5

Utiliser des types de données composites

ORACLE

Objectifs

- créer des enregistrements PL/SQL définis par l'utilisateur
- créer des enregistrements avec l'attribut `%ROWTYPE`
- créer une table `INDEX BY`
- créer une table d'enregistrements `INDEX BY`
- faire la distinction entre les enregistrements, les tables, et les tables d'enregistrement

ORACLE

Types de données composites

- On va voir deux catégories de données composites PL/SQL :
  - enregistrements
  - table `INDEX BY`
- Les types de données composites contiennent des composantes internes
- Les types de données composites sont réutilisables

ORACLE

Enregistrements PL/SQL

- Ils doivent être composés d'un ou plusieurs champs de type scalaire, `RECORD` ou `INDEX BY`
- Leur structure est similaire à celle des enregistrements en langage de programmation
- Ils traitent un ensemble de champs comme une unité logique
- Ils permettent d'extraire une ligne de données à partir d'une table pour la traiter

ORACLE

Créer un enregistrement PL/SQL

Syntaxe :

```
TYPE type_name IS RECORD
      (field_declaration[, field_declaration]…);
identifier    type_name;
```

Où *field_declaration* est :

```
field_name {field_type | variable%TYPE
            | table.column%TYPE | table%ROWTYPE}
            [[NOT NULL] {:= | DEFAULT} expr]
```

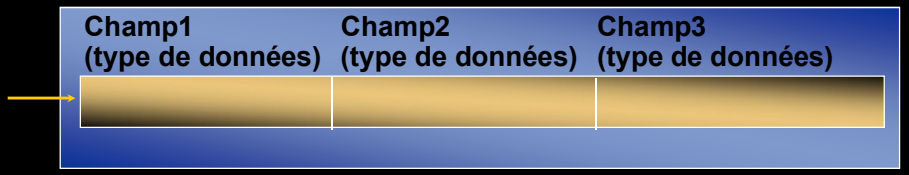

Créer un enregistrement PL/SQL

Déclarer des variables pour le stockage du nom, du poste et du salaire d'un nouvel employé.

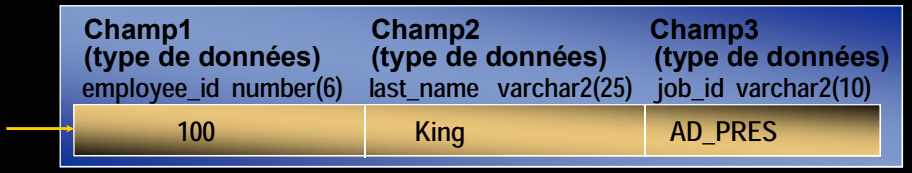
Exemple :

```
...
  TYPE emp_record_type IS RECORD
    (last_name    VARCHAR2(25),
     job_id       VARCHAR2(10),
     salary       NUMBER(8,2));
  emp_record      emp_record_type;
...
```


Structure d'un enregistrement PL/SQL



Example:



Attribut %ROWTYPE

- Permet de déclarer une variable à partir d'un ensemble de colonnes d'une table ou d'une vue de base de données
- Doit être précédé du nom de la table de base de données
- Les champs de l'enregistrement tirent leurs noms et leurs types de données des colonnes de la table ou de la vue

Avantages liés à l'utilisation de l'attribut %ROWTYPE

- Il n'est pas nécessaire de connaître le nombre et les types de données des colonnes de la table de base de données sous-jacente
- Le nombre et les types des colonnes de la table de base de données sous-jacente peuvent être modifiés à l'exécution
- L'attribut permet d'extraire une ligne avec l'instruction SELECT *

Attribut %ROWTYPE

Exemples :

Déclarer une variable pour le stockage des informations relatives à un service de la table DEPARTMENTS

```
dept_record    departments%ROWTYPE;
```

Déclarer une variable pour le stockage des informations relatives à un employé de la table EMPLOYEES

```
emp_record     employees%ROWTYPE;
```


