SOMMAIRE

1 Généralités

PL/SQL est un langage procédural spécifique à Oracle, alors que SQL est un langage standard non procédural, il est utilisable dans les mêmes conditions que SQL.

Un script PL/SQL comprend d'une part,

des instructions spécifiques à PL/SQL comme :

des définitions de variables,
des traitements conditionnels,
des traitements en boucle,
des traitements de curseur,
des traitements d'exception

et d'autre part,

des instructions spécifiques à SQL comme :

SELECT ...
UPDATE, INSERT, DELETE, ...
COMMIT, ROLLBACK, ...
TO_CHAR, TO_DATE, SUBSTR ...
...

2 Le bloc PL/SQL

Le moteur PL/SQL interprète un ensemble de commandes contenu dans un "bloc" PL/SQL.
Un bloc PL/SQL est articulé en trois parties :

DECLARE
    déclaration des variables, des constantes, des exceptions, des curseurs, ...
BEGIN
    instructions PL/SQL et SQL
EXCEPTION
    traitements des exceptions
END;
/ **(doit être sur le premier caractère de la dernière ligne)**

Les exemples du support utilisent deux tables définies dans le script ci-dessous :

```
DROP TABLE INVENT
;
CREATE TABLE INVENT
(NOMPROD      VARCHAR2(25),
 QUANTITE     NUMBER(6)   )
;
INSERT INTO INVENT VALUES
('Produit 1', 10);
INSERT INTO INVENT VALUES
('Produit 2', 20);
INSERT INTO INVENT VALUES
('Produit 3', 30);
DROP TABLE JOURNAL
;
CREATE TABLE JOURNAL
(TEXTE          VARCHAR2(50),
 DATETEXTE      DATE        )
;
```

REMARQUE :
Après chaque exemple, examiner (par SELECT ...) les résultats dans les tables modifiées par les traitements et lancer la commande **ROLLBACK**.

3 Les variables

Il existe 3 types de variable.

- Les variables locales : de type ORACLE, de type BOOLEEN, faisant référence au dictionnaire des données.
- Les variables de l'environnement extérieur à PL/SQL : ces variables sont préfixées par **:**.
- Les variables définies dans l'environnement d'exécution de SQL (ces variables sont préfixées par &).

3.1 Déclaration des variables

Les variables locales sont définies dans la partie DECLARE de PL/SQL.
Variable de type ORACLE :

```
DECLARE
        nom         VARCHAR2(10);
        quantite    NUMBER;
        datjour     DATE;
        prix        NUMBER(7,2);
BEGIN
...
END;
/
```

Variable de type BOOLEEN :

```
DECLARE
       reponse     BOOLEAN; -- valeur TRUE, FALSE, NULL
BEGIN
...
END;
/
```

Variable faisant référence au dictionnaire des données :

```
DECLARE
   nom     emp.nom%TYPE; -- nom, de même type que colonne nom de emp
BEGIN
...
END;
/

DECLARE
emp_enreg  emp%ROWTYPE; -- emp_enreg a même struct. qu'une
                        -- ligne de emp
BEGIN
...
END;
/
```

Chaque variable de emp_enreg a même nom et même type que la colonne de la table emp associée.

```
DECLARE
 commission  number(7,2);
 salaire     commission%TYPE; -- salaire a même type que commission
BEGIN
...
END;
/
```

3.2 Initialisation et visibilité des variables

L'initialisation se fait par l'opérateur **:=** dans la partie DECLARE, BEGIN, EXCEPTION ou SELECT ... INTO dans la partie BEGIN ou traitement d'un curseur dans la partie BEGIN

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

```
DECLARE
        nom         CHAR(10)  := 'HUGUES';
        quantite    NUMBER    := 15;
        datjour     DATE      := '20/05/02';
        reponse     BOOLEAN   := TRUE;
BEGIN
...
END;
/

DECLARE
        tauxtva     CONSTANT NUMBER(4,2) := 20;
BEGIN
...
END;
/

DECLARE
        prix    NUMBER(7,2) NOT NULL := 100;
BEGIN
...
END;
/
```

```
DECLARE
       nom_emp       emp.nom%TYPE;
       salaire       emp.salaire%TYPE;
       tx_comm       emp.tx_commission%TYPE;
       nom_dept      dept.nom%TYPE;
BEGIN
   select emp.nom, salaire, tx_commission, dept.nom
     into nom_emp, salaire, tx_comm,       nom_dept
     from emp, dept
    where nom = 'Dumas'    -- le select doit ramener une seule ligne
      and emp.nodept = dept.nodept;
...
END;
/
```

Remarque : si le select doit ramener plusieurs lignes, alors il faut utiliser un curseur.

