', 'Paris', '1345', '1500','3-MAR-1989' );
insert into  vol values(110, 3,6,'Nice', 'Toulouse', '1230', '1325','6-MAR-1989' );
insert into  vol values(120,4,3,'Nice', 'Paris', '0745', '0900','21-JUN-1989' );
insert into  vol values(125, 12,6,'Paris', 'Nice', '1330', '1845','10-JAN-1989' );
insert into  vol values(130, 4,8,'Toulouse','Beauvais', '0630','0750', '27-MAR-1989' );
insert into  vol values(111, 5,3,'Nice', 'Paris', '0800', '0920','4-DEC-89' );
insert into  vol values(135, 8,5,'Paris', 'Toulouse', '1200','1320','22-MAR-1989' );
insert into  vol values(140, 14,9,'Lyon', 'Nice', '0700', '0750','4-JUN-1989' );
insert into  vol values(150, 1,1,'Paris', 'Nantes','1630', '1725','28-MAR-1989' );
insert into  vol values(153, 2,3,'Lyon', 'Nice', '1210', '1300','6-NOV-1989' );
insert into  vol values(156, 9,2,'Paris', 'Lyon', '0230', '0320','14-JAN-1989' );
insert into  vol values(200, 5,3,'Nice', 'Toulouse', '2030', '2125','17-JUN-1989' );
insert into  vol values(210, 14,7,'Nice', 'Nantes', '1430', '1525','14-OCT-1989' );
insert into  vol values(236, 8,4,'Lyon', 'Toulouse', '2130', '2250','15-OCT-1989' );
insert into  vol values(240, 13,10, 'Nice', 'Paris', '2300', '2355','19-NOV-1989' );
insert into  vol values(250, 13,4,'Bordeaux', 'Paris', '2300','2355', '25-DEC-89' );
insert into  vol values(260, 13,5,'Bordeaux', 'Paris', '2300','2355', '30-NOV-1989' );
insert into  vol values(270, 13,9,'Paris', 'New york', '1400','2300', '3-JAN-1989' );
insert into  vol values(280, 8,9,'Nice', 'Mulhouse', '1200','1320','21-MAR-1989' );
insert into  vol values(290, 3,8,'Beauvais', 'Marseille', '1230','1425', '9-MAR-1989' );
insert into  vol values(310, 19,8,'Beauvais', 'Marseille', '1230','1425', '9-MAR-1989' );
commit;
```