Tables INDEX BY

- Ces tables comportent 2 composants :
  - une clé primaire de type BINARY_INTEGER
  - une colonne de type scalaire ou RECORD
- Leur taille peut augmenter de façon dynamique

Créer une table INDEX BY

Syntaxe :

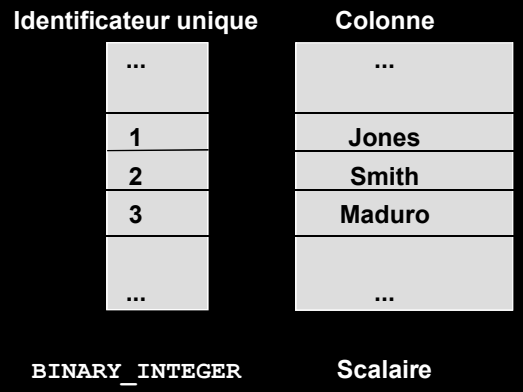
```
TYPE type_name IS TABLE OF
     {column_type | variable%TYPE
     | table.column%TYPE} [NOT NULL]
     | table.%ROWTYPE
     [INDEX BY BINARY_INTEGER];
identifier    type_name;
```

Déclarer une variable INDEX BY pour le stockage des noms

Exemple :

```
...
TYPE ename_table_type IS TABLE OF
                              employees.last_name%TYPE
  INDEX BY BINARY_INTEGER;
ename_table ename_table_type;
...
```

Structure des tables INDEX BY



Créer une table INDEX BY

```
DECLARE
  TYPE ename_table_type IS TABLE OF
       employees.last_name%TYPE
       INDEX BY BINARY_INTEGER;
  TYPE hiredate_table_type IS TABLE OF DATE
       INDEX BY BINARY_INTEGER;
  ename_table       ename_table_type;
  hiredate_table    hiredate_table_type;
BEGIN
  ename_table(1)    := 'CAMERON';
  hiredate_table(8) := SYSDATE + 7;
    IF ename_table.EXISTS(1) THEN
      INSERT INTO ...
    ...
END;
/
```


Utiliser les méthodes des tables INDEX BY

Les méthodes suivantes facilitent l'utilisation des tables INDEX BY :

- EXISTS
- COUNT
- FIRST and LAST
- DELETE

Table d'enregistrements INDEX BY

- Définir une variable TABLE avec un type de données PL/SQL admis
- Déclarer une variable PL/SQL pour le stockage des informations d'un service.

Exemple :

```
DECLARE
  TYPE dept_table_type IS TABLE OF
       departments%ROWTYPE
       INDEX BY BINARY_INTEGER;
  dept_table dept_table_type;
  -- Each element of dept_table is a record
```

Exemple de table d'enregistrements INDEX BY

```
SET SERVEROUTPUT ON
DECLARE
   TYPE emp_table_type is table of
      employees%ROWTYPE INDEX BY BINARY_INTEGER;
   my_emp_table  emp_table_type;
   v_count       NUMBER(3):= 104;
BEGIN
  FOR i IN 100..v_count
  LOOP
        SELECT * INTO my_emp_table(i) FROM employees
        WHERE employee_id = i;
  END LOOP;
  FOR i IN my_emp_table.FIRST..my_emp_table.LAST
  LOOP
     DBMS_OUTPUT.PUT_LINE(my_emp_table(i).last_name);
  END LOOP;
END;
```


Synthèse

Apprendre à :

- définir et référencer des variables PL/SQL possédant des types de données composites :
  - enregistrements PL/SQL
  - tables INDEX BY
  - tables d'enregistrements INDEX BY
- définir un enregistrement PL/SQL en utilisant l'attribut %ROWTYPE

Présentation de l'exercice 5

Dans cet exercice, vous allez :

- déclarer des tables INDEX BY
- traiter des données en utilisant les tables INDEX BY
- déclarer un enregistrement PL/SQL
- traiter des données à l'aide d'un enregistrement PL/SQL