4 Les traitements conditionnels

```
IF         condition1 THEN       traitement1;
ELSIF      condition2 THEN       traitement2;
ELSE       traitement3;
END IF;
```

Remarque : IF, THEN, END IF sont obligatoires.

/*
EX301

Objectif :
Suite à la saisie d'un nom d'employé,
si son titre est vide enregistrer l'information dans la table journal,
sinon si c'est un Representant augmenter son taux de commission de 10% et enregistrer l'information dans la table journal,
sinon mettre à jour son taux de commission à vide et enregistrer l'information dans la table journal.

Observation :
utiliser une variable de l'environnement d'exécution de SQL et une variable de PL/SQL
utiliser une structure simple de IF ... THEN ... ELSIF ... THEN ... ELSE
pas de gestion d'exception si l'employé n'existe pas

*/

```
ACCEPT nom_emp PROMPT 'Nom Employe : '
DECLARE
       fonction   VARCHAR2(25);
       texte      VARCHAR2(50);
BEGIN
       SELECT titre
         INTO fonction
         FROM emp
        WHERE nom = '&nom_emp';
   IF       fonction IS NULL
   THEN     texte := '&nom_emp'||' n''a pas de fonction !';
   ELSIF    fonction = 'Representant'
   THEN     UPDATE emp
               SET tx_commission = tx_commission * 1.10
             WHERE nom = '&nom_emp';
            texte := '&nom_emp'||' tx commission augmente de 10 %';
   ELSE     UPDATE emp
               SET tx_commission = null
             WHERE nom = '&nom_emp';
            texte := '&nom_emp'||' commission vide';
   END IF;
   INSERT into journal VALUES (texte, sysdate);
END;
/
```

5 Les traitements en boucle

5.1 Boucle de base

```
/*
EX302
Objectif :
Enregistrer dans la table INVENT 10 lignes contenant les nombres de 1 à 10.
Observation :
Utiliser une boucle de base LOOP ... END LOOP
*/

DECLARE
       nombre NUMBER := 1;
BEGIN
   LOOP
       INSERT INTO invent (quantite)
       VALUES (nombre);
       nombre := nombre + 1;
       EXIT WHEN nombre > 10;
   END LOOP;
END;
/
```

5.2 Boucle FOR

```
/*
EX303
Objectif :
Enregistrer dans la table INVENT le résultat de factorielle 7.
Observation :
Solution 1 - Utiliser une boucle FOR ... LOOP ... END LOOP et les produits 1x2x3 ... x7
Solution 2 - Utiliser une boucle FOR ... LOOP ... END LOOP et les produits 7x6x5 ... x1
*/

DECLARE
       facteur      NUMBER := 1;
       factorielle  NUMBER := 7;
BEGIN
   FOR i IN 1..factorielle
   LOOP
       facteur := facteur * i;
   END LOOP;
   INSERT into invent (quantite,nomprod)
   VALUES (facteur,'FACTORIELLE '||to_char(factorielle));
END;
/
```

```
DECLARE
       facteur      NUMBER := 1;
       factorielle  NUMBER := 7;
BEGIN
   FOR i IN reverse 1..factorielle
   LOOP
       facteur := facteur * i;
   END LOOP;
   INSERT into invent (quantite,nomprod)
   VALUES (facteur,'FACTORIELLE '||to_char(factorielle));
END;
/
```

5.3 Boucle WHILE

/*

EX304

Objectif :

Enregistrer dans la table INVENT le résultat du calcul du reste de la division de 3278 par 9.

Observation :

Utiliser une boucle WHILE ... LOOP ... END LOOP

*/

```
DECLARE
    reste NUMBER := 3278;
BEGIN
    WHILE reste >= 9
    LOOP
       reste := reste - 9;
    END LOOP;
    INSERT INTO invent (quantite,nomprod)
    VALUES (reste,'Reste 3278/9');
END;
/
```

6 Les traitements des curseurs

Un curseur est une zone mémoire de taille fixe utilisée par ORACLE pour analyser, interpréter et exécuter tout ordre SQL.

Curseur implicite : curseur généré et géré par ORACLE pour chaque ordre SQL.

Curseur explicite : curseur déclaré, généré et géré dans un bloc PL/SQL pour traiter un SELECT qui ramène plusieurs lignes.

L'utilisation d'un curseur explicite se fait en quatre étapes :

Déclaration, ouverture, traitements des lignes, fermeture.

