Bases de Données

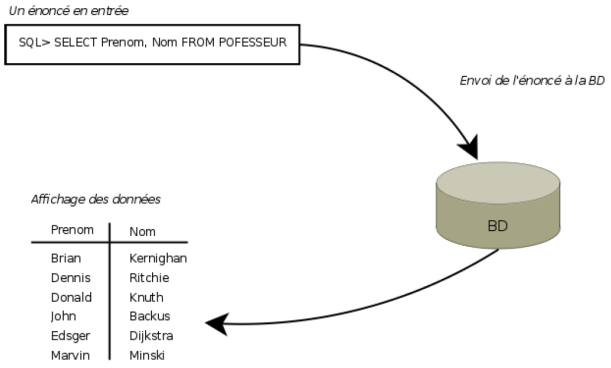
C3-PL/SQL

Lina Soualmia

Université de Rouen
LITIS - Équipe TIBS-CISMeF
lina.soualmia@chu-rouen.fr

16 septembre 2015

Plan



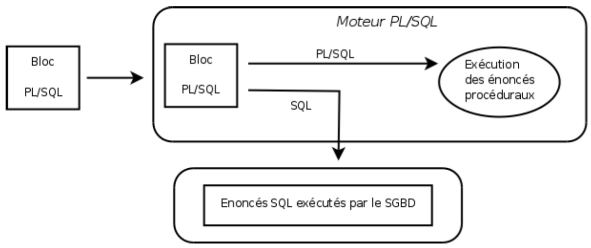
Clauses SQL

SELECT	Interrogation des données
INSERT UPDATE DELETE	Langage de Manipulation de Données (LMD)
CREATE ALTER DROP RENAME TRUNCATE	Langage de Définition de Données (LDD)
GRANT REVOKE	Langage de Contrôle de Données (LCD)
COMMIT ROLLBACK SAVEPOINT	Contrôle de transaction

PL/SQL

- Procedural Language for Structured Query Language
- Langage fournissant une interface procédurale au SGBD Oracle
- Intégration du langage SQL en lui apportant une dimension procédurale
- Réalisation de traitements algorithmiques (ce que ne permet pas SQL)
- Mise à disposition de la plupart des mécanismes classiques de programmation des langages hôtes tels que C, PASCAL, C++, JAVA ...

Environnement PL/SQL :

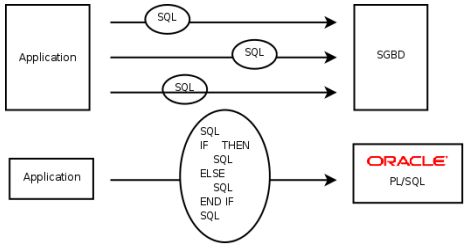


Avantages de PL/SQL

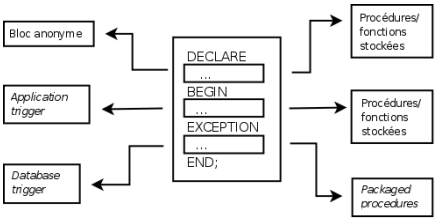
- PL/SQL complément de SQL (qui n'est pas procédural)
- Mécanismes offerts par PL/SQL :
  - Structures itératives : `WHILE` *** `LOOP`, `FOR` *** `LOOP`, `LOOP` ***
  - Structures conditionnelles : `IF` *** `THEN` *** `ELSE` , `ELSEIF` *** `ENDIF`, `CASE` ***
  - Déclaration des curseurs et des tableaux
  - Déclaration de variables
  - Affectation de valeurs aux variables
  - Branchements : `GOTO`, `EXIT`
  - Exceptions : `EXCEPTION`

Avantages de PL/SQL

- Intégration : Meilleure cohérence du code avec les données Utilisation de librairies standards prédéfinies
- Blocs : Traitement par/de blocs SQL dans un énoncé PL/SQL (optimisation des transactions réseaux)



- Exécution de programmes modulaires
- Traitements complexes (cas particuliers, erreurs)
- Traitements des exceptions



En résumé :

- Langage portable
- Utilisation de variables de stockage
- Utilisation de type simple ou de type structuré dynamiquement (%TYPE, %ROWTYPE, ...etc.)
- Utilisation des structures de contrôle des langages procéduraux
- Gestion et manipulation des erreurs
- Création d'ordres SQL dynamiques

PL - SQL

- Langage portable
- Utilisation de variables de stockage
- Utilisation de type simple ou de type structuré dynamiquement (%TYPE,% ROWTYPE, ...etc.)
- Utilisation des structures de contrôle des langages procéduraux
- Gestion et manipulation des erreurs
- Création d'ordres SQL dynamiques

Utilisation de PL/SQL sous 3 formes

- Bloc de code, exécuté comme une commande SQL
- Fichier de commandes PL/SQL
- Programme stocké :
  - procédure,
  - fonction,
  - package
  - ou trigger (déclencheur)