6 Ecrire des curseurs explicites

Objectifs

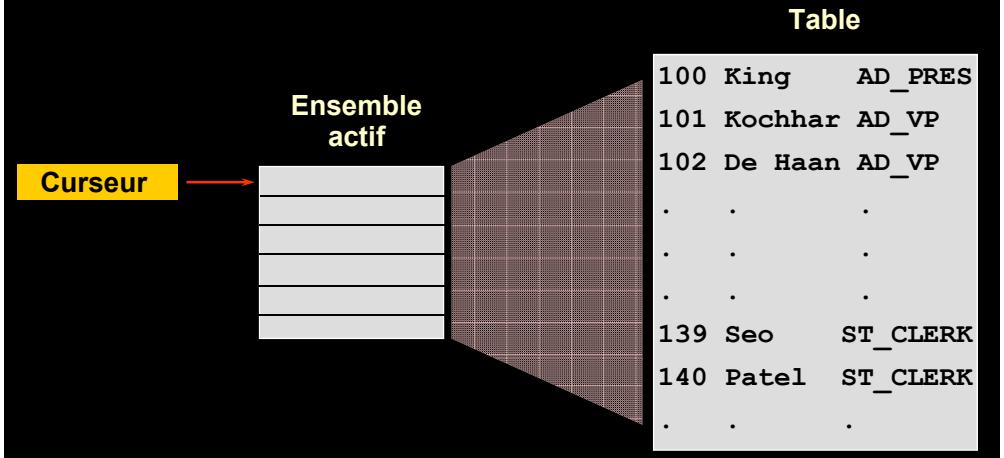
- faire la différence entre un curseur implicite et un curseur explicite
- savoir quand et pourquoi utiliser un curseur explicite
- utiliser une variable de type RECORD en PL/SQL
- écrire une boucle `FOR` de curseur

A propos des curseurs

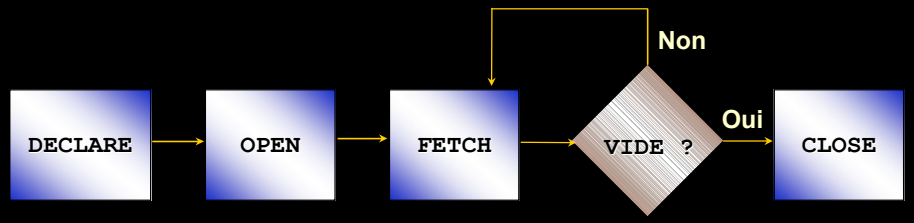
Chaque instruction SQL exécutée par le serveur Oracle a son propre curseur individuel qui lui est associé :

- curseurs implicites : déclarés pour toutes les instructions `LMD`
- curseurs explicites : déclarés et nommés par le programmeur pour les requêtes select multilignes

Fonctions des curseurs explicites



Contrôler les curseurs explicites



- Créer une zone SQL nommée
- Identifier l'ensemble actif
- Charger la ligne en cours dans des variables
- Tester l'existence de lignes
- Si des lignes existent, revenir à `FETCH`
- Libérer l'ensemble actif

Contrôler les curseurs explicites

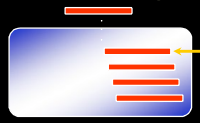
1. Ouvrir le curseur
2. Extraire une ligne
3. Fermer le curseur

1. Ouvrir le curseur.

Contrôler les curseurs explicites

1. Ouvrir le curseur
2. Extraire une ligne
3. Fermer le curseur

2. Extraire une ligne à l'aide du curseur.

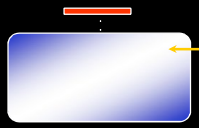


Continuer jusqu'à ce que
le curseur soit vide

Contrôler les curseurs explicites

1. Ouvrir le curseur
2. Extraire une ligne
3. Fermer le curseur

3. Fermer le curseur.



Déclarer le curseur

Syntaxe :

```
CURSOR cursor_name IS
     select_statement;
```

- Ne pas inclure la clause `INTO` dans la déclaration du curseur
- S'il existe un ordre précis de traitement des lignes, utiliser la clause `ORDER BY` dans l'interrogation