6.1 Utilisation d’un curseur explicite

/*
EX305
Objectif :
Enregistrer dans la table invent le nom et le salaire des employés du département 41 dont le salaire est supérieur à 1830.
Observation :
Utiliser un curseur déclaré et la structure OPEN ... LOOP FETCH ... END LOOP ... CLOSE
*/

```
DECLARE
   nom       emp.nom%TYPE;
   salaire   emp.salaire%TYPE;
   CURSOR serv_41 is
          SELECT   nom, salaire
          FROM     emp
          WHERE    nodept = 41
          ORDER BY salaire;
BEGIN
   OPEN serv_41;
     LOOP
       FETCH serv_41 INTO nom, salaire;
       IF salaire > 1830
       THEN
            INSERT INTO invent (nomprod,quantite)
            VALUES (nom, salaire);
       END IF;
       EXIT WHEN salaire = 5300;
     END LOOP;
   CLOSE SERV_41;
END;
/
```

/*
EX306
Objectif :
Suite à la saisie d'un nombre n, enregistrer dans la table invent les n plus hauts salaires de la table emp.
Observation :
Utiliser un curseur déclaré et la structure OPEN ... FOR ... LOOP FETCH ... END LOOP ... CLOSE

*/

```
ACCEPT nombre PROMPT 'Nombre de salaires : '
DECLARE
       nom       emp.nom%TYPE;
       salaire   emp.salaire%TYPE;
   CURSOR c1 is
          SELECT   nom, salaire
          FROM     emp
          ORDER BY salaire DESC;
BEGIN
   OPEN c1;
   FOR i IN 1..&nombre
     LOOP
       FETCH c1 INTO nom, salaire;
       INSERT INTO invent (nomprod,quantite)
       VALUES (nom, salaire);
     END LOOP;
   CLOSE c1;
END;
/
```

6.2 Les attributs d'un curseur

Les attributs d'un curseur donnent des indications sur l'état du curseur.

Attribut	Type	Signification
%FOUND	booléen	curseur%FOUND si le dernier FETCH a ramené une ligne alors curseur%FOUND = TRUE
%NOTFOUND	booléen	curseur%NOTFOUND si le dernier FETCH a ramené aucune ligne alors curseur%NOTFOUND = TRUE
%ISOPEN	booléen	curseur%ISOPEN si le curseur est ouvert alors curseur%ISOPEN = TRUE
%ROWCOUNT	numérique	curseur%ROWCOUNT curseur%ROWCOUNT = le rang de la dernière ligne ramenée par le FETCH

/*

EX307

Objectif : Enregistrer dans la table invent le nom et le salaire des employés du département 41 qui ont un salaire supérieur à 1830.

Observation : Utiliser un curseur déclaré. Sortir de la boucle en utilisant le statut du curseur%NOTFOUND.

*/

```
DECLARE
   nom       emp.nom%TYPE;
   salaire   emp.salaire%TYPE;
   CURSOR serv_41 is SELECT nom, salaire FROM emp WHERE nodept = 41;
BEGIN
   OPEN serv_41;
     LOOP
       FETCH serv_41 INTO nom, salaire;
       EXIT WHEN serv_41%NOTFOUND;
         IF salaire > 1830 THEN
            INSERT INTO invent (nomprod,quantite) VALUES (nom, salaire);
         END IF;
     END LOOP;
   CLOSE SERV_41;
END;
/
```

/*

EX308

Objectif : Enregistrer dans la table invent le nom et le salaire des employés du département 41 dont le salaire est supérieur à 2200. Si leur nombre est inférieur ou égal à 2, les enregistrer tous sinon enregistrer que 2 lignes.

Observation : Tenir compte de la situation où il y a aucun employé satisfaisant les conditions.

*/

```
DECLARE
   nom       emp.nom%TYPE;
   salaire   emp.salaire%TYPE;
   CURSOR serv_41 is
          SELECT   nom, salaire FROM emp
          WHERE    nodept = 41
            AND    salaire > 2200;
BEGIN
   OPEN serv_41;
     LOOP
       FETCH serv_41 INTO nom, salaire;
       EXIT WHEN serv_41%NOTFOUND
              OR serv_41%ROWCOUNT > 2;
         INSERT INTO invent (nomprod,quantite)
                     VALUES (nom, salaire);
     END LOOP;
   CLOSE SERV_41;
END;
/
```

