

- Initialisation des variables
 - Elle peut se faire par :
 - L'ordre **SELECT ... INTO ...** dans la section **BEGIN**
 - Si le SELECT retourne 1 ligne => C'est ok
 - Si le SELECT retourne 0 ligne => NO_DATA_FOUND
 - » erreur PL/SQL générée (on doit utiliser les curseurs)
 - Si le SELECT retourne plusieurs lignes => TOO_MANY_ROWS
 - » erreur PL/SQL générée (on doit utiliser les curseurs)

Variables

- Initialisation des variables
 - Elle peut se faire par :
 - Le traitement d' un curseur (à voir plus tard)

Variables

- Visibilité d' une variable
 - Dans son bloc de déclaration
 - Dans les blocs imbriqués (si elle n' a pas été redéfinie)

```
DECLARE
    num_emp CHAR(15);
    Salaire NUMBER;
BEGIN
    ...
    DECLARE
        num_emp NUMBER;
        Commission NUMBER;
    BEGIN
        ...
    END
    ....
END;
```

num_emp (CHAR) et Salaire

num_emp (NUMBER), Salaire, Commission

num_emp (CHAR) et Salaire

Traitements conditionnels

- Définition
 - Exécution d'instructions en fonction du résultat d'une condition
- Syntaxe

```
IF conditioni THEN  
    traitementi;  
[ELSIF conditionj THEN  
    traitementj;  
[ELSE  
    traitementk;  
END IF;
```

Opérateurs utilisés dans les conditions sont :

=, <, <=, >, >=, !=, IS NULL, IS NOT NULL, BETWEEN, LIKE, AND, OR, etc.

Traitements conditionnels

- Exemple

```
DECLARE
    emploi CHAR(10);
    nom VARCHAR2(10) := 'MILLER';
    mes CHAR(30);
BEGIN
    SELECT job INTO emploi FROM emp WHERE ename = nom;
    IF emploi IS NULL THEN mes := nom || ' n"a pas d"emploi';
        ELSIF emploi = 'CLERK' THEN
            UPDATE emp SET comm = 1000 WHERE ename = nom;
            mes := nom || ' commission modifiée';
        END IF;
    dbms_output.put_line(mes);
COMMIT;
END;
/
```

Traitements conditionnels

- Vous pouvez également utiliser *goto*

```
DECLARE
    emploi CHAR(10);
    nom VARCHAR2(10) := 'MILLER';
    mes CHAR(30);
BEGIN
    SELECT job INTO emploi FROM emp WHERE ename = nom;
    IF emploi = 'CLERK' THEN GOTO remplissage;
    END IF;
    UPDATE emp SET comm = 1000 WHERE ename = nom;
    mes := nom||' commission modifiée';
    GOTO final;
<< remplissage>>
    mes := nom||' est avocat';
<<final>>
    dbms_output.put_line(mes);
COMMIT;
END;
/
```

Traitements répétitifs

- Définition
- Boucles
 - De base
 - FOR
 - WHILE
 - CURSOR .. FOR

Traitements répétitifs

- Boucle de base
 - Syntaxe

```
BEGIN
    ...
    LOOP [<<nom_boucle>>]
        Instructions;
        [EXIT [nom_boucle] [WHEN condition];]
    END LOOP [Nom_boucle];
    ...
END;
```


Traitements répétitifs

- Boucle de base
 - Exemple : Afficher le factoriel de 9

```

DECLARE
    compteur NUMBER(2) := 1;
    resultat NUMBER(10) := 1;
BEGIN
    LOOP
        resultat := resultat * compteur;
        compteur := compteur + 1;
        IF compteur = 10 THEN
            EXIT;
        END IF;
    END LOOP;
    dbms_output.put_line(resultat);
END;
/

```

```

DECLARE
    compteur NUMBER(2) := 1;
    resultat NUMBER(10) := 1;
BEGIN
    LOOP
        resultat := resultat * compteur;
        compteur := compteur + 1;
        EXIT WHEN compteur = 10;
    END LOOP;
    dbms_output.put_line(resultat);
END;
/

```

```

DECLARE
    compteur NUMBER(2) := 1;
    resultat NUMBER(10) := 1;
BEGIN
    LOOP <<factoriel>>
        resultat := resultat * compteur;
        compteur := compteur + 1;
        EXIT WHEN compteur = 10;
    END LOOP factoriel;
    dbms_output.put_line(resultat);
END;
/