Structure d'un bloc PL/SQL

```
-- Section déclarative, optionnelle
DECLARE
      Variables, curseurs, exceptions, ...

-- Section exécutable, obligatoire
 BEGIN
      Instructions SQL et PL/SQL
      Possibilités de blocs fils (imbrication de blocs)

-- Section de traitement des exceptions, optionnelle
 EXCEPTION optionnelle
       Traitement des exceptions (gestion des erreurs)

-- Terminaison du bloc, obligatoire
 END ;
/
```


Types de blocs :

- Bloc anonyme
- Procédure
- Fonction

Bloc anonyme :

- Structure classique (1 à 3 sections)
- Un bloc ne peut être vide. Il doit contenir une instruction (il peut donc contenir l'instruction NULL).

```
[DECLARE]
BEGIN
    Instructions
[EXCEPTION]
END;
/
```


Bloc anonyme : exemple

```
DECLARE x INTEGER;
BEGIN
x := 1 ;
END;
```

```
DECLARE
    varx VARCHAR2(5) .
BEGIN
    SELECT nom_ colonne
    INTO varx
    FROM nom_table
EXCEPTION
    WHEN nom_exception THEN
    ...
END ;
/
```


Procédure :

Bloc PL/SQL nommé puis compilé et stocké dans la base

```
PROCEDURE nom IS
BEGIN
    Instructions
[EXCEPTION]
END;
/
```



```
PROCEDURE procedure_exemple IS
    varx VARCHAR2(5)
    BEGIN
        SELECT nom_colonne
        INTO varx
        FROM nom_table
    EXCEPTION
        WHEN nom_exception THEN
        ...
    END ;
/
```

Commande SQL qui crée la procédure PL/SQL compile et stocke dans la base le bloc PL/SQL compris entre le BEGIN et le END, référencé par 'procedure_exemple'

Procédure

Remarques :

- Exécution (auto) de la procédure :
  - SQL> `EXECUTE proc_exemple`
- Pas d'exécution de procédure (ou d'instructions) en fin de transaction (`COMMIT`, `ROLLBACK`, Ordre DDL)
- La décision d'enregistrement ou d'annulation de la transaction en cours est à réaliser par le programme appelant la procédure

Fonction :

Procédure retournant une valeur

```
FUNCTION nom_fonction RETURN type_données IS
    BEGIN
        Instructions
    RETURN valeur;
    [EXCEPTION]
    END;
/
```



```
CREATE OR REPLACE FUNCTION solde (nc INTEGER)
    RETURN REAL IS le_solde REAL;
    BEGIN
        SELECT solde INTO le_solde FROM clients
        WHERE no_cli = nc;
        RETURN le_solde;
    END;
/
```


Exemple : utilisation de fonction au sein d'une requête SQL

```
SQL> SELECT solde(1000) FROM dual ;
Solde(1000)
-----------
12024,50
```

L'appel d'une fonction comme une procédure provoque une erreur. Exemple : fonction mdp(INTEGER)

```
SQL> execute mdp(2);
BEGIN mdp(2); END;
*
ERREUR a la ligne 1 :
ORA-06550: line 1, column 7:
PLS-00221: 'MDP' is not a procedure or is undefined
```


Types de variables :

- Variables locales
  - Constantes
  - Composites
  - Références
- Variables de l'environnement extérieur à PL/SQL
  - Attachées (Bind)
  - Hôtes (Host)

Déclaration des variables

Syntaxe :

```
Identificateur [CONSTANT] type_de_données [NOT NULL] [:=expression];
```

Exemple

```
DECLARE
    var_date_naissance  DATE;
    var_departement     NUMBER(2) NOT NULL := 10;
    var_ville           VARCHAR2(13) := 'Rouen';
    c_cumul             CONSTANT NUMBER := 1000;
```


Assignation des variables

Syntaxe :

```
Identificateur := expression ;
```

Exemples :

- Affecter la date de naissance du fils d'un employé

```
var_date_naissance := '23-SEP-2004';
```

- Fixer le nom d'un employé à 'Clémentine'

```
var_nom := 'Clémentine';
```


Exemples

```
v_job               VARCHAR2(9);
v_count             BINARY INTEGER := 0 ;
v_salaire_total     NUMBER(9,2);
v_date              DATE := SYSDATE +7;
v_taux_taxe         CONSTANT NUMBER(3,2) := 8.25 ;
v_valide            BOOLEAN NOT NULL := TRUE;
```


Attribut TYPE

- Définition : déclaration d'une variable associée à :
  - une colonne d'une table dans la BD
  - une variable précédemment définie

```
v_nom           emp.nom%TYPE;
v_sal_annuel    NUMBER(7,2);
v_sal_mensuel   v_sal_annuel%TYPE := 2000;
```


