PL/SQL

Cours 2*

19 février 2019

Note globale : DS Promo (2/3) + DS de groupe (1/3)

Isabelle Gonçalves - isabelle.goncalves@univ-lyon1.fr

* contient des extraits des cours de Guillaume Cabanac, Fouad Dahak, Yacine Bouhabel, Amrouche Karima, Richard Grin, Adnene Belfodil, Anes Bendimerad, Alain Pillot, Mooneswar Ramburrun + de la documentation Oracle

Introduction

Comment lier des requêtes SQL, en utilisant des variables et des structures de contrôle (boucles et alternatives), pour les traitements complexes ?

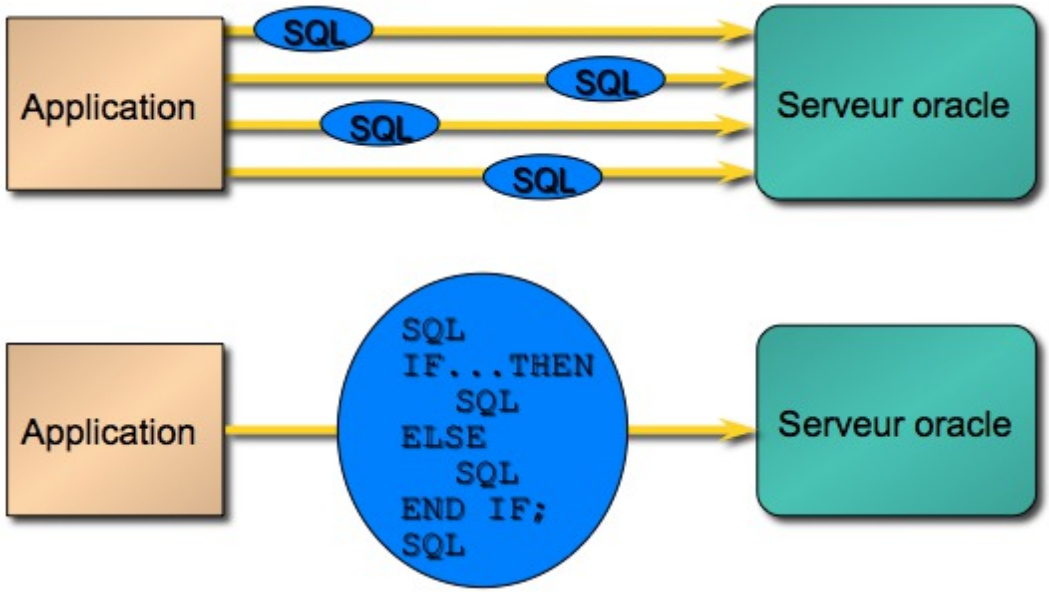
- Extension procédurale du langage SQL : PL/SQL
  - variables
  - structures de contrôle
  - gestion des erreurs
  - structurée en blocs

Le langage PL/SQL

Permet d'inclure des requêtes d'interrogation (LID) et des ordres de manipulation de données (LMD) à l'intérieur d'une structure algorithmique.

Introduction

- PL/SQL peut améliorer les performances d'une application (un seul appel au serveur => réduction du trafic réseau)



Le langage PL/SQL

C'est un langage spécifique à Oracle. Il existe d'autres langages pour les autres SGBD (exemples : T-SQL pour Microsoft SQL Server, PL/pgSQL pour PostgreSQL)

Introduction

- Trois types de blocs : Anonyme, Procédure ou Fonction.
- Un bloc peut contenir d'autres blocs, et il doit contenir au moins une instruction (null est une instruction autorisée).

```
[DECLARE]
        - - déclaration des variables, curseurs, exceptions utilisateur
BEGIN
        - - instructions SQL ou PL/SQL
[EXCEPTION]
        - - actions à exécuter en cas d'erreurs
END ;
```

Introduction

- Chaque instruction se termine par ;
- Les commentaires sur une ligne sont précédés de - - et les multilignes sont entre /* */
- La casse n'a pas d'incidence sur les instructions et les variables.

```
DECLARE
        v_variable VARCHAR2(5);
BEGIN
        SELECT nom_colonne
        INTO v_variable
        FROM nom_table;
EXCEPTION
        WHEN exception-nom-erreur THEN
...
END;
```