Déclarer le curseur

Exemple :

```
DECLARE
  CURSOR emp_cursor IS
    SELECT employee_id, last_name
    FROM   employees;

  CURSOR dept_cursor IS
    SELECT *
    FROM   departments
    WHERE  location_id = 170;
BEGIN
  ...
```


Ouvrir le curseur

Syntaxe :

```
OPEN  cursor_name;
```

- Ouvrir le curseur pour exécuter l'interrogation et identifier l'ensemble actif
- Si l'interrogation ne renvoie pas de ligne, aucune exception n'est déclenchée
- Utiliser les attributs de curseur pour tester le résultat après une extraction

Extraire les données à partir du curseur

Syntaxe :

```
FETCH cursor_name INTO  [variable1, variable2, ...]
                        | record_name];
```

- Extraire les valeurs de la ligne en cours pour les placer dans des variables
- Inclure le même nombre de variables
- Ajuster la position des variables par rapport aux colonnes
- Vérifier si le curseur contient des lignes.

Extraire les données à partir du curseur

Exemple :

```
LOOP
  FETCH emp_cursor INTO v_empno,v_ename;
  EXIT WHEN ...;
  ...
    -- Process the retrieved data
  …
END LOOP;
```

Fermer le curseur

Syntaxe :

```
CLOSE     cursor_name;
```

- Fermer le curseur après avoir terminé le traitement des lignes
- Rouvrir le curseur, si nécessaire
- Ne pas essayer d'extraire les données d'un curseur s'il a été fermé.

Attributs d'un curseur explicite

Obtenir les informations d'état concernant un curseur

Attribut	Type	Description
%ISOPEN	Booléen	Prend la valeur TRUE si le curseur est ouvert
%NOTFOUND	Booléen	Prend la valeur TRUE si la dernière extraction ne renvoie pas de ligne
%FOUND	Booléen	Prend la valeur TRUE si la dernière extraction renvoie une ligne ; complément de %NOTFOUND
%ROWCOUNT	Nombre	Prend la valeur correspondant au nombre total de lignes renvoyées jusqu'à présent

Attribut %ISOPEN

- Extraire les lignes uniquement lorsque le curseur est ouvert
- Utiliser l'attribut de curseur %ISOPEN avant de réaliser une extraction pour vérifier si le curseur est ouvert

Exemple :

```
IF NOT emp_cursor%ISOPEN THEN
   OPEN emp_cursor;
END IF;
LOOP
  FETCH emp_cursor...
```


Contrôler plusieurs extractions

- Traiter plusieurs lignes à partir d'un curseur explicite en utilisant une boucle
- Extraire une ligne à chaque itération
- Utiliser les attributs de curseur explicite pour contrôler le succès de chaque extraction.

Attributs %NOTFOUND et %ROWCOUNT

- Utiliser l'attribut de curseur %ROWCOUNT pour extraire un nombre exact de lignes
- Utiliser l'attribut de curseur %NOTFOUND pour déterminer dans quels cas la sortie de la boucle doit s'effectuer.

Exemple

```
DECLARE
      v_empno  employees.employee_id%TYPE;
      v_ename  employees.last_name%TYPE;
      CURSOR emp_cursor IS
        SELECT employee_id, last_name
        FROM   employees;
    BEGIN
      OPEN emp_cursor;
      LOOP
        FETCH emp_cursor INTO v_empno, v_ename;
        EXIT WHEN emp_cursor%ROWCOUNT > 10 OR
                          emp_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE (TO_CHAR(v_empno)
                              ||'  '|| v_ename);
      END LOOP;
      CLOSE emp_cursor;
END ;
```


Curseurs et enregistrements

Traiter les lignes de l'ensemble actif en extrayant les valeurs pour les placer dans un enregistrement PL/SQL.

```
DECLARE
  CURSOR emp_cursor IS
    SELECT  employee_id, last_name
    FROM    employees;
  emp_record   emp_cursor%ROWTYPE;
BEGIN
  OPEN emp_cursor;
  LOOP
    FETCH emp_cursor INTO emp_record;
  ...
```

```
emp_record
employee_id          last_name
100                  King
```


Boucles FOR de curseur

Syntaxe :

```
FOR record_name IN cursor_name LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

- La boucle FOR de curseur simplifie le traitement des curseurs explicites
- Des opérations d'ouverture, d'extraction, de sortie et de fermeture ont lieu de manière implicite
- L'enregistrement est déclaré implicitement.

Boucles FOR de curseur

Afficher une liste des employés travaillant dans le service des ventes.