Bloc PL/SQL

Syntaxe et directives :

- Les instructions peuvent être écrites sur plusieurs lignes
- Les unités lexicales peuvent être séparées par des espaces
  - Délimiteurs
  - Identificateurs
  - Littéraux (ou constantes)
  - Commentaires

Les littéraux

- Les dates et les chaînes de caractères délimitées par deux simples côtes

```
v_nom := 'Loulou le pou' ;
```

- Les nombres peuvent être des valeurs simples ou des expressions

```
v_annee := to_number ( to_char ( v_date_fin_periode, 'YY ' ) )
```


Commentaires dans le code :

- Précéder un commentaire écrit sur une seule ligne par '– –'.
- Placer un commentaire écrit sur plusieurs lignes entre les symboles '/*' et '*/'.
- Exemple :

```
v_salaire NUMBER(9, 2);
BEGIN
/* Ceci est un commentaire qui peut
être écrit sur plusieurs lignes (ici 2 lignes)*/
END; --Ceci est un commentaire sur une ligne
```


Les Fonctions SQL en PL/SQL :

- Les fonctions sur les nombres
- Les fonctions sur les chaînes de caractères
- Les fonctions de conversion de type de données
- Les fonctions de dates

Exemples :

- Recomposer l'adresse d'un employé :

```
v_adr_complete := v_rue||CHR(32)||v_ville||CHR(32)
```

- Convertir le nom en majuscule

```
v_nom := UPPER(v_nom) ;
```

- Extraction d'une partie de la chaîne

```
v_chr := Substr('PL/SQL',4,3);
```

- Replacement d'une chaîne par une autre

```
v_chr := Replace('Serv1/Prod/tb_client','Prod','Valid');
```


Blocs imbriqués et portée des variables :

```
x BINARY INTEGER ;
BEGIN --Début de la portée de x
...
DECLARE
    y NUMBER;
    BEGIN --Début de la portée de y
    ...
    END; --Fin de la portée de y
    ...
END; --Fin de la portée de x
```


Opérateurs dans PL/SQL :

- Identiques à SQL :
  - Logique
  - Arithmétique
  - Concaténation
  - Parenthèses pour contrôler l'ordre des opérations
- Opérateur supplémentaire :
  - Opérateur d'exponentiation **

Utilisation des variables liées :

- Pour référencer une variable en PL/SQL, on doit préfixer son nom par un ' :'
- Exemple :

```
: code_retour := 0 ;
IF verifier_crédit_ok (compt no) THEN
    : code_retour := 1 ;
END IF ;
```


Instructions SQL dans PL/SQL :

- Extraire une ligne de données à partir de la base de données par la commande SELECT : un seul ensemble de valeurs peut être retourné.
- Effectuer des changements aux lignes dans la base de données par les commandes du LMD
- Contrôler des transactions avec les commandes COMMIT, ROLLBACK, SAVEPOINT
- Déterminer les résultats du LMD avec des curseurs implicites

Instruction SELECT dans PL/SQL :

- Récupérer une donnée de la base avec SELECT.
- Syntaxe :

```
SELECT liste_sélection
INTO {nom_var [ , nom_var ] ...| nom_record}
FROM nom_table
WHERE condition;
```


Instruction SELECT dans PL/SQL :

- La clause INTO est obligatoire
- Exemple

```
DECLARE
    v_deptno NUMBER(2);
    v_loc VARCHAR2(15);
BEGIN
    SELECT deptno, loc
    INTO v_deptno, v_loc
    FROM dept
    WHERE nom_d = 'INFORMATIQUE';
    ...
END;
```


Instruction SELECT dans PL/SQL :

- Retourne la somme des salaires de tous les employés d'un département donné :

```
DECLARE
    v_som_sal   emp.sal%TYPE;
    v_deptno NUMBER NOT NULL := 10;
BEGIN
    SELECT sum (sal) --fonction
    INTO v_som_sal
    FROM emp
    WHERE deptno = v_deptno;
END;
```

Manipulation de données en PL/SQL

- Effectuer des mises à jour des tables de la base utilisant les commandes du LMD :
  - INSERT
  - UPDATE
  - DELETE

Insertion de données

- Ajouter les informations d'un nouvel employé dans la table emp

```
DECLARE
    v_empno NUMBER NOT NULL := 105 ;
BEGIN
    INSERT INTO emp(empno, emp_nom, poste, deptno)
    VALUES (v_empno,' Clémentine','Directrice',10)
END;
```


Mise à jour de données

- Augmenter le salaire de tous les employés dans la table emp qui ont le poste d'Enseignant.

```
DECLARE
    v_augm_sal  emp.sal%TYPE := 2000;
BEGIN
    UPDATE emp
    SET sal := sal + v_augm_sal
    WHERE job = 'Enseignant';
END;
```