Variables : Déclaration

Déclaration

nom-variable [CONSTANT] type-variable [NOT NULL] [:= | DEFAULT expression] ;

```
DECLARE
        v_hiredate DATE;
        v_deptno NUBER(2) NOT NULL :=10;
        v_location VARCHAR2(13) :='Atlanta';
        c_comm CONSTANT NUMBER := 1400;
```

- Possibilité de déclarer et d'affecter en même temps
- Options possibles :
  - DEFAULT valeur
  - initialisation obligatoire pour NOT NULL et pour CONSTANT

Variables : Affichage

- Pour activer les affichages en sortie d'Oracle dans SQL Developer :

```
SET SERVEROUTPUT ON;
```

- On utilise les procédures du package Oracle DBMS_OUTPUT
- Procédure PUT_LINE : affichage de la valeur d'une variable.
  - || pour faire des concaténations
  - chr(10) pour un \n
  - chr(9) pour une tabulation

```
DBMS_OUTPUT.PUT_LINE (chr(9) || 'Salaire mensuel : ' || v_sal);
```

Variables : Types

Leur type peut être :

- simples (scalaires) : VARCHAR2, NUMBER, DATE, BOOLEAN ...
- peut être déterminé par référence avec %TYPE derrière :
  - nomTable.nomColonne
  - nomVariable

```
v_ename emp.ename%TYPE
v_salary NUMBER(8,2)
v_min v_salary%TYPE
```

- composés : RECORD (enregistrement) et collections
- non structurés : LOB (données de grandes tailles : textes, images ...)