6.3 Simplifier l'écriture

/*
EX309
Objectif :
Enregistrer dans la table invent le nom et la rémunération totale (salaire + salaire * tx_commission) des employés dont la rémunération totale est supérieure à 2135.
Observation :
Utiliser un curseur déclaré et une variable de type curseur%ROWTYPE
*/

```
DECLARE
   CURSOR c1 is
   SELECT   nom, salaire*(1+nvl(tx_commission,0)/100) Remuneration
   FROM     emp;
   c1_enreg  c1%ROWTYPE;  -- la structure de c1_enreg est identique
                          -- aux colonnes ramenées par le curseur c1
BEGIN
   OPEN c1;
     LOOP
       FETCH c1 INTO c1_enreg;        -- c1_enreg est renseigne par
       EXIT WHEN c1%NOTFOUND;         -- le FETCH
       IF c1_enreg.Remuneration > 2135 -- un element de c1_enreg
       THEN
            INSERT INTO invent (nomprod,quantite)
            VALUES (c1_enreg.nom, c1_enreg.Remuneration);
       END IF;
     END LOOP;
   CLOSE c1;
END;
/
```

/*
EX310
Objectif : Enregistrer dans la table invent le nom et le salaire des employés dont le salaire est supérieur à 4000.
Observation : Utiliser un curseur déclaré et une boucle du type FOR ... IN curseur.
*/

```
DECLARE
   CURSOR sal_4000 is
      SELECT   nom, salaire FROM emp WHERE    salaire > 4000;
BEGIN
   FOR enreg IN sal_4000
     LOOP
          INSERT INTO invent (nomprod,quantite)
          VALUES (enreg.nom, enreg.salaire);
     END LOOP;
END;
/
```

est équivalent à :

/*
EX311
Objectif :
Enregistrer dans la table invent le nom et le salaire des employés dont le salaire est supérieur à 4000.
Observation :
Ex310 est équivalent à Ex311, mais içi, écriture sans simplification
*/

```
DECLARE
   CURSOR sal_4000 is
          SELECT   nom, salaire
          FROM     emp
          WHERE    salaire > 4000;
   enreg  sal_4000%ROWTYPE;
BEGIN
   OPEN sal_4000;
     LOOP
       FETCH sal_4000 INTO enreg;
       EXIT WHEN sal_4000%NOTFOUND;
            INSERT INTO invent (nomprod,quantite)
            VALUES (enreg.nom, enreg.salaire);
     END LOOP;
   CLOSE sal_4000;
END;
/
```

/*
EX312
Objectif : Enregistrer dans la table invent le nom et le salaire des employés dont le salaire est supérieur à 4000.
Observation : Utiliser une boucle FOR ... IN et un curseur non déclaré de manière explicite.
*/

```
BEGIN
   FOR enreg IN  (SELECT   nom, salaire
                     FROM   emp
                    WHERE   salaire > 4000)
     LOOP
          INSERT INTO invent (nomprod,quantite)
          VALUES (enreg.nom, enreg.salaire);
     END LOOP;
END;
/
```

/*
EX313

Objectif : Enregistrer dans la table invent le nom et le salaire des employés dont le salaire est égal à l'un des n (à saisir) plus hauts salaires.
Observation : Utiliser deux curseurs déclarés dont l'un est un curseur paramétré, et deux boucles FOR imbriquées.
*/

```
ACCEPT nombre PROMPT 'Nombre de salaires : '
DECLARE
   CURSOR csal IS
       SELECT DISTINCT salaire    -- les differents salaires de emp
         FROM emp
       ORDER BY salaire desc;
   CURSOR cemp (param_sal NUMBER) is
       SELECT   nom, salaire      -- les employes qui ont un salaire
         FROM   emp               -- donne
        WHERE   salaire = param_sal;
BEGIN
 FOR sal IN csal             -- pour chaque salaire different de emp
 LOOP
   EXIT WHEN csal%ROWCOUNT > &nombre;  -- seulement les n premiers
   FOR employe IN cemp (sal.salaire)   -- pour les employes qui ont
     LOOP                              -- ce salaire
         INSERT INTO invent (nomprod,quantite)
         VALUES (employe.nom, employe.salaire);
     END LOOP;
 END LOOP;
END;
/
```

7 La gestion des exceptions

Les exceptions permettent de gérer des situations spécifiques qui sont soit des situations internes à ORACLE (tentative de division par zéro, tentative d'utilisation d'une valeur en doublon sur un index unique, etc. ...), soit des situations contrôlées dans le script PL/SQL.