Suppression de données

- Suppression des lignes appartenant au département 10 de la table emp

```
DECLARE
    v_deptno    emp.deptno%TYPE := 10;
BEGIN
    DELETE FROM emp
    WHERE deptno = v deptno ;
END;
```


Structures de contrôle dans PL/SQL

- IF conditionnel :

```
IF      THEN        END IF;

IF      THEN        ELSE        END IF ;

IF      THEN        ELSIF       END IF ;
```

- Les boucles :

```
LOOP    END     LOOP;

FOR     LOOP    END LOOP;

WHILE   LOOP    END LOOP;
```


Instruction IF

- Syntaxe :

```
IF condition THEN
    énoncés;
[ELSIF condition THEN
    énoncés;]
[ELSE
    énoncés;]
END IF ;
```


Instruction IF

- Exemple de IF simple :
- Mettre le ID de l'employé 'MARK' à 101.

```
IF v_nom = 'MARK' THEN
    v_ID := 101;
END IF ;
```


IF simple

- Si le nom de l'employé est 'Clémentine', lui attribuer le poste 'Enseignant', le département 102 et une commission de 25%

```
...
IF v_nom = 'Clémentine' THEN
    v_poste := 'Enseignant';
    v_deptno := 102;
    v_nouv_comm := salaire*0.25;
END IF ;
...
```

IF-THEN-ELSE

- Si le nom de l'employé est 'Clémentine', lui attribuer le poste 'Enseignant', le département 102 et une commission de 25%,
- Sinon message d'erreur

```
...
IF v_nom = 'Clémentine' THEN
    v_poste := 'Enseignant';
    v_deptno := 102;
    v_nouv_comm := salaire*0.25;
ELSE
    DBMS_OUTPUT.PUT_LINE('Employé inexistant') ;
END IF ;
...
```


IF-THEN-ELSIF

- Pour une valeur donnée en entrée, retourner une valeur calculée.

```
...
IF v_debut > 100 THEN RETURN (2*v_debut);
ELSIF v_debut >= 50 THEN RETURN (5*v_debut);
ELSE RETURN (1*v_debut);
END IF;
...
```


Boucle de base

- Syntaxe

```
LOOP --délimiteur
    énoncé 1 ; --énoncé
    ...
EXIT [WHEN condition]; --énoncé EXIT
END LOOP;
```

WHEN : condition est une variable booléenne ou expression (TRUE,FALSE,NULL).

Boucle de base

- Insérer 10 articles avec la date du jour

```
...
v_date DATE;
v_compteur NUMBER(2):=1;
BEGIN
...
    v_date := SYSDATE;
    LOOP
        INSERT INTO article (Artno, ADate )
        VALUES (v_compteur, v_date);
        v_compteur := v_compteur + 1;
        EXIT WHEN v_compteur > 10;
    END LOOP;
```


Boucle FOR

- Syntaxe

```
FOR indice IN [REVERSE]
    borne_inf ... borne_sup LOOP
    énoncé 1;
    énoncé 2;
    ...
END LOOP;
```

- Utiliser la boucle FOR pour raccourcir le test d'un nombre d'itérations
- Ne pas déclarer l'indice, il est déclaré implicitement.

Boucle FOR

- Insérer Nb articles de 1 a Nb (entré par l'utilisateur) avec la date du système en utilisant la boucle FOR

```
ACCEPT Nb PROMPT 'Donner le nombre d'"article:'
...
    v_date DATE;
BEGIN
    ...
    v_date := SYSDATE;
    FOR i IN 1 . . &Nb LOOP
        INSERT INTO article(Artno, ADate)
        VALUES (i ,v_date);
    END LOOP;
...
```


Boucle WHILE

- Syntaxe :

```
WHILE condition LOOP
    énoncé 1;
    énoncé 2 ;
    ...
END LOOP;
```

- La condition est évaluée en début de chaque itération
- Utiliser la boucle WHILE pour répéter des énoncés tant que la condition est vraie

Boucle WHILE

```
ACCEPT p_itemtot PROMPT 'Donner le total max d''article
DECLARE
    v_date DATE;
    v_compteur NUMBER(2):=1;
BEGIN
...
    v_date := SYSDATE;
    WHILE v_compteur <= &p_itemtot LOOP
        INSERT INTO article (Artno, ADate)
        VALUES (v_compteur, v_date);
        v_compteur := v_compteur + 1;
    END LOOP;
```

Boucles imbriquées et Labels

- Imbriquer les boucles à niveaux multiples
- Utiliser les `labels` pour distinguer les blocs et les boucles.
- Quitter la boucle extérieure avec un `EXIST` référençant le `label`.