Variables : Affectation

- opérateur d'affectation

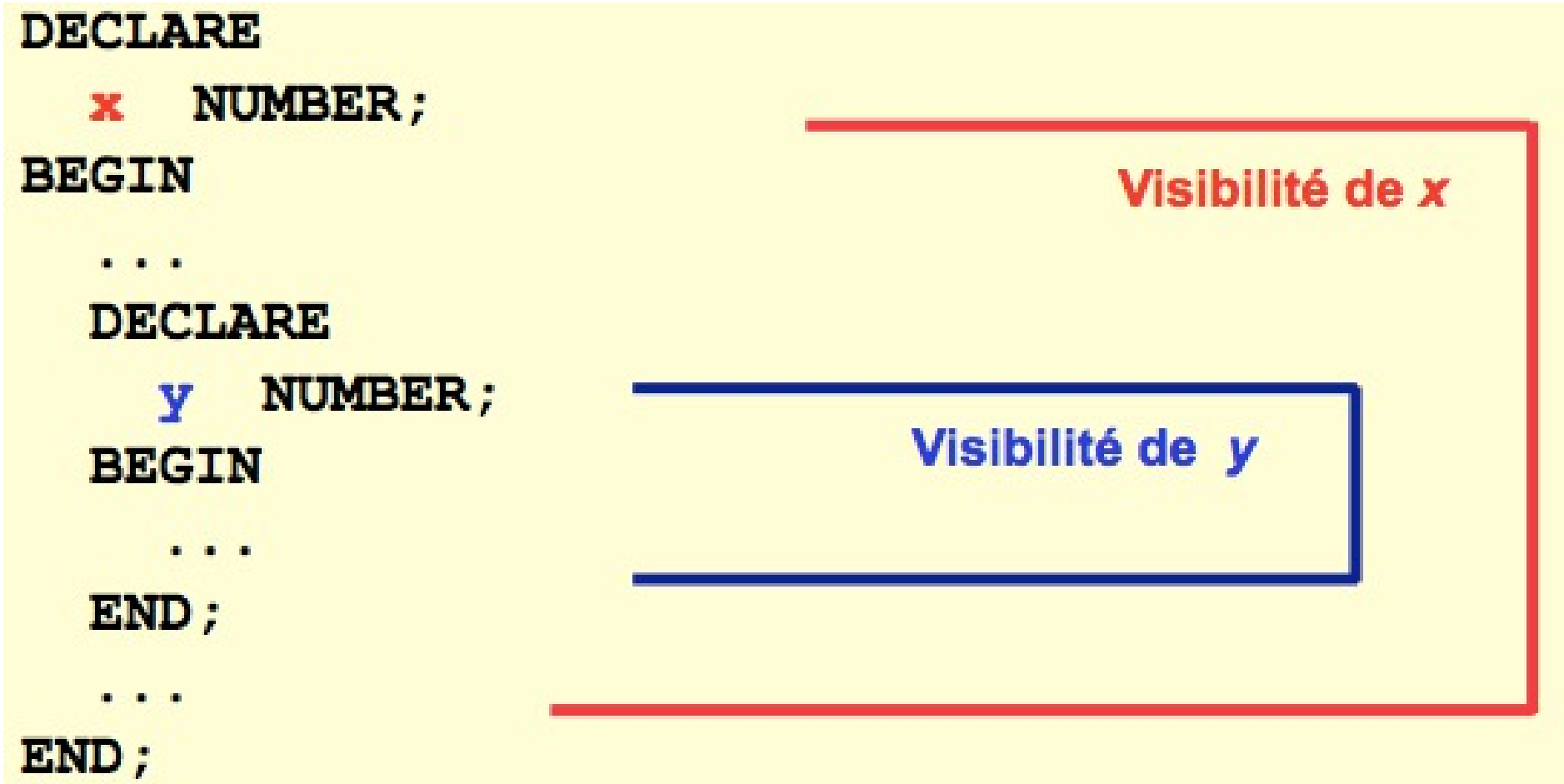
```
nom_variable := expression
```

- ordre SELECT inclus dans un bloc PL/SQL

```
SELECT col1, col2
INTO nom_variable1, nom_variable2
FROM nom_table
```

Variables : Visibilité

Portée habituelle des langages à blocs



Variables : type RECORD

Permet de stocker une collection de données de types différents
Permet de stocker une ligne d'une table par exemple.

Déclaration 1ère possibilité

```
nom_variable_enregistrement nom_table%ROWTYPE;
```

Déclaration 2ème possibilité

```
TYPE nom_type IS RECORD (nom_champ1 type_champ1,
nom_champ2 type_champ2...);
nom_variable_enregistrement nom_type;
```

```
SELECT *
INTO nom_variable_enregistrement
FROM nom_table
WHERE id = 40;
```

Structures de contrôle : l'alternative

- IF

```
IF condition THEN
        instructions
[ELSIF conditions THEN
        instructions ]
[ELSE
        instructions ]
END IF ;
```

Structures de contrôle : l'alternative

- CASE

```
CASE variable - - nombre entier ou chaîne
      WHEN valeur1 THEN
            instructions
      WHEN valeur2 THEN
            instructions
       ...
      [ELSE
            instructions ]
END CASE ;
```

D'autres formes de CASE existent.

Structures de contrôle : l'alternative

- exemple d'une autre forme de CASE

```
val := CASE city
       WHEN 'TORONTO THEN 'RAPTORS'
       WHEN 'LOS ANGELES' THEN 'LAKERS'
       ELSE 'NO TEAM'
END;
```

Structures de contrôle : les boucles

- le FOR

```
FOR indice in [REVERSE] borneInf..borneSup
LOOP
        instruction1 ;
        instruction2 ;
        ...
END LOOP ;
```

- déclaration implicite de l'indice
- indice inutilisable en dehors de la boucle
- interdiction d'affecter une valeur à l'indice

Structures de contrôle : les boucles

- la LOOP

```
LOOP
        instruction1 ;
        ...
        EXIT [WHEN condition] ;
END LOOP ;
```

- le WHILE

```
WHILE condition LOOP
        instruction1 ;
        ...
END LOOP ;
```

LDD : Exercices

Faire les exercices 1 et 2 du TD2

Les transactions

Une transaction est un ensemble d'ordres (SQL) indivisibles, faisant passer la base de données d'un état cohérent à un autre en une seule étape.

- Début de transaction (implicite)
  - au premier ordre SQL rencontré au départ ou après une transaction.
- Fin de transaction
  - implicite après un ordre LDD
  - explicite avec COMMIT (validation) ou ROLLBACK (annulation)

En cours de transaction, seul l'utilisateur ayant effectué les modifications les voit.

Les curseurs

- Tout ordre SQL utilise un curseur (zone de travail privée) pour s'exécuter
- Les curseurs implicites sont créés pour chaque ordre SQL et sont nommés SQL
- Le curseur implicite est associé à un code statut à la fin de l'exécution de l'ordre SQL :
  - SQL%ROWCOUNT le nombre de lignes traitées
  - SQL%FOUND vrai si au moins une ligne traitée
  - SQL%NOTFOUND vrai si aucune ligne traitée

En PL/SQL, on a parfois besoin de récupérer plusieurs lignes correspondant à une contrainte particulière. La directive SELECT... INTO ne permet de récupérer qu'une seule ligne à la fois. Les curseurs explicites vont permettre l'accès multilignes.