```
DECLARE
  CURSOR emp_cursor IS
    SELECT last_name, department_id
    FROM   employees;
BEGIN
  FOR emp_record IN emp_cursor LOOP
         -- implicit open and implicit fetch occur
    IF emp_record.department_id = 80 THEN
      ...
  END LOOP; -- implicit close occurs
END;
/
```


Boucles FOR de curseur utilisant des sous-interrogations

Il n'est pas nécessaire de déclarer le curseur.

Exemple :

```
BEGIN
  FOR emp_record IN (SELECT last_name, department_id
                     FROM   employees) LOOP
         -- implicit open and implicit fetch occur
    IF emp_record.department_id = 80 THEN
      ...
  END LOOP; -- implicit close occurs
END;
```


Synthèse

Apprendre à :

- différencier les types de curseur :
  - curseurs implicites : utilisés pour toutes les instructions LMD et les interrogations monolignes
  - curseurs explicites : utilisés pour les interrogations portant sur zéro, une ou plusieurs lignes
- manipuler les curseurs explicites
- évaluer l'état du curseur en utilisant des attributs de curseur
- utiliser les boucles FOR de curseur

Présentation de l'exercice 6

Dans cet exercice, vous allez :

- déclarer et utiliser des curseurs explicites pour interroger les lignes d'une table
- utiliser une boucle FOR de curseur
- appliquer des attributs de curseur pour contrôler l'état du curseur

7 Concepts avancés sur les curseurs explicites

ORACLE

Objectifs

- écrire un curseur utilisant des paramètres
- déterminer dans quels cas une clause `FOR UPDATE` doit être utilisée dans un curseur
- déterminer dans quels cas la clause `WHERE CURRENT OF` doit être utilisée
- écrire un curseur utilisant une sous-interrogation

ORACLE

Curseurs paramétrés

Syntaxe :

```
CURSOR cursor_name
  [(parameter_name datatype, ...)]
IS
  select_statement;
```

- Transmettre des paramètres au curseur au moment de son ouverture et de l'exécution de l'interrogation
- Ouvrir un curseur explicite à plusieurs reprises, en renvoyant un ensemble actif différent à chaque fois

```
OPEN  cursor_name(parameter_value,.....) ;
```

ORACLE

Curseurs paramétrés

Transmettre le numéro du service et l'intitulé du poste à la clause `WHERE`, dans l'instruction `SELECT` du curseur

```
DECLARE
  CURSOR emp_cursor
  (p_deptno NUMBER, p_job VARCHAR2) IS
     SELECT employee_id, last_name
     FROM   employees
     WHERE  department_id = p_deptno
     AND    job_id = p_job;
BEGIN
  OPEN emp_cursor (80, 'SA_REP');
  . . .
  CLOSE emp_cursor;
  OPEN emp_cursor (60, 'IT_PROG');
  . . .
END;
```

ORACLE

Clause FOR UPDATE

Syntaxe :

```
SELECT ...
FROM          ...
FOR UPDATE [OF column_reference][NOWAIT];
```

- Utiliser un verrouillage explicite pour interdire l'accès pendant la durée d'une transaction
- Verrouiller les lignes *avant* la mise à jour ou la suppression.

Clause FOR UPDATE

Extraire les employés qui travaillent dans le service 80 et mettre à jour leur salaire.