```

Traitements répétitifs

- Boucle FOR
 - Syntaxe

```
BEGIN
    ...
    FOR compteur IN [REVERSE] exp1 .. exp2
    LOOP
        Instructions;
    END LOOP;
    ...
END;
```



Pas besoin de le déclarer avant

Traitements répétitifs

- Boucle FOR
 - Exemple : Afficher le factoriel de 9

```
DECLARE
    resultat NUMBER(10) := 1;
BEGIN
    FOR i IN 1..9
    LOOP
        resultat := resultat * i;
    END LOOP;
    dbms_output.put_line(resultat);
END;
/
```

```
DECLARE
    resultat NUMBER(10) := 1;
BEGIN
    FOR i IN REVERSE 1..9
    LOOP <<factoriel>>
        resultat := resultat * i;
        dbms_output.put_line(resultat);
    END LOOP factoriel;
    dbms_output.put_line(resultat);
END;
/
```

Traitements répétitifs

- Boucle WHILE
 - Syntaxe

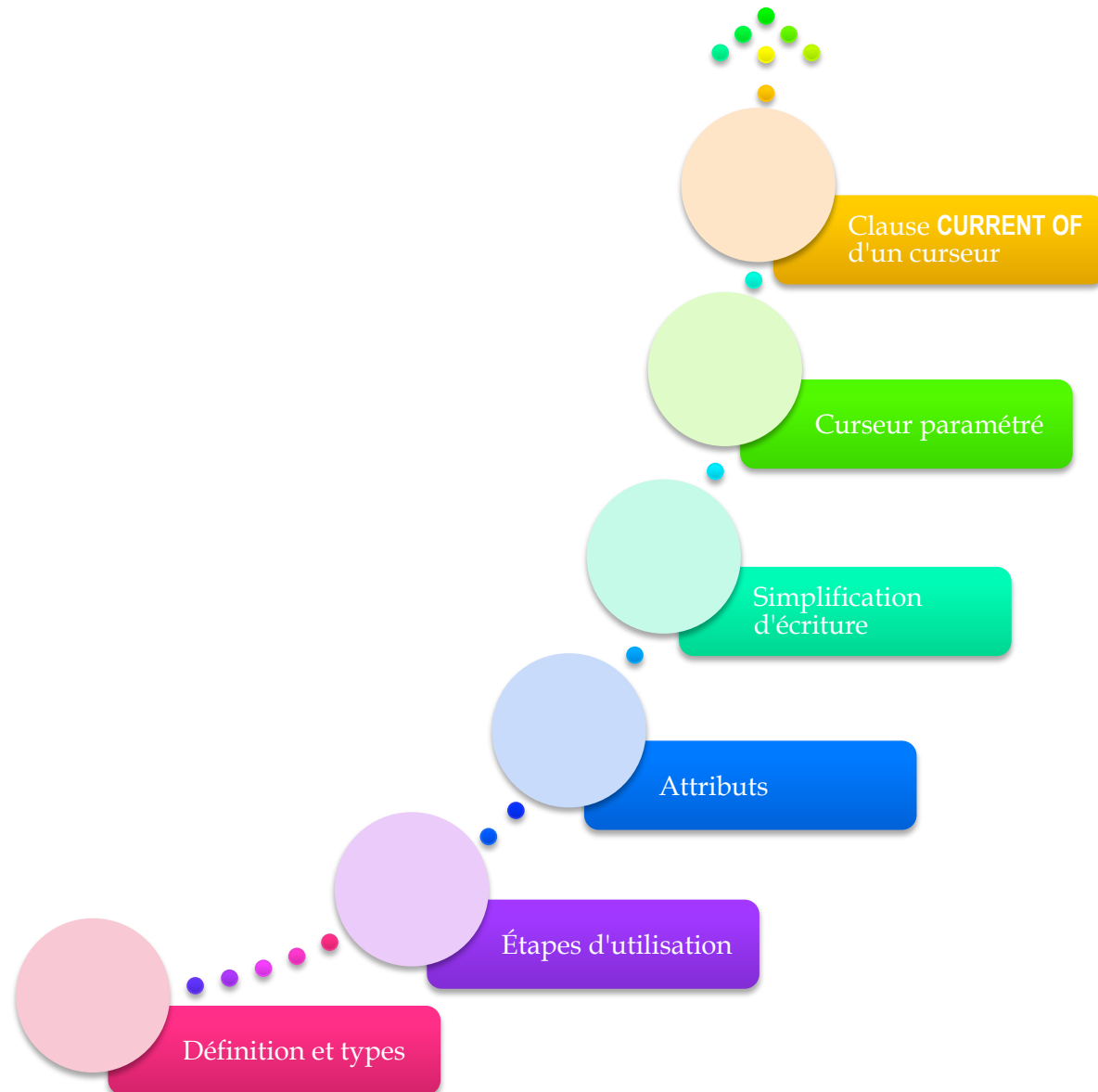
```
BEGIN
    ...
    WHILE condition
    LOOP
        Instructions;
    END LOOP;
    ...
END;
```