Les curseurs implicites

Exemple pour afficher le nombre de lignes supprimées.

```
BEGIN
        DELETE FROM contrat
        WHERE id_cnt like '0016000%';
        DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT || ' rows
deleted.');
END;
```

Les curseurs explicites

- Déclarés explicitement par le programmeur
- Possèdent un code statut en plus, %ISOPEN, vrai si le curseur est ouvert
- Utilisation en 4 étapes :
  - Déclaration (dans la zone DECLARE)
  - Ouverture (OPEN) : exécute la requête et génère des lignes de résultat côté serveur
  - Avancement ligne par ligne (FETCH)
  - Fermeture (CLOSE) : libération des ressources

Remarque

Possibilité de ré-ouvrir le même curseur

Les curseurs explicites : syntaxe

- Déclaration

```
CURSOR nom_curseur IS
        un énoncé SELECT ;
```

Remarque

Ne pas inclure la clause INTO dans la déclaration du curseur.

- Ouverture

```
OPEN nom_curseur ;
```

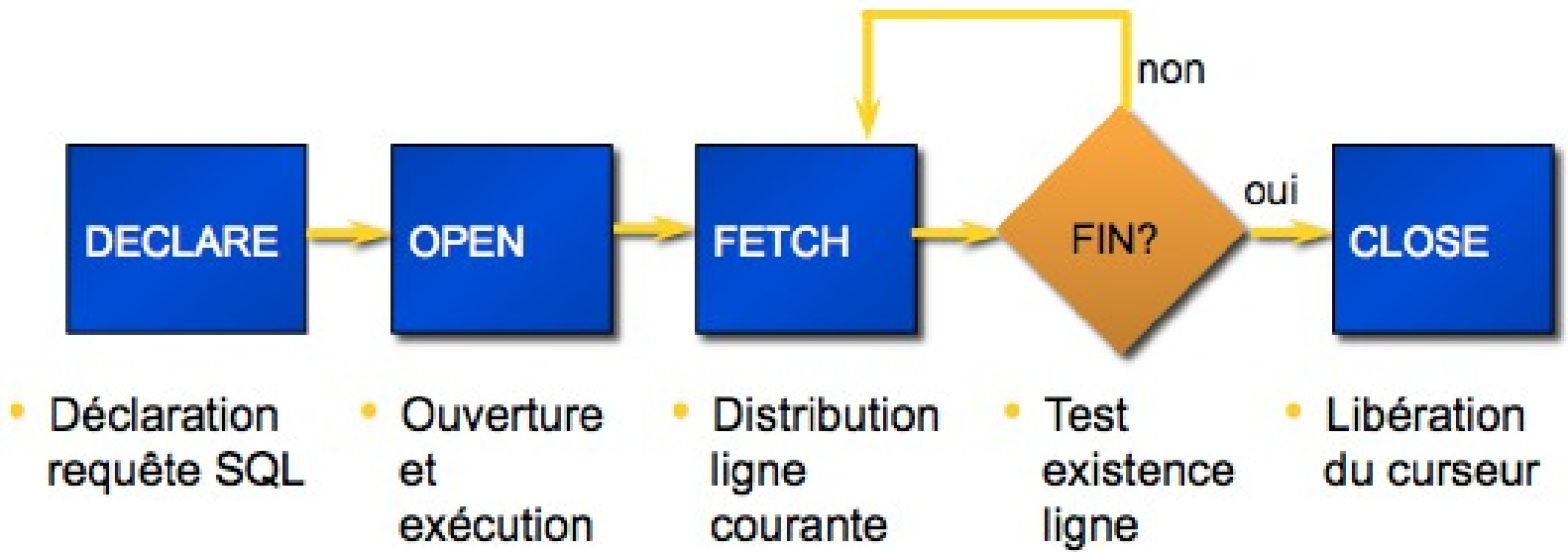
- Distribution des lignes

```
FETCH nom_curseur INTO [variable1, variable2,...] |
[nom_enregistrement] ;
```