Boucles imbriquées et Labels

```
BEGIN
    << boucle_ext>>
    LOOP
        v_compteur := v_compteur + 1;
        EXIT WHEN v_compteur > 10;
            <<boucle_int>>
            LOOP
                ...
                EXIT boucle_ext WHEN total_fait='OUI';
                --ici on quitte les deux boucles
                EXIT WHEN int_fait='OUI';
                --ici on quitte uniquement la boucle interne
                ...
            END LOOP boucle_int;
        ...
    END LOOP boucle_ext;
```


Types de données complexes

- Types :
  - `RECORDS`
  - `TABLES`
- Contiennent des composants internes
- Sont réutilisables

`RECORDS` PL/SQL

- Contiennent des champs qui sont soit :
  - des scalaires,
  - des records
  - ou des tables PL/SQL
- Structure similaire à des enregistrements dans les langages de programmation classiques
- Très utiles pour rechercher des lignes de données dans une table pour les traiter

Création d'un `RECORD` PL/SQL :

- Syntaxe

```
TYPE <Nom_Enreg> IS RECORD
    (Champ1 Type1
    .
    .
    ChampN TypeN );
```

```
TYPE TProd IS RECORD
    (
        V_Ref_Pro NUMBER(4),
        V_Des_Pro VARCHAR2(30),
        V_Pri_Uni NUMBER(7,2)
    );
```


Attribut %`ROWTYPE`

- Déclaration d'une variable associée à une collection de colonnes dans une table ou une vue de la BD
- Le nom de la table doit précéder `%ROWTYPE`
- Les champs dans le record prennent leurs noms et types des colonnes de la table ou la vue en question

Attribut %`ROWTYPE`

Exemples :

- Déclarer une variable `personne` pour stocker les mêmes informations concernant une personne telles qu'elles sont stockées dans la table `PERS` :
  - `personne PERS%ROWTYPE`
- Déclarer une variable `article` pour stocker les mêmes informations concernant un article telles qu'elles sont stockées dans la table `ART`
  - `article ART%ROWTYPE`

Tables `PL/SQL`

- Le type de données complexe `TABLE` offre au développeur un mécanisme pour traiter les tableaux
- Il se compose de deux colonnes :
  - Une clé primaire de type `BINARY_INTEGER`
  - Une colonne de type scalaire ou `RECORD`

Création d'une table PL/SQL

- Syntaxe

```
TYPE <Nom_table> IS TABLE OF <type_t>
INDEX BY BINARY INTEGER;
```

- Exemple

```
TYPE type_etud_nom IS TABLE OF etud.nom%TYPE
INDEX BY BINARY INTEGER;

etud_nom type_etud_nom;
```


Structure d'une table PL/SQL en mémoire

Clé primaire(BINARY_INTEGER)	Colonne (Scalaire)
1	Toto
2	Titi
3	Tata
..	..

Créer une table PL/SQL :

```
SQL> DECLARE
  2      TYPE    type_etud_nom   IS TABLE OF varchar2(10)
                                 INDEX BY BINARY_INTEGER;
  3      etud_nom       type_etud_nom;
  4  BEGIN
  5  select nom
  6  into etud_nom(1)
  7  from etud
  8  where etudid = 6;
  9  dbms_output.put_line(etud_nom(1));
 10  dbms_output.put_line(etud_nom(2));
 11  end;
 12  /
mark
DECLARE
*
ERREUR à la ligne 1 :
ORA-01403: Aucune donnée trouvée
```


TABLE de RECORDS en PL/SQL :

- Définit une table dont la deuxième colonne est un enregistrement au lieu d'un scalaire
- Pour définir la deuxième colonne :
  - Soit on utilise l'attribut %ROWTYPE
  - Soit en utilisant un RECORD déjà défini

TABLE de RECORDS PL/SQL

```
DECLARE
    TYPE type_etud_nom IS TABLE OF etud%rowtype
            INDEX BY BINARY INTEGER ;
    etud_nom    type_etud_nom;
BEGIN
    SELECT nom
    INTO etud_nom(1).nom
    FROM etud
    ...
END;
```


TABLE de RECORDS PL/SQL

```
DECLARE
    TYPE rec_etud IS RECORD(id etud.etudid%TYPE, nom etud.nom%TYPE);
    TYPE type_etud_nom IS TABLE OF rec_etud%ROWTYPE INDEX BY BINARY INTE
    etud_nom    type_etud_nom;
BEGIN
    SELECT nom
    INTO etud_nom(1).nom
    FROM etud
    ...
END;
```


Les curseurs dans SQL

- Un curseur est une zone de travail privée de SQL (zone tampon)
- Il y a deux types de curseurs :
  - Curseurs implicites
  - Curseurs explicites
- Oracle utilise les curseurs implicites pour analyser et exécuter les énoncés de SQL
- Les curseurs explicites sont déclarés explicitement par l'utilisateur du SGBD

Les attributs des curseurs SQL : ils permettent de tester les résultats des énoncés SQL

SQL%ROWCOUNT	Nombre de lignes affecté par l'énoncé SQL le plus récent (renvoie un entier).
SQL%FOUND	attribut booléen qui prend la valeur TRUE si l'énoncé SQL le plus récent affecte une ou plusieurs lignes.
SQL%NOTFOUND	attribut booléen qui prend la valeur TRUE si l'énoncé SQL le plus récent n'affecte aucune ligne.
SQL%ISOPEN	Prend toujours la valeur FALSE parce que PL/SQL ferme les curseurs implicites immédiatement après leur exécution.