/*
EX314
Objectif :
Augmenter de 5% le taux de commission des employés en respectant les règles suivantes :
si la fonction de l'employé est vide, détecter cette situation avec une EXCEPTION déclarée et mettre à jour la table journal,
si la fonction = 'Representant', faire la mise à jour et mettre à jour la table journal,
si la fonction <> 'Representant', modifier le taux de commission à vide et mettre à jour la table journal.
Observation :
Utilisation de la notion d'exception.
*/

```
ACCEPT nom_emp PROMPT 'Nom employe : '
DECLARE
       fonction       VARCHAR2(25);
       texte          VARCHAR2(40);
       fonction_vide  EXCEPTION;      -- nom de l'exception
BEGIN
   SELECT titre
     INTO fonction
     FROM emp WHERE nom = '&nom_emp';
   IF          fonction IS NULL          -- situation à contrôler
   THEN        RAISE fonction_vide;
   ELSIF       fonction = 'Representant'
   THEN        UPDATE emp
               SET    tx_commission = tx_commission * 1.05
               WHERE  nom = '&nom_emp';
               texte := '&nom_emp'||' commission augmentee de 5 %';
   ELSE        UPDATE emp SET tx_commission = null
               WHERE  nom = '&nom_emp';
               texte := '&nom_emp'||' commission vide';
   END IF;
   INSERT into journal VALUES (texte, sysdate);
EXCEPTION
WHEN fonction_vide
THEN texte := '&nom_emp'||' n''a pas de fonction !'; -- traitement
     INSERT into journal VALUES (texte, sysdate);    -- de l'exception
END;
/
```

/*
EX315
Objectif:
Enregistrer dans la table dept un nouveau département. Contrôler la taille du no de département par rapport à la structure de la colonne nodept de la table dept (NUMBER(7)).
Observation :
Utiliser une exception déclarée et définie avec la commande PRAGMA EXCEPTION_INIT.
*/

```
accept noservice  prompt 'Numero service : '
accept nomservice prompt 'Nom service : '
accept noregion   prompt 'No region : '
DECLARE
   noservice_tropgrand    EXCEPTION;
   PRAGMA EXCEPTION_INIT (noservice_tropgrand, -1438);
BEGIN
   INSERT INTO dept
   values (&noservice,initcap('&nomservice'),&noregion,null);
EXCEPTION
WHEN noservice_tropgrand
THEN INSERT INTO journal(texte)
     values ('Le numero '||to_char(&noservice)||' est trop grand.');
END;
/
```

Exceptions prédéfinies non déclarées :

Nom de l'exception	No de l'erreur gérée dans Oracle
CURSOR_ALREADY_OPEN	-6511
DUP_VAL_ON_INDEX	-1
INVALID_CURSOR	-1001
INVALID_NUMBER	-1722
LOGIN_DENIED	-1017
NO_DATA_FOUND	+100
NOT_LOGGED_ON	-1012
PROGRAM_ERROR	-6501
STORAGE_ERROR	-6500
TIMEOUT_ON_RESOURCE	-51
TOO_MANY_ROWS	-1427
VALUE_ERROR	-6502
ZERO_DIVIDE	-1476
OTHERS	toutes les autres
...	...

8 Les fonctions spécifiques à PL/SQL

SQLCODE : retourne le code de l'erreur courante
SQLERRM : retourne le texte de l'erreur courante

/*
EX316
Objectif :
Enregistrer dans la table journal le code d'erreur et le message associé, générés par le moteur SQL.
Observation :
Utiliser l'exception non déclarée OTHERS, et les fonctions SQLCODE et SQLERRM.
*/

```
DECLARE
   nom      VARCHAR2(10);
   code     NUMBER;
   libel    VARCHAR2(50);
BEGIN
   SELECT nom
   INTO   nom
   FROM   emp
   WHERE  noemp = 9999;    -- employe qui n'existe pas
EXCEPTION
WHEN OTHERS
THEN code  := SQLCODE;
     libel := SQLERRM;
     INSERT INTO journal(texte)
     VALUES (to_char(code)||' '||substr(libel,1,30));
END;
/
```

9 Compléments

/*
EX317
Objectif :
Créer un script qui permet de mettre à jour le salaire d'un employé donné en fonction de la moyenne des salaires des employés qui ont le même titre. Si l'employé a un salaire plus grand ou égal à la moyenne, l'augmentation est de 5 %, sinon le nouveau salaire est égal à la moyenne.
Observation :
Afficher les données avant mise à jour et après mise à jour.
*/

```
accept nom prompt 'Entrer le nom d''un employe : '
set verify off
SELECT nom "Nom", salaire "Salaire actuel", titre "Titre"
  FROM emp WHERE nom = initcap('&nom');
SELECT avg(salaire) "Salaire moyen pour ce titre"
FROM emp WHERE titre = (SELECT titre FROM emp
                        WHERE nom = initcap('&nom'));

DECLARE
   titre_emp    emp.titre%type;
   sal_moy      emp.salaire%type;
   sal_emp      emp.salaire%type;
BEGIN
   SELECT salaire, titre
     INTO sal_emp, titre_emp
     FROM emp WHERE nom = initcap('&nom');
   SELECT avg(salaire)
     INTO sal_moy FROM emp WHERE titre = titre_emp;
   IF      sal_emp >= sal_moy
      THEN sal_emp := sal_emp * 1.05;
      ELSE sal_emp := sal_moy;
   END IF;
   UPDATE emp
      SET salaire = sal_emp
    WHERE nom = initcap('&nom');
END;
/

SELECT nom "Nom", salaire "Nouveau salaire"
  FROM emp  WHERE nom = initcap('&nom');
SELECT avg(salaire) "Salaire moyen pour ce titre"
  FROM emp WHERE titre = (SELECT titre FROM emp
                          WHERE nom = initcap('&nom'));
```