- Fermeture

```
CLOSE nom_curseur ;
```

Les curseurs explicites : mise en oeuvre



Remarques

- La commande FETCH distribue la ligne courante et met les informations dans des variables
- La commande FETCH renvoie un code statut qui peut être testé

Les curseurs explicites : exemple

```
DECLARE
 CURSOR c_emp IS SELECT eno, ename, title FROM emp WHERE
city='TORONTO';
  v_emp c_emp%ROWTYPE;
BEGIN
 OPEN c_emp;
 LOOP
   FETCH c_emp INTO v_emp;
   EXIT WHEN c_emp%NOTFOUND;
   DBMS_OUTPUT.PUT('numero :'||v_emp.eno);
   DBMS_OUTPUT.PUT(' nom :'||v_emp.ename);
   DBMS_OUTPUT.PUT_LINE(' situation :'||v_emp.title);
 END LOOP;
 CLOSE c_emp;
END;
```

Les curseurs explicites : FOR

- Simplification de syntaxe car exécute implicitement OPEN, FETCH et CLOSE
- Déclare implicitement le RECORD
- Syntaxe :

```
FOR nom_enregistrement IN nom_curseur LOOP
        instruction1 ;
        ... ;
END LOOP ;
```

Les curseurs explicites : exemple

```
DECLARE
 CURSOR c_emp IS SELECT eno, ename, title FROM emp WHERE
city='TORONTO';
BEGIN
 FOR v_emp IN c_emp LOOP
  DBMS_OUTPUT.PUT('numero :'||v_emp.eno);
  DBMS_OUTPUT.PUT(' nom :'||v_emp.ename);
  DBMS_OUTPUT.PUT_LINE(' situation :'||v_emp.title);
 END LOOP;
END;
```

```
numero :E1 nom :KRUNAL situation :MANAGER
numero :E2 nom :KAUSHIK situation :PROGRAMMER
numero :E3 nom :RAMESH situation :SUPPORT STAFF
numero :E4 nom :JAGDISH situation :SUPPORT STAFF
```

Les curseurs explicites : paramètres

- Un curseur paramétré peut servir plusieurs fois dans un même bloc avec des valeurs de paramètres différentes
- Affectation des valeurs de paramètres lors de l'ouverture du curseur
- On doit fermer le curseur entre chaque utilisation de paramètres différents (rappel : le FOR ferme le curseur automatiquement)
- Syntaxe :

```
CURSOR nom_curseur (paramètre1 type[, ...])
IS
        requête ;
```

Les curseurs explicites : paramètres

```
DECLARE
 CURSOR c_emp (p_dept NUMBER) IS
  SELECT dept, nom FROM emp WHERE dept = p_dept;
BEGIN
 FOR v_emp IN c_emp(10) LOOP
  DBMS_OUTPUT.PUT_LINE('Nom : ' || UPPER(v_emp.nom));
 END LOOP;
 FOR v_emp IN c_emp(20) LOOP
  DBMS_OUTPUT.PUT_LINE('Nom : ' || UPPER(v_emp.nom));
 END LOOP;
END;
```

Les curseurs explicites : clause FOR UPDATE

- Utilisée lors des mises à jour avec un curseur
- Verrouille les lignes sélectionnées, avant l'exécution d'un ordre UPDATE ou DELETE, tant que le curseur n'a pas traité la ligne.
- Libération des lignes avec COMMIT ou ROLLBACK

Syntaxe

```
SELECT ...
...
FROM nom_table
FOR UPDATE [OF nom_col] [NOWAIT];
```

- NOWAIT retourne le contrôle à l'utilisateur si la table est verrouillée par un autre utilisateur.

Les curseurs explicites : clause WHERE CURRENT OF

- Utilisée pour supprimer ou modifier une ligne référencée par la clause FOR UPDATE
- Concerne la dernière ligne distribuée par le curseur

Syntaxe

```
WHERE CURRENT OF nom_curseur
```

Les curseurs explicites : clause WHERE CURRENT OF

```
DECLARE
 CURSOR sal_cursor IS
  SELECT sal FROM emp WHERE deptno = 30
  FOR UPDATE OF sal NOWAIT;
BEGIN
 FOR v_emp IN sal_cursor LOOP
  UPDATE emp
  SET sal = v_emp.sal * 1.1
  WHERE CURRENT OF sal_cursor;
 END LOOP;
 COMMIT;
END;
```

LDD : Exercices

Faire l'exercice 3 du TD2

Les exceptions

Ce sont des erreurs qui surviennent durant une exécution. Il en existe de 2 types :

- déclenchées implicitement par le serveur
  - Prédéfinies (possèdent un nom)
  - Non prédéfinies (sans nom)
- Définies et déclenchées explicitement par l'utilisateur

Capture de l'exception :

- Une exception ne provoque pas nécessairement l'arrêt du programme si elle est capturée par un bloc (dans la partie EXCEPTION)
- Une exception non capturée se propage à l'environnement (bloc de niveau supérieur ou procédure appelante).