Traitements répétitifs

- Boucle WHILE
 - Exemple : Afficher le factoriel de 9

```
DECLARE
    resultat NUMBER(10) := 1;
    compteur NUMBER(2) := 1;
BEGIN
    WHILE compteur < 10
    LOOP
        resultat := resultat * compteur;
        compteur := compteur + 1;
    END LOOP ;
    dbms_output.put_line(resultat);
END;
/
```

Curseurs en PL/SQL



Curseurs en PL/SQL

- Définition
 - Zone de mémoire utilisée par Oracle pour analyser et interpréter les commandes SQL
 - Les statuts d'exécution de chaque commande SQL se trouvent dans le curseur
- Types
 - Curseur implicite
 - Généré par Oracle pour chaque commande SQL d'un bloc
 - **Curseur explicite**
 - Généré par l'utilisateur pour traiter une commande SELECT ramenant **plusieurs lignes**

- Étapes d'utilisation d'un curseur explicite
 - Déclaration du curseur
 - Ouverture du curseur
 - Traitement des lignes
 - Fermeture du curseur

Curseurs en PL/SQL

- Étapes d'utilisation d'un curseur explicite
 - Déclaration du curseur
 - Dans la section DECLARE avec
 - Son nom
 - Son ordre SELECT

Curseurs en PL/SQL

- Étapes d'utilisation d'un curseur explicite
 - Déclaration du curseur
 - Syntaxe

```
DECLARE
...
CURSOR nom_curseur IS ordre_SELECT;
```

- Exemple

```
DECLARE
CURSOR dept_21 IS
SELECT * FROM emp WHERE deptno = 21;
```

- Étapes d'utilisation d'un curseur explicite
 - Ouverture du curseur
 - Est faite dans la section BEGIN
 - Est indispensable pour
 - Faire exécuter l'ordre SELECT
 - Allouer de la mémoire
 - Analyser syntaxiquement et sémantiquement l'ordre SELECT
 - Assurer la cohérence en positionnant des verrous éventuels (Si SELECT ... FOR UPDATE)

Curseurs en PL/SQL

- Étapes d'utilisation d'un curseur explicite
 - Ouverture du curseur
 - Syntaxe

```
BEGIN
    ...
    OPEN nom_curseur;
```

- Exemple

```
DECLARE
    CURSOR dept_21 IS
        SELECT * FROM emp WHERE deptno = 21;
BEGIN
    OPEN dept_21;
    ...
```

- Étapes d'utilisation d'un curseur explicite
 - Traitement des lignes
 - Après l'exécution du curseur, on peut traiter le résultat dans une variable (ex: RecordSet)
 - Une seule ligne est ramenée à la fois

Curseurs en PL/SQL

- Étapes d'utilisation d'un curseur explicite
 - Traitement des lignes
 - Syntaxe

```
BEGIN
```

```
...
```

```
FETCH nom_curseur INTO liste_variables;
```

Curseurs en PL/SQL

- Étapes d'utilisation d'un curseur explicite
 - Traitement des lignes
 - Exemple

```
DECLARE
    CURSOR dept_21 IS SELECT ename, sal FROM emp WHERE deptno = 21;
    nom emp.ename%TYPE;
    salaire emp.sal%TYPE;
BEGIN
    OPEN dept_21;
    LOOP
        FETCH dept_21 INTO nom, salaire;
        IF Salaire > 200 THEN
            dbms_output.put_line(nom || ' a un salaire de ' || salaire);
        END IF;
        EXIT WHEN SALAIRE >=2500 ;
    END LOOP;
END;
/
```

- Étapes d'utilisation d'un curseur explicite
 - Fermeture du curseur
 - Après le traitement des lignes, on ferme le curseur pour libérer la place mémoire

Curseurs en PL/SQL

- Étapes d'utilisation d'un curseur explicite
 - Fermeture du curseur
 - Syntaxe

```
BEGIN  
...  
CLOSE nom_curseur;
```