```
DECLARE
  CURSOR emp_cursor IS
    SELECT employee_id, last_name, department_name
    FROM   employees,departments
    WHERE  employees.department_id =
           departments.department_id
    AND employees.department_id = 80
    FOR UPDATE OF salary NOWAIT;
```


Clause WHERE CURRENT OF

Syntaxe :

```
WHERE CURRENT OF cursor ;
```

- Utiliser les curseurs pour mettre à jour ou supprimer la ligne en cours
- Inclure la clause FOR UPDATE dans l'interrogation du curseur pour verrouiller au préalable les lignes
- Utiliser la clause WHERE CURRENT OF pour référencer la ligne en cours à partir d'un curseur explicite

Clause WHERE CURRENT OF

```
DECLARE
CURSOR sal_cursor IS
 SELECT e.department_id, employee_id, last_name, salary
 FROM   employees e, departments d
 WHERE  d.department_id = e.department_id
  and   d.department_id = 60
 FOR UPDATE OF salary NOWAIT;
 BEGIN
 FOR emp_record IN sal_cursor
LOOP
  IF emp_record.salary < 5000 THEN
       UPDATE employees
       SET    salary = emp_record.salary * 1.10
       WHERE CURRENT OF sal_cursor;
  END IF;
END LOOP;
END;
/
```

Curseurs contenant des sous-interrogations

Exemple :

```
DECLARE
  CURSOR my_cursor IS
    SELECT t1.department_id, t1.department_name,
           t2.staff
    FROM   departments t1, (SELECT department_id,
                                   COUNT(*) AS STAFF
                            FROM employees
                            GROUP BY department_id) t2
    WHERE t1.department_id = t2.department_id
    AND   t2.staff >= 3;
...
```


Synthèse

Apprendre à :

- renvoyer des ensembles actifs différents à l'aide de curseurs paramétrés
- définir des curseurs contenant des sous-interrogations
- manipuler des curseurs explicites à l'aide de commandes contenant les clauses :
  - `FOR UPDATE`
  - `WHERE CURRENT OF`

Présentation de l'exercice 7

Dans cet exercice, vous allez :

- déclarer et utiliser des curseurs explicites paramétrés
- utiliser un curseur avec la clause `FOR UPDATE`

8 Traiter les exceptions

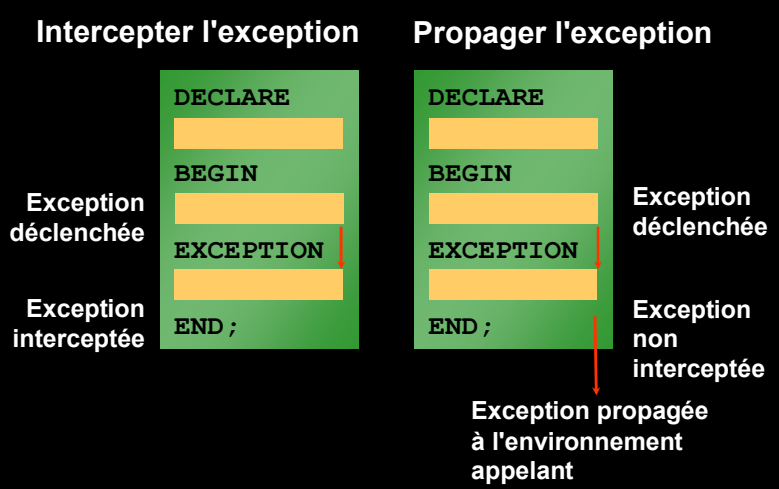
Objectifs

- définir des exceptions PL/SQL
- reconnaître les exceptions non traitées
- lister et utiliser les différents types de traitement des exceptions PL/SQL
- intercepter les erreurs non prédéfinies
- décrire l'effet d'une propagation des exceptions dans des blocs imbriqués
- personnaliser les messages d'erreurs PL/SQL

Traiter des exceptions en PL/SQL

- Une exception est un identificateur PL/SQL détecté pendant la phase d'exécution.
- Comment est-elle déclenchée ?
  - A la suite d'une erreur Oracle.
  - Explicitement, par le programme.
- Comment la traiter ?
  - En l'interceptant à l'aide d'un gestionnaire.
  - En la propageant à l'environnement appelant.

Traiter les exceptions



Types d'exception

- Exception prédéfinie du serveur Oracle
- Exception non prédéfinie du serveur Oracle

} Exception déclenchée implicitement

- Exception définie par l'utilisateur — Exception déclenchée explicitement

Intercepter les exceptions

Syntaxe :