Les exceptions : capture

```
EXCEPTION
 WHEN exception1 [OR exception2...] THEN
  instruction1 ;
  instruction2 ;
  ...
[ WHEN exception2 [OR exception4...] THEN
  instruction1 ;
  instruction2 ;
  ...]
[ WHEN OTHERS THEN
  instruction1 ;
  instruction2 ;
  ...]
```

Les exceptions serveur prédéfinies

Elles sont émises par le serveur (SGBDR) et sont repérables par leur nom.
Par exemple :

- NO_DATA_FOUND : quand le SELECT...INTO ne ramène aucune ligne
- TOO_MANY_ROWS : quand le SELECT...INTO ramène plusieurs lignes
- ZERO_DIVIDE : Division par 0
- VALUE_ERROR : erreur numérique (conversion de chaîne en nombre)
- ...

Informations associées à toute erreur :

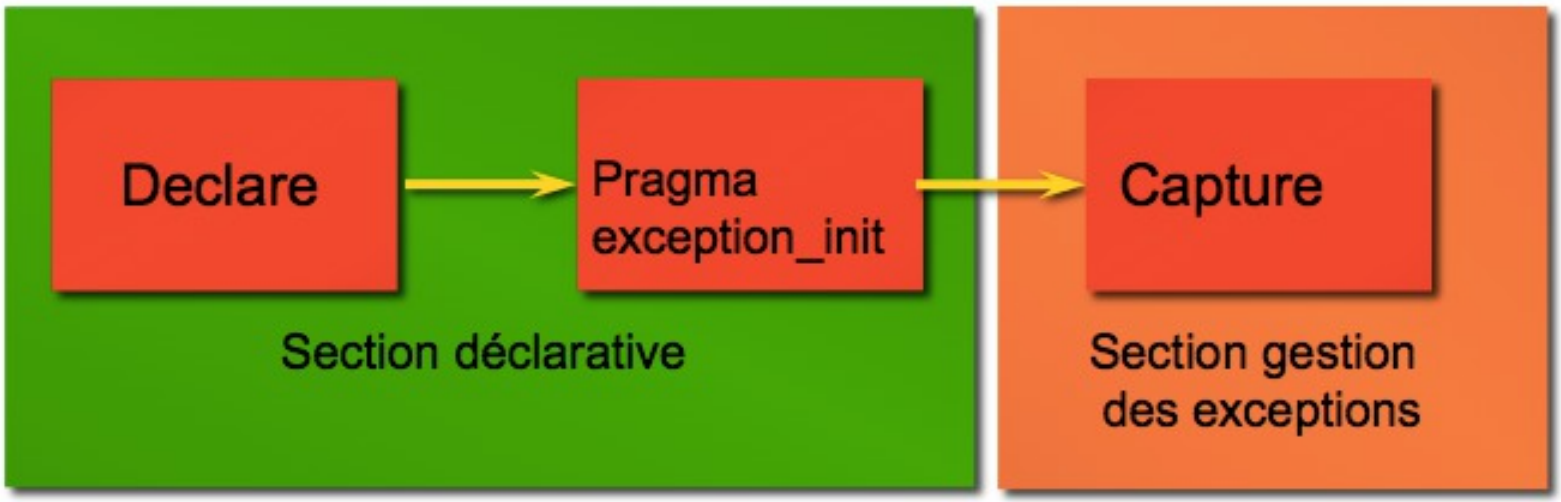
- SQLCODE : Valeur numérique de l'erreur (généralement négative)
- SQLERRM : Texte du message associé à l'erreur

```
EXCEPTION
 WHEN OTHERS THEN
  dbms_output.put_line(SQLCODE || ' : ' || SQLERRM);
```