Curseurs en PL/SQL

- Étapes d'utilisation d'un curseur explicite
 - Fermeture du curseur
 - Exemple

```
DECLARE
    CURSOR dept_21 IS SELECT ename, sal FROM emp WHERE deptno = 21;
    nom emp.ename%TYPE;
    salaire emp.sal%TYPE;
BEGIN
    OPEN dept_21;
    LOOP
        FETCH dept_21 INTO nom, salaire;
        IF Salaire > 200 THEN
            dbms_output.put_line(nom || ' a un salaire de ' || salaire);
        END IF;
        EXIT WHEN SALAIRE >=2500 ;
    END LOOP;
    CLOSE dept_21;
END;
/
```

- Exemple
 - Trouver les n plus gros salaires de la table emp

```
PROMPT Donner le nombre de salaires
ACCEPT nombre_salaire
DECLARE
    CURSOR gros_salaire IS SELECT sal, ename FROM emp order by sal DESC;
    nom emp.ename%TYPE;
    salaire emp.sal%TYPE;

BEGIN
    OPEN gros_salaire;
    FOR i IN 1 .. &nombre_salaire
    LOOP
        FETCH gros_salaire INTO salaire, nom;
        dbms_output.put_line(nom || ' a un salaire de ' || salaire);

    END LOOP;
    CLOSE gros_salaire;
END;
/
```

Attributs d'un curseur

- Définition
 - Donnent des indications sur son état
 - %FOUND et %NOTFOUND
 - Dernière ligne traitée
 - %ISOPEN
 - Ouverture d'un curseur
 - %ROWCOUNT
 - Nombre de lignes déjà traitées

Attributs d'un curseur

- %FOUND
 - Donne une valeur booléenne TRUE
 - Implicite: SQL%FOUND
 - Quand INSERT, UPDATE, DELETE traitent au moins une ligne
 - SELECT .. INTO ramène une et une seule ligne
 - Explicite: nom_curseur%FOUND
 - Le dernier FETCH ramène une ligne

Attributs d'un curseur

- %FOUND

```
DECLARE
    CURSOR dept_21 IS SELECT ename, sal FROM emp WHERE deptno = 21;
    nom emp.ename%TYPE;
    salaire emp.sal%TYPE;
BEGIN
    OPEN dept_21;
    FETCH dept_21 INTO nom, salaire;
    WHILE dept_21%FOUND
    LOOP
        IF Salaire > 200 THEN
            dbms_output.put_line(nom || ' a un salaire de ' || salaire);
        END IF;
        FETCH dept_21 INTO nom, salaire;
    END LOOP;
    CLOSE dept_21;
END;
/
```

Attributs d'un curseur

- %NOTFOUND
 - Donne une valeur booléenne TRUE
 - Implicite: SQL%NOTFOUND
 - Quand INSERT, UPDATE, DELETE ne traitent aucune ligne
 - SELECT .. INTO ne ramène pas de ligne
 - Explicite: nom_curseur%NOTFOUND
 - Le dernier FETCH ne ramène pas de ligne

Attributs d'un curseur

- %NOTFOUND

```
DECLARE
    CURSOR dept_21 IS SELECT ename, sal FROM emp WHERE deptno = 21;
    nom emp.ename%TYPE;
    salaire emp.sal%TYPE;
BEGIN
    OPEN dept_21;
    LOOP
        FETCH dept_21 INTO nom, salaire;
        EXIT WHEN dept_21%NOTFOUND;
        IF Salaire > 200 THEN
            dbms_output.put_line(nom || ' a un salaire de ' || salaire);
        END IF;
    END LOOP;
    CLOSE dept_21;
END;
/
```


Attributs d'un curseur

- %ISOPEN
 - Donne une valeur booléenne
 - Implicite: SQL%ISOPEN
 - Toujours FALSE car ORACLE referme les curseurs après utilisation
 - Explicite: nom_curseur%ISOPEN
 - TRUE quand le curseur est ouvert

Attributs d'un curseur

- %ISOPEN

```
DECLARE
    CURSOR dept_21 IS SELECT ename, sal FROM emp WHERE deptno = 21;
    nom emp.ename%TYPE;
    salaire emp.sal%TYPE;
BEGIN
    IF NOT (dept_21%ISOPEN) THEN OPEN dept_21;
    END IF;
    LOOP
        FETCH dept_21 INTO nom, salaire;
        EXIT WHEN dept_21%NOTFOUND;
        IF Salaire > 200 THEN
            dbms_output.put_line(nom || ' a un salaire de ' || salaire);
        END IF;
    END LOOP;
    CLOSE dept_21;
END;
/
```