/*
EX318
Objectif :
Créer un script qui permet d'afficher le n éme et le n+1 ème plus haut salaire, en ne tenant pas compte des doublons sur salaire.
*/

```
accept rg prompt 'Rang du salaire considere : '
DECLARE
  v_nom    emp.nom%type;
  v_sal    emp.salaire%type;
  CURSOR c1 IS SELECT nom, salaire FROM emp ORDER BY salaire desc;
BEGIN
 OPEN c1;
  LOOP
   FETCH c1 INTO v_nom, v_sal;
   EXIT WHEN c1%NOTFOUND OR c1%ROWCOUNT > &rg + 1;
   IF    c1%ROWCOUNT = &rg     THEN INSERT INTO journal(texte)
   values (to_char(&rg)||'eme salaire : '||v_nom||' '|| to_char(v_sal));
   ELSIF c1%ROWCOUNT = &rg + 1 THEN INSERT INTO journal(texte)
   values (to_char(&rg+1)||'eme salaire : '||v_nom||' '|| to_char(v_sal));
   END IF;
  END LOOP;
 CLOSE C1;
END;
/
```

/*
EX318b
Objectif :
Modifier le script précédent pour tenir compte des doublons sur salaire.
*/

```
accept rg prompt 'Rang du salaire considere : '
DECLARE
   CURSOR c1 IS SELECT distinct salaire FROM emp ORDER BY salaire desc;
   v_sal     emp.salaire%type;
BEGIN
  OPEN c1;
  LOOP
   FETCH c1 INTO v_sal;
   EXIT WHEN c1%NOTFOUND OR c1%ROWCOUNT > &rg + 1;
   IF    c1%ROWCOUNT = &rg     THEN INSERT INTO journal(texte)
        select to_char(&rg)||'eme salaire : '|| nom||' '||to_char(v_sal)
          from emp where salaire = v_sal;
   ELSIF c1%ROWCOUNT = &rg + 1 THEN INSERT INTO journal(texte)
        select to_char(&rg+1)||'eme salaire : '|| nom||' '||to_char(v_sal)
          from emp where salaire = v_sal;
   END IF;
  END LOOP;
  CLOSE C1;
END;
/
```

10 Les procédures et les fonctions

Les procédures (objets qui effectuent des traitements) et fonctions (objets qui retournent une valeur) sont des unités de programme, écrites en PL/SQL, stockées au niveau du dictionnaire de la base et gérées comme des objets Oracle (tables, vues, …).

- Stockées sous forme texte (source) et pseudo compilée (pseudo code, P-code)

- Utilisables dans différents outils Oracle (SQL, …)

- Manipulation riche des données à travers une instruction

- Maintenance centralisée pour toutes les applications faisant référence à l'objet

- Accès aux procédures en fonction du profil utilisateur

- ...

Création :

```
/*
EX401
Objectif :
Créer une procédure qui permet d'enregistrer dans une table prévue à cet effet le nom d'un
employé donné responsable d'un ou plusieurs clients, son titre, le nom de sa région, son
chiffre d'affaire total et la date d'enregistrement dans la table.
Observation :
Utilisation d'une procédure de mise à jour d'une table qui fait appel à une fonction spécifique
de calcul.
*/

CREATE TABLE table_chaff
      (nomemp               VARCHAR2(25),
       titre                VARCHAR2(25),
       nomregion            VARCHAR2(50),
       chaff                NUMBER,
       dat_ins              DATE);

CREATE OR REPLACE FUNCTION Chiffre_aff (par_noemp NUMBER)
RETURN NUMBER IS
      Ca                    NUMBER ;
BEGIN
      SELECT SUM(quantite_livree * prix) INTO Ca
        FROM emp, client, com, lgncom
       WHERE emp.noemp = client.novendeur
         AND client.noclient = com.noclient
         AND com.nocom = lgncom.nocom
         AND emp.noemp = par_noemp ;
      RETURN Ca ;
END ;
/

CREATE OR REPLACE PROCEDURE nom_caf (par_noemp NUMBER) IS
      Varnomemp       VARCHAR2(25) ;
      Vartitre        VARCHAR2(25) ;
      Varnomregion    VARCHAR2(50) ;
      Varchaff        NUMBER ;
BEGIN
      SELECT emp.nom, emp.titre, region.nom
        INTO Varnomemp, Vartitre, Varnomregion
        FROM emp, dept, region
       WHERE emp.nodept = dept.nodept
         AND dept.noregion = region.noregion
         AND emp.noemp = par_noemp ;
      Varchaff := Chiffre_aff(par_noemp) ;
      INSERT INTO table_chaff
      VALUES (Varnomemp, Vartitre, Varnomregion, Varchaff, sysdate) ;
END ;
/
```