Les exceptions serveur prédéfinies : exemple

```
DECLARE
  v_nom VARCHAR2(15);
BEGIN
  SELECT nom INTO v_nom FROM emp WHERE prenom='John';
 DBMS_OUTPUT.PUT_LINE('Le nom de John est : ' || v_nom);
EXCEPTION
 WHEN TOO_MANY_ROWS THEN
  DBMS_OUTPUT.PUT_LINE('too many rows returned by the query');
END;
```

Les exceptions serveur non prédéfinies

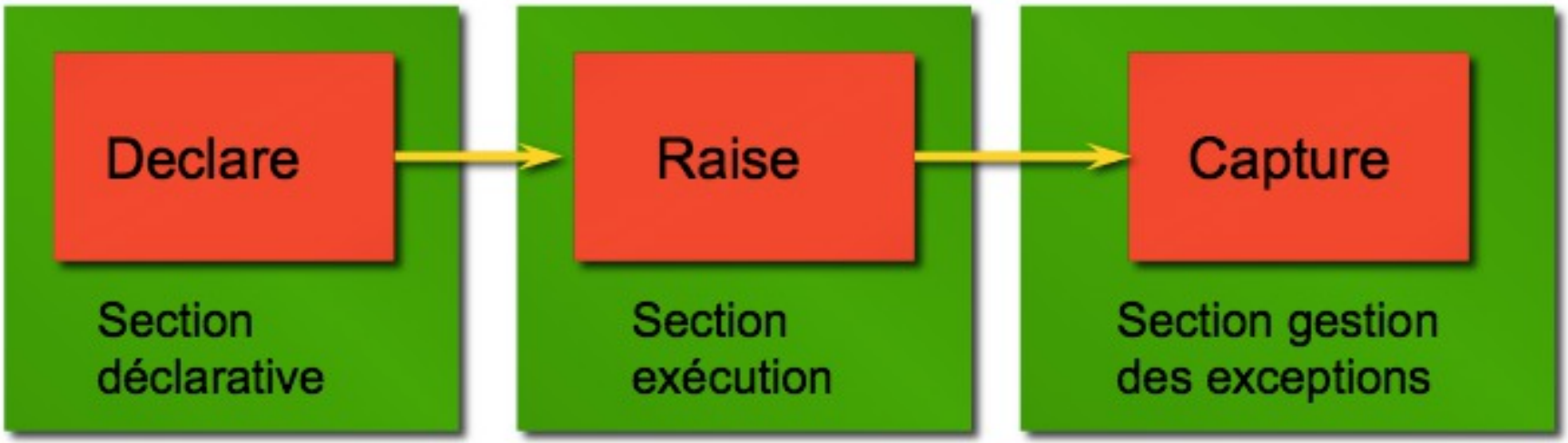


- Déclaration d' une d' exception
- Association à l' erreur serveur
- Traitement de l' exception

Les exceptions serveur non prédéfinies : exemple

```
DECLARE
   e_emps_remaining EXCEPTION;
   PRAGMA EXCEPTION_INIT (e_emp_remaining, -2292);
- - ORA-002292 integrity constraint violated - child record found
BEGIN
   DELETE FROM dept
   WHERE deptno = 20;
   COMMIT;
EXCEPTION
 WHEN e_emps_remaining THEN
  DBMS_OUTPUT.PUT_LINE('Suppression impossible
   car des employés existent');
END;
```

Les exceptions utilisateur



- Déclaration d'un nom d'exception
- Génération de l'exception
- Traitement de l'exception

Les exceptions utilisateur : exemple

```
DECLARE
  v_salaire NUMBER(10,2);
  TOO_LOW EXCEPTION;
BEGIN
  SELECT sal INTO v_salaire FROM emp WHERE dept='50';
  IF v_salairel < 300 THEN
    RAISE TOO_LOW;
 END IF;
EXCEPTION
 WHEN TOO_LOW THEN
 DBMS_OUTPUT.PUT_LINE('salaire trop bas');
END;
```

La procédure RAISE_APPLICATION_ERROR

- Utilisable :
  - dans la section Exécution
  - dans la section Exception
- La génération de l'erreur est conforme au standard du serveur et est traitable comme telle
- Le numéro d'erreur doit être compris entre -20,000 et -20,999

```
EXCEPTION
 WHEN NO_DATA_FOUND THEN
  RAISE_APPLICATION_ERROR(-20201, 'Ligne NON trouvée');
END;
```