Attributs d'un curseur

- %ROWCOUNT
 - Donne une valeur numérique
 - Implicite: SQL% ROWCOUNT
 - Le nombre de lignes traitées avec INSERT, UPDATE et DELETE
 - Avec SELECT .. INTO
 - 0 quand aucune ligne n'est renvoyée
 - 1 quand 1 ligne est renvoyée
 - 2 quand plusieurs lignes sont renvoyées
 - Explicite: nom_curseur%ROWCOUNT
 - Indique le nombre de lignes envoyé par FETCH

Attributs d'un curseur

- % ROWCOUNT

```
DECLARE
    CURSOR dept_21 IS SELECT ename, sal FROM emp WHERE deptno = 21;
    nom emp.ename%TYPE;
    salaire emp.sal%TYPE;
BEGIN
    IF NOT (dept_21%ISOPEN) THEN OPEN dept_21;
    END IF;
    LOOP
        FETCH dept_21 INTO nom, salaire;
        EXIT WHEN dept_21%NOTFOUND or dept_21%ROWCOUNT >15;
        IF Salaire > 200 THEN
            dbms_output.put_line(nom || ' a un salaire de ' || salaire);
        END IF;
    END LOOP;
    CLOSE dept_21;
END;
/
```

Simplification d'écriture

- On peut utiliser une déclaration de variables plus simple
- Syntaxe

```
DECLARE
```

```
...
```

```
CURSOR nom_curseur IS ordre_SELECT;  
var_enregistrement nom_curseur%ROWTYPE;
```

```
BEGIN
```

```
...
```

```
FETCH nom_curseur INTO var_enregistrement;  
var_enregistrement.attribut := ...
```

Simplification d'écriture (ici)

- Exemple

```
DECLARE
  CURSOR dept_21 IS
    SELECT ename, sal+NVL(comm, 0) salaire FROM emp WHERE deptno=21;
  dept_21_enreg dept_21%ROWTYPE;
BEGIN
  OPEN dept_21;
  LOOP
    FETCH dept_21 INTO dept_21_enreg;
    EXIT WHEN dept_21%NOTFOUND;
    IF dept_21_enreg.salaire > 200 THEN
      dbms_output.put_line(dept_21_enreg.ename || ' a un salaire
        de ' || dept_21_enreg.salaire);
    END IF;
  END LOOP;
  CLOSE dept_21;
END;
/
```

Toute *expression* dans le
SELECT doit être renommée

Simplification d'écriture

- Pour faciliter la programmation des curseurs dans les boucles

```
DECLARE
```

```
...
```

```
CURSOR nom_c curseur IS ordre_SELECT;
```

```
var_c
```

```
BEGIN
```

```
OPEN
```

```
LOOP
```

```
FETCH
```

```
IF
```

```
/* ----
```

```
END
```

```
CLOSE
```

```
DECLARE
```

```
...
```

```
CURSOR nom_c curseur IS ordre_SELECT;
```

```
BEGIN
```

```
FOR var_enregistrement IN nom_c LOOP
```

```
/* ----- */
```

```
END LOOP;
```

Simplification d'écriture

- Exemple

```
DECLARE
    CURSOR dept_21 IS
        SELECT ename, sal+NVL(comm, 0) salaire FROM emp where deptno=21;

BEGIN
    FOR dept_21_enreg IN dept_21
    LOOP
        IF dept_21_enreg.salaire > 2000 THEN
            dbms_output.put_line(dept_21_enreg.ename || ' a un salaire de ' ||
                                dept_21_enreg.salaire);
        END IF;
    END LOOP;

END;
/
```


Curseur paramétré

- Permet de réutiliser un même curseur dans le même bloc avec des valeurs différentes
- Syntaxe

CHAR, NUMBER, DATE, BOOLEAN (sans taille)

```
DECLARE
```

```
...
```

```
CURSOR nom_curseur(par1 TYPE par2 TYPE ...) IS ordre SELECT;
```

```
DECLARE
```

```
BEGIN
```

```
...
```

```
CURSOR nom_curseur(par1 TYPE, par2 TYPE, ...) IS ordre_SELECT;
```

```
...
```

```
BEGIN
```

```
...
```

```
FOR nom_enregistrement IN nom_curseur (val1, val2, ...);
```

```
...
```