Les attributs des curseurs SQL :

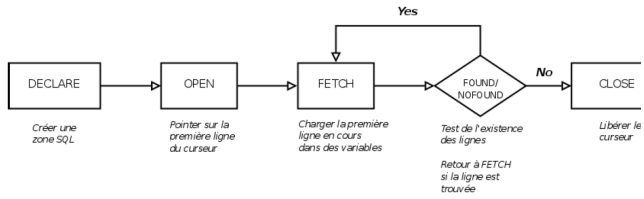
- Supprimer de la table ITEM des lignes ayant un ordre spécifié et afficher le nombre de lignes supprimées :

```
DECLARE
    v_ordid     NUMBER:=605;
BEGIN
    DELETE FROM item
    WHERE ordid = v_ordid;
    DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT || 'Lignes supprimées')
END;
```

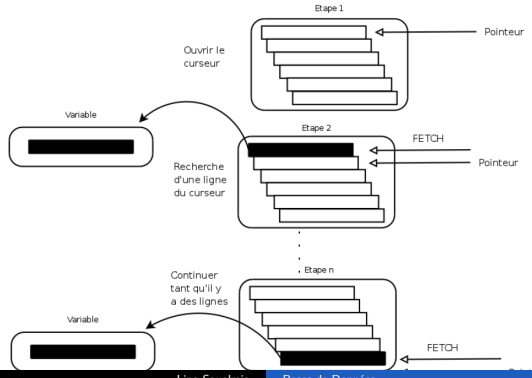

Les curseurs :

- Chaque énoncé SQL exécuté par Oracle a son propre curseur :
- Curseurs implicites :
  - déclarés pour tout énoncé SELECT du LMD ou PL/SQL
- Curseurs explicites :
  - déclarés et nommés par le programmeur

Contrôle des curseurs explicites :



Contrôle des curseurs explicites :



Déclaration des curseurs

- Syntaxe

```
CURSOR nom_du_curseur IS
un_énoncé SELECT;
```

- Ne pas inclure la clause INTO dans la déclaration du curseur
- Si le traitement des lignes doit être fait dans un ordre spécifié, on utilise la clause ORDER BY dans la requête

Déclaration des curseurs

- Exemple

```
DECLARE
    CURSOR C1 IS
    SELECT Ref_Art, Nom_Art, Qte_Art
    FROM Article
    WHERE Qte_Art<500;
    ...
```


Ouverture du curseur

- Syntaxe
  - `OPEN NomCurseur;`
- Ouvrir le curseur pour exécuter la requête et identifier l'ensemble actif
- Si la requête ne renvoie aucune ligne, aucune exception n'aura lieu
- Utiliser les attributs des curseurs pour tester le résultat du FETCH

Recherche des données dans le curseur

- Syntaxe :

```
FETCH nom_du_curseur
    INTO [variable1, [variable2...]| nom_de_record];
```

- Principe : rechercher les informations de la ligne en cours et les mettre dans des variables.

Recherche des données dans le curseur

```
FETCH C1 INTO v_Ref_Art, v_Nom_Art, v_Qte_Art;

OPEN Cur_Etud;
LOOP
    FETCH Cur_Etud INTO Rec_Etud ;
    {--traitement des données recherchées}
    ...
END LOOP;
```


Fermeture du curseur :

- Syntaxe :
  - `CLOSE NomCurseur;`
- Fermer le curseur après la fin du traitement des lignes
- Rouvrir le curseur si nécessaire
- On ne peut pas rechercher des informations dans un curseur si ce dernier est fermé

Les attributs du curseur explicite

- Obtenir les informations le l'état du curseur (CUR_EXP)

Attribut	Type	Description
CUR_EXP%ISOPEN	BOOLEAN	Prend la valeur TRUE si le curseur est ouvert
CUR_EXP%NOTFOUND	BOOLEAN	Prend la valeur TRUE si le FETCH le plus récent ne retourne aucune ligne
CUR_EXP%FOUND	BOOLEAN	Prend la valeur TRUE si le FETCH le plus récent retourne une ligne
CUR_EXP%ROWCOUNT	NUMBER	Retourne le nombre de lignes traitées jusqu'ici

Contrôle des recherches multiples :

- Traitement de plusieurs lignes d'un curseur en utilisant une boucle
- Rechercher une seule ligne à chaque itération
- Utiliser les attributs du curseur explicite pour tester le succès de chaque FETCH