```
EXCEPTION
  WHEN exception1 [OR exception2 . . .] THEN
    statement1;
    statement2;
    . . .
  [WHEN exception3 [OR exception4 . . .] THEN
    statement1;
    statement2;
    . . .]
  [WHEN OTHERS THEN
    statement1;
    statement2;
    . . .]
```


Règles d'interception des exceptions

- Le mot-clé `EXCEPTION` débute la section de traitement des exceptions.
- Plusieurs gestionnaires d'exceptions sont permis.
- Un seul gestionnaire d'exceptions est exécuté avant de sortir du bloc.
- `WHEN OTHERS` est la dernière clause.

Intercepter les erreurs prédéfinies du serveur Oracle

- Utiliser le nom standard à l'intérieur du sous-programme de traitement des exceptions.
- Exemples d'exceptions prédéfinies :
  - `NO_DATA_FOUND`
  - `TOO_MANY_ROWS`
  - `INVALID_CURSOR`
  - `ZERO_DIVIDE`
  - `DUP_VAL_ON_INDEX`

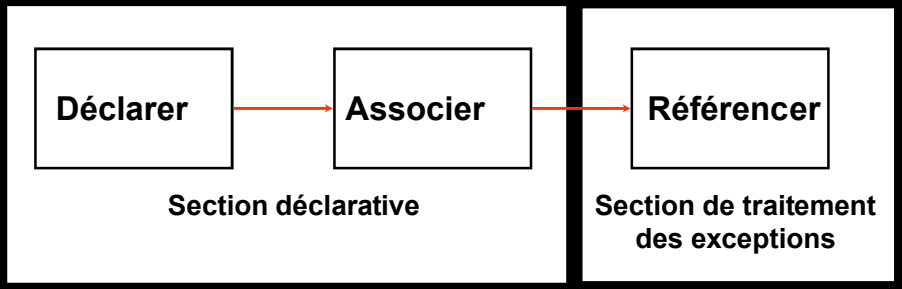
Exceptions prédéfinies

Syntaxe :

```
BEGIN
. . .
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    statement1;
    statement2;

  WHEN TOO_MANY_ROWS THEN
    statement1;
  WHEN OTHERS THEN
    statement1;
    statement2;
    statement3;
END;
```

Intercepter les erreurs non prédéfinies du serveur Oracle



Nommer l'exception

Coder `PRAGMA EXCEPTION_INIT`

Traiter l'exception déclenchée

Erreur non prédéfinie

Intercepter une violation de contrainte d'intégrité (code d'erreur du serveur Oracle –2292).

```
DEFINE p_deptno = 10
DECLARE
  e_emps_remaining EXCEPTION;                    1
  PRAGMA EXCEPTION_INIT                          2
    (e_emps_remaining, -2292);
BEGIN
  DELETE FROM departments
  WHERE  department_id = &p_deptno;
  COMMIT;
EXCEPTION
  WHEN e_emps_remaining  THEN                    3
   DBMS_OUTPUT.PUT_LINE ('Cannot remove dept ' ||
   TO_CHAR(&p_deptno) || '.  Employees exist. ');
END;
```


Fonctions d'interception des exceptions

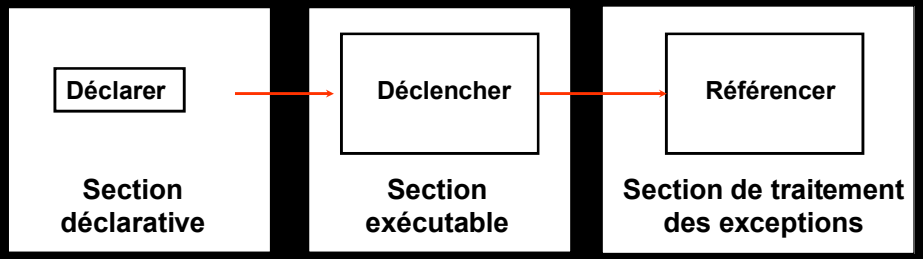
- `SQLCODE` : renvoie la valeur numérique du code d'erreur
- `SQLERRM` : renvoie le message associé au code d'erreur

Fonctions d'interception des exceptions

Exemple :