Curseur paramétré

- Trouver les n meilleurs salaires au département 21

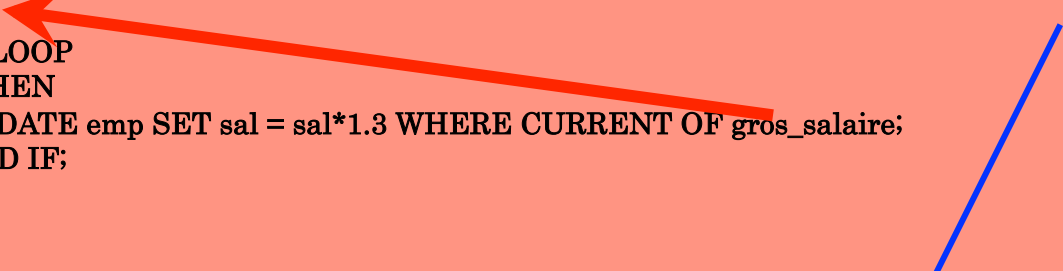
```
PROMPT Donner le nombre de salaires  
ACCEPT nombre_salaire  
PROMPT Donner le numéro du département
```

```
AC  
DE DECLARE  
        CURSOR gros_salaire IS SELECT sal, ename FROM emp WHERE deptno=&num_dept order by  
        sal DESC;  
        CURSOR c1_salaire(psal NUMBER) IS SELECT ename, sal FROM emp WHERE sal = psal;  
  
BE BEGIN  
        FOR I IN gros_salaire LOOP  
            EXIT WHEN gros_salaire%ROWCOUNT > &nombre_salaire;  
  
            FOR JJ IN c1_salaire(I.sal) LOOP  
                dbms_output.put_line(JJ.ename || ' a un salaire de ' || JJ.sal);  
            END LOOP;  
        END LOOP;  
  
EN  
/ END;  
/
```

Clause CURRENT OF

- Augmenter les employés dont le salaire est supérieur à 1500

```
DECLARE
  CURSOR gros_salaire IS SELECT sal, ename FROM emp WHERE deptno = 21 FOR UPDATE OF sal;
BEGIN
  FOR I IN gros_salaire LOOP
    IF I.sal > 1500 THEN
      UPDATE emp SET sal = sal*1.3 WHERE CURRENT OF gros_salaire;
    END IF;
  END LOOP;
END;
/
```



Seulement la colonne sal du département 21 sera verrouillée

Gestion des erreurs

- Section EXCEPTION
- Anomalie programme utilisateur
- Erreur Oracle

- Section EXCEPTION
 - Permet d'affecter un traitement approprié aux erreurs survenues lors de l'exécution du bloc PL/SQL
 - 2 types d'erreurs
 - Interne (SQLCODE !=0)
 - Quand un bloc PL/SQL viole une règle d'Oracle ou dépasse une limite dépendant du système d'exploitation
 - Erreur programme utilisateur

- Section EXCEPTION
 - Solution
 - Définir et donner un nom à chaque erreur (différent pour erreur utilisateur et Oracle)
 - Utiliser des variables de type EXCEPTION dans la partie DECLARE
 - Définir le(s) traitement(s) correspondant à effectuer dans la partie EXCEPTION

Gestion des erreurs

- Anomalie programme utilisateur
 - Syntaxe

```
DECLARE
    ...
    nom_erreur EXCEPTION;
    ...
BEGIN
    ...
    IF (anomalie) THEN RAISE nom_erreur;
    ...
EXCEPTION
    WHEN nom_erreur THEN (traitement); -- Sortie du bloc après exécution du traitement
    ...
END;
/
```

Gestion des erreurs

- Anomalie programme utilisateur
 - Exemple

Provient d'un autre programme

```
DECLARE
    pas_comm EXCEPTION;
    Salaire emp.sal%TYPE;
    commission emp.comm%TYPE;
BEGIN
    SELECT sal, comm INTO Salaire, commission FROM emp WHERE empno = :num_emp;
    IF (commission = 0 OR commission IS NULL) THEN RAISE pas_comm;
    END IF;
EXCEPTION
    WHEN pas_comm THEN
        dbms_output.put_line('Pas de commission');
END;
/
```


Gestion des erreurs

- Erreur Oracle
 - Syntaxe

```
DECLARE
    ...
    nom_erreur EXCEPTION;
    PRAGMA EXCEPTION_INIT(nom, code_erreur);
    ...
BEGIN
    ...
    ⚡ Dès que l'erreur Oracle est rencontrée, passage automatique à la section EXCEPTION
    ...
EXCEPTION
    WHEN nom_erreur THEN (traitement1) ;
    WHEN TOO_MANY_ROWS OR NO_DATA_FOUND THEN (traitement2) ;
    WHEN OTHERS THEN (traitement3) ;-- Sortie du bloc après exécution du
traitement
    ...
END;
```