L'attribut %ISOPEN

- La recherche des lignes n'est possible que si le curseur est ouvert
- Utiliser l'attribut %ISOPEN avant un FETCH pour tester si le curseur est ouvert ou non
- Exemple :

```
IF NOT C1%ISOPEN THEN
    OPEN C1
END IF;
LOOP
    FETCH C1 ...
```


Attributs %FOUND,%NOTFOUND et %ROWCOUNT :

- Utiliser l'attribut %ROWCOUNT pour fournir le nombre exact des lignes traitées
- Utiliser les attributs %FOUND et %NOTFOUND pour formuler le test d'arrêt de la boucle

Attributs %FOUND,%NOTFOUND et %ROWCOUNT :

```
LOOP
    FETCH curs1 INTO v_etudid, v_nom;
    IF curs1%ROWCOUNT > 20 THEN
        ...
    EXIT WHEN curs1%NOTFOUND;
    ...
END LOOP;
```


Obtenir les informations d'état du curseur

```
DECLARE
    nom     emp.ename%TYPE;
    salaire emp.sal%TYPE;
    CURSOR C1 IS SELECT ename, NVL(sal,0) FROM emp;
BEGIN
    OPEN C1 ;
    LOOP
        FETCH C1 INTO nom, salaire;
        EXIT WHEN C1%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE (nom ||'gagne'||salaire||'euros'
    END LOOP;
    CLOSE C1 ;
END;
```

Les curseurs et les RECORDS

- Traitement des lignes de l'ensemble actif par l'affectation des valeurs à des RECORDS PL/SQL.

```
...
    CURSOR Etud_Curs IS
        SELECT etudno, nom, age, adr
        FROM etud
        WHERE age<26;
    Etud_Record     Etud_Curs%ROWTYPE;
BEGIN
    OPEN Etud_Curs;
    ...
    FETCH Etud_Curs INTO Etud_Record;
    ...
```


Les boucles FOR des curseurs

- Syntaxe :

```
FOR nom_record IN nom_curseur LOOP
    --traitement des informations
    --utiliser des ordres SQL
    --utiliser des ordres PL/SQL
END LOOP;
...
```

- Un raccourci pour le traitement des curseurs explicites
- OPEN, FETCH et CLOSE se font de façon implicite
- Ne pas déclarer le record, il est déclaré implicitement

Les boucles FOR des curseurs

```
DECLARE
    CURSOR Cur_Etud IS SELECT * FROM Etud;
BEGIN
    FOR Rec_Etud IN Cur_Etud LOOP
        DBMS_OUTPUT.PUT_LINE(Rec_Etud.etudid||' '||Rec_Etud.nom ||' '||Rec_Etud
    END LOOP;
END;
/
```


Manipulation des exceptions en PL/SQL

- Le traitement des exceptions PL/SQL :
  - mécanisme pour manipuler les erreurs rencontrées lors de l'exécution
- Possibilité de continuer l'exécution :
  - si l'erreur n'est pas suffisamment importante pour produire la terminaison de la procédure
- Décision de continuer une procédure après erreur :
  - décision que le développeur doit faire en fonction des erreurs possibles

Types des exceptions

- Déclenchées implicitement
  - Exceptions Oracle prédéfinies
  - Exceptions Oracle non-prédéfinies
- Déclenchées explicitement
  - Exceptions définies par l'utilisateur

Capture des exceptions :

- Syntaxe

```
EXCEPTION
    WHEN exception_1 [OR exception_2...] THEN
        énoncé_1;
        énoncé_2;
    ...
    [WHEN exception_2 [OR exception_4...] THEN
        énoncé_3;
        énoncé_4;
    ...]
    [WHEN OTHERS THEN
        énoncé_5;
        énoncé_6;
    ...]
```


Capture des exceptions prédéfinies :

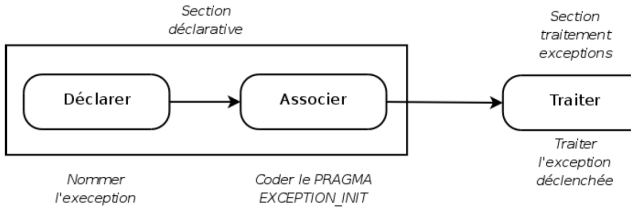
- Faire référence au nom dans la partie traitement des exceptions
- Quelques exceptions prédéfinies :
  - NO_DATA_FOUND
  - TOO_MANY_ROWS
  - INVALID_CURSOR
  - ZERO_DIVIDE
  - DUP_VAL_ON_INDEX

Exceptions prédéfinies

- Exemple

```
...
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        énoncé_1; énoncé_2;
        DBMS_OUTPUT.PUTLINE (TO_CHAR(etudno)||'Non valide');
    WHEN TOO_MANYROWS THEN
        énoncé_3; énoncé_4;
        DBMS_OUTPUT.PUT_LINE ('Données invalides');
    WHEN OTHERS THEN
        énoncé_5; énoncé_6;
        DBMS_OUTPUT.PUT_LINE('Autres erreurs');
```