Suppression :

DROP FUNCTION Chiffre_aff ;
DROP PROCEDURE Nom_caf ;

Utilisation :

Dans l'environnement d'exécution de SQL : EXECUTE nom_caf (11) ;
A partir d'un bloc PL/SQL : `Varchaff := Chiffre_aff(par_noemp) ;`

Erreurs de syntaxe à la création ou à la modification :

Dans l'environnement d'exécution de SQL : SHOW ERRORS

Traçage des erreurs d'exécution :

Dans l'environnement d'exécution de SQL,
SET SERVEROUTPUT ON et utilisation du package DBMS_OUTPUT. PUT_LINE

Exceptions PL/SQL

Procédure RAISE_APPLICATION_ERROR (no_erreur, message) avec no_erreur compris entre –20000 et –20999, et message, texte de 2ko maximum.

/*
EX402
Objectif :
Créer une procédure qui permet de mettre à jour le salaire d'un employé. Contrôler l'existence de l'employé et le taux d'augmentation qui doit être compris entre 1 et 1.1 .
Observation :
Utiliser le statut RAISE_APPLICATION_ERROR et la gestion des messages associée.
*/

```
CREATE OR REPLACE
PROCEDURE augment_salaire (par_noemp NUMBER,par_coeff NUMBER) IS
test_existe NUMBER(1);
BEGIN
SELECT 1 INTO test_existe FROM emp WHERE noemp = par_noemp;
     BEGIN
     IF nvl(par_coeff,0) between 1 and 1.1
     THEN UPDATE emp
          SET salaire = salaire * par_coeff WHERE noemp = par_noemp ;
     ELSE RAISE_APPLICATION_ERROR
     (-20001,'Le coefficient d''augmentation doit être compris entre 1 et 1.1') ;
     END IF ;
END ;
EXCEPTION
WHEN NO_DATA_FOUND THEN RAISE_APPLICATION_ERROR
     (-20002,'L''employé '||TO_CHAR(par_noemp)||' n''existe pas') ;
END;
/
```

Vues du dictionnaire

USER_SOURCE

11 Les packages

Les packages sont des objets, stockées au niveau du dictionnaire de données. Ils permettent d'encapsuler en un seul module des unités de traitements (procédures, fonctions, variables, curseurs, …) fonctionnellement dépendants.

- Composées de deux parties : la SPECIFICATION et le CORPS

- Stockées dans le dictionnaire de données (source et pseudo code)

- Les unités de traitement qui les composent sont écrites en PL/SQL et peuvent faire référence à des procédures et fonctions cataloguées

- Les procédures, fonctions, ..., qu'ils encapsulent peuvent être appelées à partir de différents outils Oracle et à partir d'autres procédures et fonctions cataloguées

- Amélioration de la modularité des traitements

- Le travail de maintenance et de gestion des privilèges sur les objets est facilité.

Spécification du package

Permet de déclarer les objets publics (procédures, fonctions, curseurs, exceptions, variables, constantes, tableaux)
accessibles aux utilisateurs ayant des droits sur le package.

Corps du package

Permet de définir le corps des procédures et fonctions publiques
accessibles aux utilisateurs ayant des droits sur le package
et le corps des procédures et fonctions privées
non accessibles aux utilisateurs ayant des droits sur le package
mais utilisées dans le corps des procédures et fonctions publiques.