/

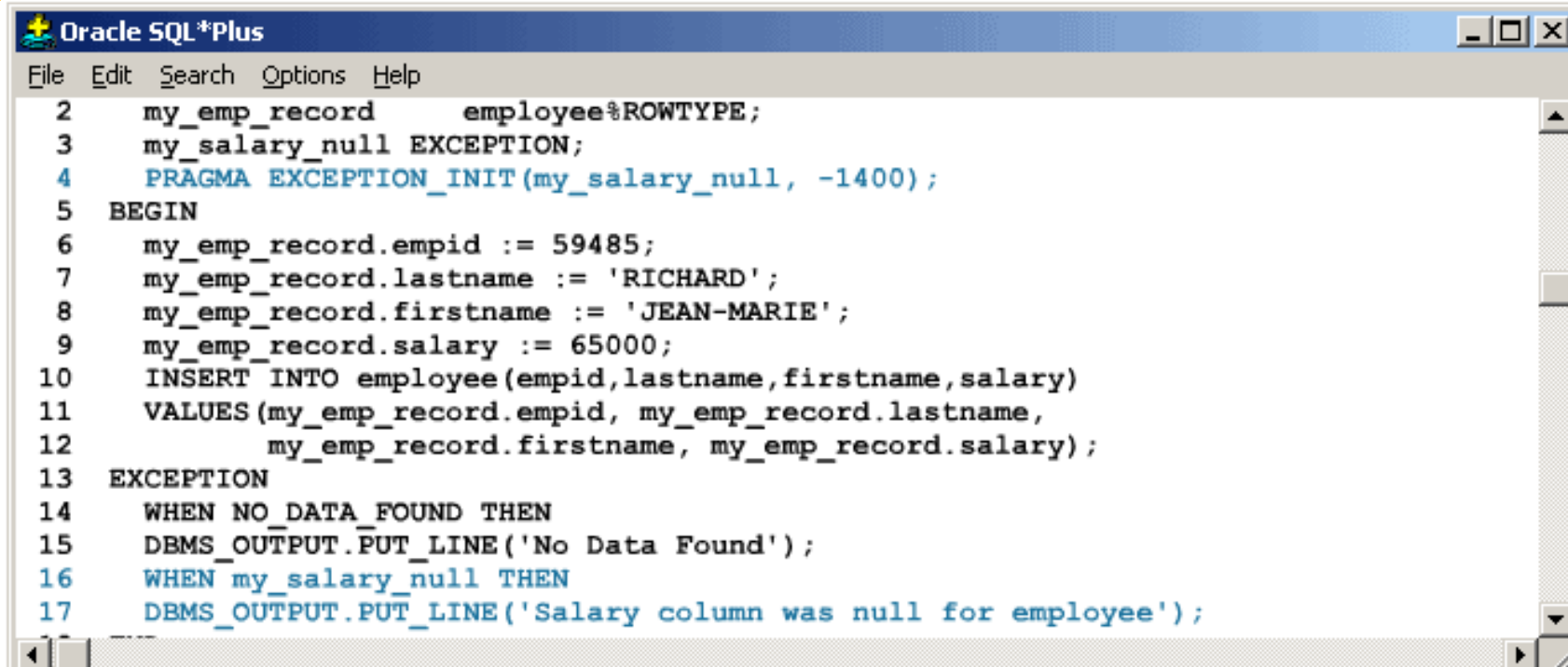
- Erreurs Oracle prédéfinies

DUP_VAL_ON_INDEX	-1
INVALID_CURSOR	-1001
INVALID_NUMBER	-1722
LOGIN_DENIED	-1017
NO_DATA_FOUND	+100
NO_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 non explicitement nommées

On peut utiliser le gestionnaire OTHERS ou EXCEPTION_INIT pour nommer ces erreurs