Capture des exceptions définies par l'utilisateur

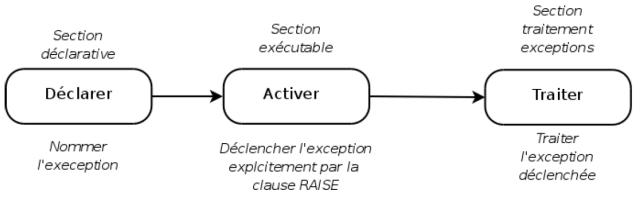


Capture des exceptions non-prédéfinies

- Exemple : Capture de l'erreur n°2291 (violation de la contrainte intégrité).

```
DECLARE
    cont_integrit_err EXCEPTION;
    PRAGMA EXCEPTION_INIT(cont_integrit_err, -2291);
    ...
BEGIN
    ...
    EXCEPTION
        WHEN contr_integrit_err THEN
        DBMS_OUTPUT.PUT_LINE ('violation de contrainte d''intégrité
        ...
END;
```


Capture des exceptions définies par l'utilisateur



Exceptions définies par l'utilisateur

```
DECLARE
    x   NUMBER := ...;
    x_trop_petit EXCEPTION;
    ...
BEGIN
    ...
    IF x < 5 THEN RAISE x_trop_petit;
    END IF ;
    ...
    EXCEPTION
        WHEN x_trop_petit THEN
        DBMS_OUTPUT.PUT_LINE('La valeur de x est trop petite!');
    ...
END;
```


Fonctions pour capturer les exceptions :

- SQLCODE :
  - Retourne la valeur numérique du code de l'erreur
- SQLERRM :
  - Retourne le message associé au numéro de l'erreur

Fonctions pour capturer les exceptions

```
...
v_code_erreur NUMBER;
v_message_erreur VARCHAR2(255);
BEGIN
    ...
    EXCEPTION
        ...
        WHEN OTHERS THEN
            ...
            v_code_erreur := SQLCODE;
            v_message_erreur := SQLERRM;
            INSERT INTO erreurs VALUES (v_code_erreur, v_message_erre
END;
```


Les sous-programmes

- Un sous programme est une séquence d'instructions PL/SQL qui possède un nom
- On distingue deux types de sous programmes :
  - Les procédures
  - Les fonctions

Les sous-programmes

- Une procédure :
  - sous-programme qui ne retourne des résultats que dans ses paramètres
- Une fonction :
  - sous-programme qui retourne des résultats dans :
    - le nom de la fonction
    - les paramètres de la fonction

Les procédures

- Syntaxe

```
DECLARE
    ...
    PROCEDURE <Nom_Proc>[(P1,...,Pn )] IS
        [Déclarations locales]
        BEGIN
            ...
        EXCEPTION
            ...
        END;
BEGIN
    /*Appel de la procédure*/
    ...
    EXCEPTION
    ...
END ;
```


Les procédures

- Syntaxe des paramètres : P1, ...Pn suivent la syntaxe

```
<Nom_Arg> [IN|OUT|IN OUT] <Type_Arg>
```

Où :

- IN : paramètre d'entrée
- OUT : paramètre de sortie
- IN OUT : paramètre d'entrée/sortie
- Par défaut le paramètre est IN

Les procédures

```
DECLARE
    PROCEDURE NouvSal (PNum IN Emp.EmpId%Type, PAug NUMBER )
    BEGIN
        SELECT Sal INTO VSal FROM Emp
        WHERE Emp_Id=PNum;
        UPDATE Emp SET Sal = VSal+PAug WHERE Emp_Id=PNum;
        COMMIT;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Employé inexistant');
    END;
BEGIN
    NouvSal (7550, 500);
    EXCEPTION
        WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE('Erreur');
END ;
```


Les procédures

```
DECLARE
    VErr NUMBER ;
    PROCEDURE NouvSal(PNum Emp.EmpId%TYPE, PAug NUMBER,PErr OUT NUMBER) IS VSal NUM
        BEGIN
            SELECT Sal INTO VSal FROM Emp WHERE EmpId=PNum;
            UPDATE Emp SET Sal = VSal+PAug WHERE EmpId=PNum;
            COMMIT;
            PErr :=0
        EXCEPTION
            WHEN NO_DATA_FOUND THEN PErr:=1;
        END;
```


Les procédures

```
BEGIN
    NouvSal (7550 ,500 ,VErr);
    IF VErr = 0 THEN
        DBMS_OUTPUT.PUT_LINE('Opération effectuée');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Employé inexistant');
    END IF;
EXCEPTION
    WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE('Erreur');
END;
/
```