```
DECLARE
  v_error_code      NUMBER;
  v_error_message   VARCHAR2(255);
BEGIN
...
EXCEPTION
...
  WHEN OTHERS THEN
    ROLLBACK;
    v_error_code := SQLCODE ;          <----
    v_error_message := SQLERRM ;       <----
    INSERT INTO errors
    VALUES(v_error_code, v_error_message);
END;
```

Intercepter les exceptions définies par l'utilisateur



Nommer l'exception.

Déclencher explicitement l'exception par l'instruction `RAISE`.

Traiter l'exception déclenchée.

Exceptions définies par l'utilisateur

Exemple :

```
DEFINE p_department_desc = 'Information Technology '
DEFINE P_department_number = 300
```

```
DECLARE
  e_invalid_department EXCEPTION;                          1
BEGIN
  UPDATE      departments
  SET         department_name = '&p_department_desc'
  WHERE       department_id = &p_department_number;
  IF SQL%NOTFOUND THEN
    RAISE e_invalid_department;                            2
  END IF;
  COMMIT;
EXCEPTION
  WHEN e_invalid_department THEN                           3
    DBMS_OUTPUT.PUT_LINE('No such department id.');
END;
```


Propager des exceptions

Des sous-blocs peuvent traiter une exception ou la transmettre à un bloc englobant.

```
DECLARE
  . . .
  e_no_rows       exception;
  e_integrity     exception;
  PRAGMA EXCEPTION_INIT (e_integrity, -2292);
BEGIN
  FOR c_record IN emp_cursor LOOP
    BEGIN
     SELECT ...
     UPDATE ...
     IF SQL%NOTFOUND THEN
       RAISE e_no_rows;
     END IF;
    END;
  END LOOP;
EXCEPTION
  WHEN e_integrity THEN ...
  WHEN e_no_rows THEN ...
END;
```


Procédure `RAISE_APPLICATION_ERROR`

Syntaxe :

```
raise_application_error (error_number,
                         message[, {TRUE | FALSE}]);
```

- La procédure permet de délivrer des messages d'erreur définis par l'utilisateur à partir de sous-programmes stockés.
- Elle permet de signaler les erreurs à l'application et d'éviter le renvoi d'exceptions non traitées.

Procédure RAISE_APPLICATION_ERROR

- Elle peut être utilisée à deux endroits :
  - section exécutable
  - section de traitement des exceptions
- Elle renvoie à l'utilisateur les conditions de l'erreur de manière cohérente par rapport aux autres erreurs du serveur Oracle

RAISE_APPLICATION_ERROR

Section exécutable :

```
BEGIN
...
  DELETE FROM employees
     WHERE  manager_id = v_mgr;
  IF SQL%NOTFOUND THEN
     RAISE_APPLICATION_ERROR(-20202,
       'This is not a valid manager');
  END IF;
   ...
```

Section de traitement des exceptions :

```
 ...
EXCEPTION
   WHEN NO_DATA_FOUND THEN
     RAISE_APPLICATION_ERROR (-20201,
        'Manager is not a valid employee.');
END;
```


Synthèse

- Types d'exception :
  - erreur prédéfinie du serveur Oracle
  - erreur non prédéfinie du serveur Oracle
  - erreur définie par l'utilisateur
- Intercepter une exception
- Traiter une exception :
  - intercepter l'exception dans un bloc PL/SQL.
  - propager l'exception.

Présentation de l'exercice 8

Dans cet exercice, vous allez :

- traiter des exceptions nommées
- créer et appeler des exceptions définies par l'utilisateur