Gestion des erreurs

- Erreur Oracle
 - Exemple



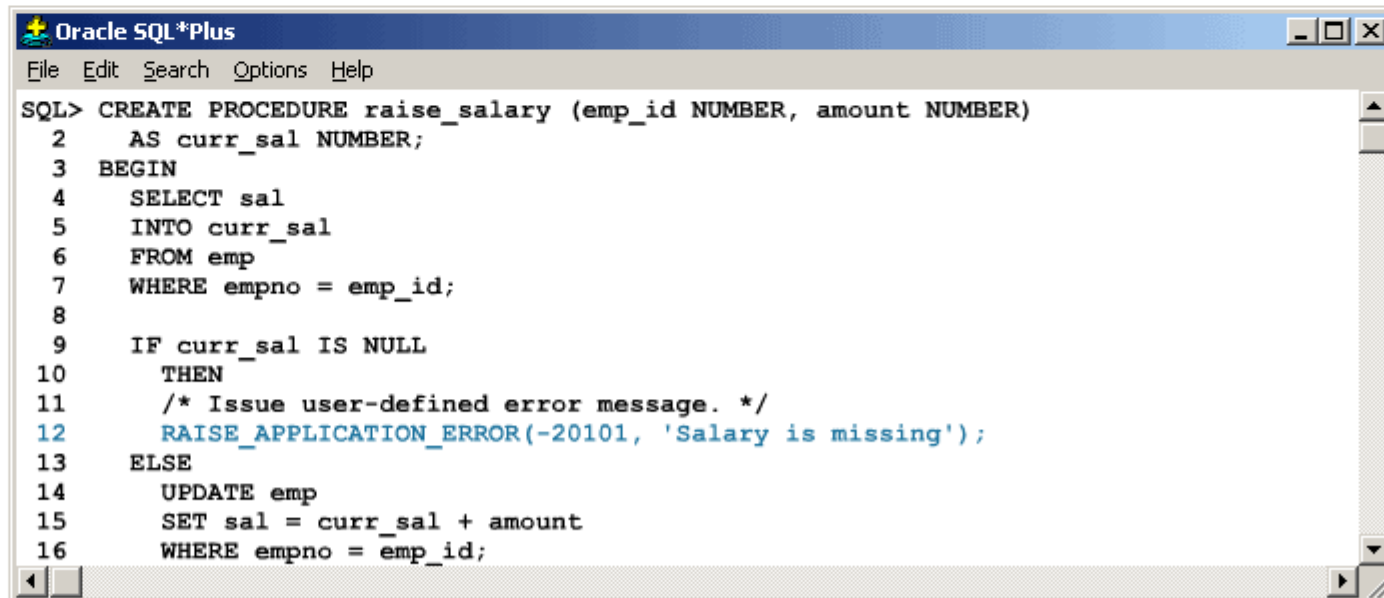
```
Oracle SQL*Plus
File Edit Search Options Help
2  my_emp_record      employee%ROWTYPE;
3  my_salary_null EXCEPTION;
4  PRAGMA EXCEPTION_INIT(my_salary_null, -1400);
5  BEGIN
6  my_emp_record.empid := 59485;
7  my_emp_record.lastname := 'RICHARD';
8  my_emp_record.firstname := 'JEAN-MARIE';
9  my_emp_record.salary := 65000;
10 INSERT INTO employee(empid,lastname,firstname,salary)
11 VALUES(my_emp_record.empid, my_emp_record.lastname,
12         my_emp_record.firstname, my_emp_record.salary);
13 EXCEPTION
14 WHEN NO_DATA_FOUND THEN
15 DBMS_OUTPUT.PUT_LINE('No Data Found');
16 WHEN my_salary_null THEN
17 DBMS_OUTPUT.PUT_LINE('Salary column was null for employee');
```

- Erreur Oracle
 - Exemple (détection de doublons sur une clé primaire)

```
DECLARE
    ...
BEGIN
    ...
    INSERT INTO dept VALUES (numéro, nom, ville);
    ...
EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
        dbms_output.put_line (numéro || 'déjà inséré') ;
    ...
END;
/
```

Gestion des erreurs

- Pour personnaliser les messages d'erreurs
 - `RAISE_APPLICATION_ERROR(code_erreur, message)`



```
Oracle SQL*Plus
File Edit Search Options Help

SQL> CREATE PROCEDURE raise_salary (emp_id NUMBER, amount NUMBER)
2   AS curr_sal NUMBER;
3   BEGIN
4     SELECT sal
5     INTO curr_sal
6     FROM emp
7     WHERE empno = emp_id;
8
9     IF curr_sal IS NULL
10    THEN
11      /* Issue user-defined error message. */
12      RAISE_APPLICATION_ERROR(-20101, 'Salary is missing');
13    ELSE
14      UPDATE emp
15      SET sal = curr_sal + amount
16      WHERE empno = emp_id;
```

Fonctions propres à PL/SQL

- SQLCODE
 - Renvoie le code numérique de l'erreur courante
- SQLERRM [(Code_erreur)]
 - Renvoie le libellé de l'erreur courante ou le libellé correspondant au code spécifié comme paramètre

That's it !!!