LDD : Exercices

Faire l'exercice 4 du TD2

Les fonctions et procédures

- Ce sont des blocs PL/SQL nommés
- Elles sont stockées dans la base de données
- Elles peuvent prendre ou pas des paramètres :
  - en entrée : mode IN (seul conseillé pour les fonctions)
  - en sortie : mode OUT
  - en entrée/sortie : mode IN OUT
- 0 (procédure) ou 1 (fonction) valeur de retour
- Elles sont dans la table user_source

Les fonctions : syntaxe

```
CREATE [OR REPLACE] FUNCTION nom_fonction
- - type sans spécification de taille
   [(paramètre1 [mode1] type1,
   (paramètre2 [mode2] type2    ...)]
RETURN type
IS | AS
– Section déclarative
BEGIN
– Section exécution
[EXCEPTION
– Section exception]
END [nom_fonction] ;
```

Les procédures : syntaxe

```
CREATE [OR REPLACE] PROCEDURE nom_procedure
- - type sans spécification de taille
   [(paramètre1 [mode1] type1,
   (paramètre2 [mode2] type2    ...)]
IS | AS
– Section déclarative
BEGIN
– Section exécution
[EXCEPTION
– Section exception]
END [nom_fonction] ;
```

Mode d'un paramètre

IN	OUT	IN OUT
Par défaut	Doit être spécifié	Doit être spécifié
Valeur transmise à la procédure	Valeur transmise à l'environnement	Valeur transmise à la procédure et renvoyée à l'environnement
Variable, constante ou expression	Variable	Variable

Remarque sur les fonctions

Il est conseillé d'éviter OUT et IN OUT avec des fonctions. Le but d'une fonction est de prendre zéro ou plus de paramètres et retourner une seule valeur. Les fonctions doivent être exemptes d'effets de bord, qui modifient les valeurs des variables non locales au sous-programme.

Procédure ou fonction ?

Procédure	Fonction
Exécutée comme une instruction PL/SQL	Utilisée dans une expression
Pas de type de retour	Type de Retour défini
Peut renvoyer plusieurs valeurs résultats	Une seule valeur résultat

Les fonctions : exemple

```
CREATE OR REPLACE FUNCTION GET_MAX (
  NB1 IN NUMBER,
  NB2 IN NUMBER
) RETURN NUMBER AS
BEGIN
  IF NB1 >= NB2 THEN
   RETURN NB1;
  ELSE
   RETURN NB2;
  END IF;
END GET_MAX;
```

Les fonctions : exemples d'appel

Dans une requête SQL

```
SELECT get_max (1, 2)
FROM dual;
```

La table DUAL du SGBD Oracle est une table spéciale avec une seule ligne et une seule colonne.

Dans un bloc PL/SQL

```
BEGIN
  DBMS_OUTPUT.PUT_LINE(GET_MAX(3,4));
END;
```

Avec la commande EXEC[UTE] de SQL*Plus

```
VARIABLE M NUMBER
EXEC :M := GET_MAX(1000,500)
PRINT M
```

Les procédures : exemple

```
CREATE OR REPLACE PROCEDURE QUERY_EMP (
  v_id IN emp.empno%TYPE,
  v_name OUT emp.ename%TYPE,
  v_salary OUT emp.sal%TYPE)
IS
BEGIN
  SELECT ename, sal INTO v_name, v_salary
  FROM emp WHERE empno =v_id;
END QUERY_EMP;
```

Mode OUT et SQL*Plus

```
VARIABLE gname VARCHAR2(15)
VARIABLE gsalary NUMBER
EXEC QUERY_EMP(7654, :gname, :gsalary)
PRINT gname
PRINT gsalary
```

Les procédures : exemple

```
CREATE OR REPLACE PROCEDURE FORMAT_PHONE
  (v_phone_no IN OUT VARCHAR2) IS
BEGIN
  v_phone_no := '(' || SUBSTR(v_phone_no,1,3) ||
   ')' || SUBSTR(v_phone_no,4,3) || '-' || SUBSTR(v_phone_no,7);
END FORMAT_PHONE;
```

Mode INTOUT et bloc PL/SQL

```
DECLARE
   g_phone_no varchar2(15);
BEGIN
   g_phone_no := '8006330575';
   format_phone(g_phone_no); /* (800)633-0575 */
   DBMS_OUTPUT.PUT_LINE(g_phone_no);
END;
```