/*
EX403
Objectif :
Création d'un package qui permet de définir une variable, une procédure à utiliser pour une embauche et une procedure à utiliser pour un départ.
*/

```
/* SPECIFICATION du package gest_pers */

CREATE OR REPLACE PACKAGE gest_pers IS
     Salaire_moyen NUMBER ;
     PROCEDURE embauche (par_noemp NUMBER, par_nosupr NUMBER,
                         par_nom VARCHAR2, par_nodept NUMBER,
                         par_titre VARCHAR2) ;
     PROCEDURE depart   (dpar_noemp NUMBER) ;
END ;
/

CREATE TABLE archive_depart
     (noemp          NUMBER(7)
     ,nom            VARCHAR2(25)
     ,titre          VARCHAR2(25)
     ,date_depart    date       );

/* SPECIFICATION du CORPS du package gest_pers */

CREATE OR REPLACE PACKAGE BODY gest_pers IS
     PROCEDURE embauche (par_noemp NUMBER, par_nosupr NUMBER,
                         par_nom VARCHAR2, par_nodept NUMBER,
                         par_titre VARCHAR2) IS
     BEGIN
     INSERT INTO emp (noemp, nosupr, nom, nodept, titre)
     Values(par_noemp, par_nosupr, par_nom, par_nodept, par_titre) ;
     END ;
     PROCEDURE depart (dpar_noemp NUMBER) IS
     BEGIN
     INSERT INTO archive_depart
     (SELECT noemp,nom,titre,sysdate FROM emp WHERE noemp = dpar_noemp) ;
     DELETE FROM emp WHERE noemp = dpar_noemp ;
     END ;
END ;
/
```

Suppression :
DROP PACKAGE BODY gest_pers ; -- corps uniquement
DROP PACKAGE gest_pers ; -- spécification et corps

Utilisation
Dans l'environnement d'exécution de SQL :
EXECUTE gest_pers.embauche(55, 1, ‘DURAND’, 50, ‘Secretaire’) ;
A partir d’un bloc PL/SQL.

12 Les triggers

Un trigger est un traitement écrit avec PL/SQL, associé à un évènement donné.

Les exemples cités ci-dessous concernent les mises à jour d'une table. Ce traitement est stocké au niveau du dictionnaire de données et est déclenché automatiquement à chaque fois que l'évènement de mise à jour auquel il est associé a lieu sur cette table.

- Associé, par exemple, à un ordre de mise à jour sur la table : INSERT, UPDATE, DELETE

- Peut faire appel à des procédures, fonctions et/ou packages

Déclenchement	Type	INSERT	UPDATE	DELETE
Une fois avant la mise à jour de la table	BEFORE	**x**	**x**	**x**
Une fois avant la mise à jour de chaque ligne de la table	BEFORE FOR EACH ROW	**x**	**x**	**x**
Une fois après la mise à jour de chaque ligne de la table	AFTER FOR EACH ROW	**x**	**x**	**x**
Une fois après la mise à jour de la table	AFTER	**x**	**x**	**x**
...	...			

/*
EX404
Objectif :
Créer un trigger rattaché à la table emp qui permet de contrôler la valeur du salaire qui ne doit pas être supérieur ou égal à deux fois le salaire moyen.
Observation :
Utiliser la variable définie dans le package gest_pers.
*/

```
CREATE OR REPLACE TRIGGER tr_salaire_moyen
BEFORE INSERT OR UPDATE OF salaire ON emp
BEGIN
     SELECT AVG(salaire) INTO gest_pers.salaire_moyen
     FROM emp ;
END ;
/


CREATE OR REPLACE TRIGGER tr_salaire_controle
BEFORE INSERT OR UPDATE OF salaire ON emp
FOR EACH ROW
BEGIN
     IF (:NEW.salaire/gest_pers.salaire_moyen) >= 2
     THEN RAISE_APPLICATION_ERROR
     (-20001, 'Augmentation trop importante.') ;
     END IF ;
END ;
/
```

Suppression

```
DROP TRIGGER tr_salaire_controle ;
```

Déclenchement en cascade

Jusqu'à 32 déclencheurs déclenchés en cascade.

Prédicats de mise à jour

Si l'événement associé au trigger est une combinaison (INSERT OR UPDATE …) il est possible de tester les prédicats INSERTING, UPDATING, DELETING

```
IF INSERTING
THEN...
END IF ;
```

Qualificateurs de colonnes

:NEW.nom_colonne	nouvelle valeur de nom_colonne lors d'une insertion ou d'une mise à jour
:OLD.nom_colonne	ancienne valeur de nom_colonne lors d'une mise à jour
:NEW.nom_colonne	NULL lors d'une suppression
:OLD.nom_colonne	NULL lors d'une insertion

Utilisation

- Déduire des valeurs de colonnes par calcul

- Eviter que des transactions logiquement invalides aient lieu

- Mettre en œuvre des sécurités complexes

- Mettre en œuvre des audits perfectionnés

Vues du dictionnaire

USER_TRIGGERS

13 Le modèle physique