Langage PL/SQL : Langage de programmation des BD

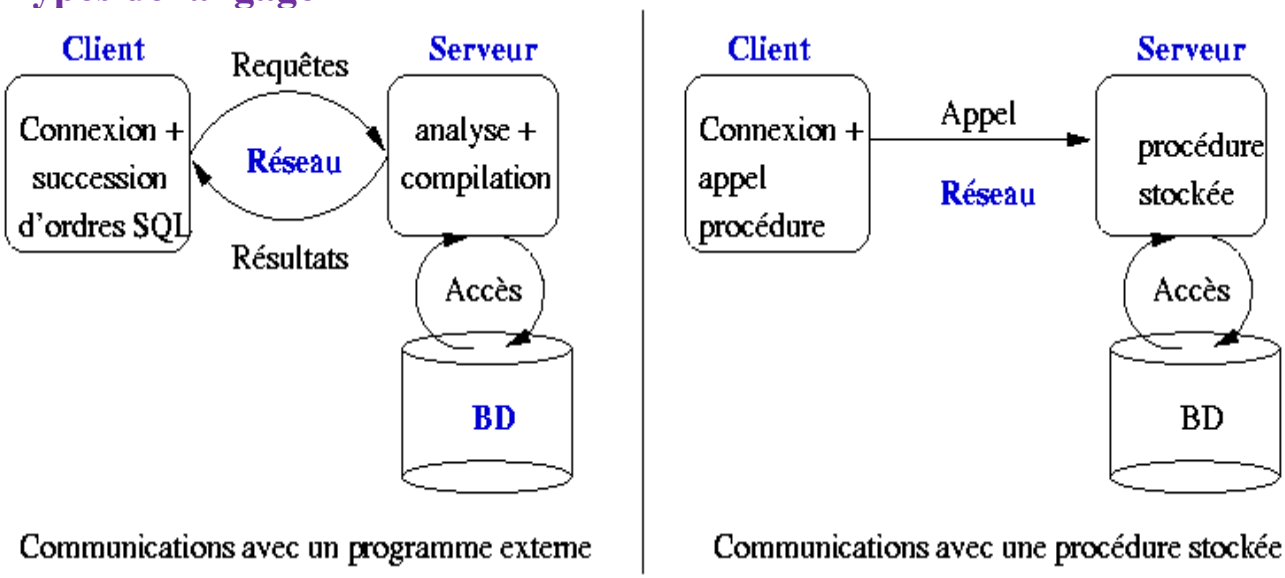
Requêtes SQL + Primitives de programmation :

En plus des requêtes SQL, effectuer des Itérations, Structures Conditionnelles, Créer des Fonctions, Variables, …).

Objectifs

Créer des procédures/fonctions stockées pour des traitements complexes sur la BD ;
Créer des Triggers pour le contrôle des mises de la BD

Types de langage



JDBC, JPA, … **PL/SQL, PDO (PHP)**

Le recours à une procédure stockée (fonctions/procédures stockées, Triggers dans PL/SQL) permet de regrouper du côté serveur l'ensemble des requêtes SQL et le traitement des données récupérées.

La procédure est compilée une fois par le SGBD, au moment de sa création, ce qui permet de l'exécuter rapidement au moment de l'appel.

De plus les échanges réseaux ne sont plus nécessaires puisque la logique de l'application est étroitement intégrée aux requêtes SQL.

Structure d'un bloc anonyme PLSQL_____

```
[DECLARE]
  [<liste_déclarations>]
 BEGIN
    <liste_instructions>
 [EXCEPTION]
    [<gestion_exceptions>]
 END ;
```

- La partie déclarative <liste_déclarations> n'est obligatoire que s'il y a des variables, constantes, curseurs, exceptions, types ou autres traitements (i.e. procédures ou fonctions) locaux au bloc.
- Le mot clef **DECLARE** ne doit être utilisé que pour introduire des déclarations dans les blocs anonymes.
- La partie <liste_instructions> est obligatoire et peut éventuellement être réduite à l'instruction **NULL ;** qui ne fait rien.
- La partie <gestion_exceptions> est optionnelle. Elle est utilisée pour traiter les anomalies détectées lors de l'exécution du bloc (Gestion des erreurs).

Les Ordres SQL valides en PL/SQL :

- Toute requête **SELECT** … **INTO** … **FROM** … dans la liste des instructions et **SELECT** … **FROM** … dans la partie déclarative pour définir un curseur.
- Les opérations de mise à jour des données : **INSERT**, **UPDATE**, **DELETE**.
- Les ordres de gestion de transaction : **COMMIT**, **ROLLBACK**

```
--Le plus petit Bloc anonyme PL/SQL
BEGIN
      NULL ;
END ;
```

```
--Bloc anonyme affichant BONJOUR
BEGIN
      DBMS_OUTPUT.PUT_LINE('BONJOUR') ;
END ;
```

```
--Bloc anonyme affichant BONJOUR  Nom Etudiant
ACCEPT Nom_Et   PROMPT 'Donnez le nom d''un étudiant'
BEGIN
      DBMS_OUTPUT.PUT_LINE('BONJOUR' || ' ' || '&Nom_Et' ) ;
END ;
```

Partie déclarative d'un bloc PL/SQL

Variables scalaires

Types : un des types prédéfinis d'ORACLE, i.e. ceux utilisés pour définir les attributs (e.g. **VARCHAR2**(n), **NUMBER**(x,y), **NUMBER**(x), **DATE** …)
ou le type **BOOLEAN** (**TRUE**, **FALSE** et … **NULL**).

<nom_variable> <nom_type> [**NOT NULL**] [**DEFAULT** <valeur>] ;
ou <nom_variable> <nom_type> [**NOT NULL**] [:=<valeur>] ;

Nom_Et VARCHAR2(25) **DEFAULT** 'RUBINI' ;
Nom_Et VARCHAR2(25) ;

<nom_variable> <nom_reference>**%TYPE** [**NOT NULL**] [**DEFAULT** <valeur>];
ou <nom_variable> <nom_reference>**%TYPE** [**NOT NULL**] [:=<valeur>] ;

Nom_Et ETUDIANT.Nom_Et%TYPE **DEFAULT** 'RUBINI' ;
Nom_Et ETUDIANT.Nom_Et%Type ;

<nom_reference> correspond au nom d'une autre variable ou d'un attribut de la base préfixé par le nom de la relation à laquelle il appartient : <nom_relation>.<nom_attribut>.

Constantes

<nom_constante> **CONSTANT** <nom_type> := valeur ;

SALAIRE NUMBER(6, 2) := 5000 ;

Variables composées (structure d'un tuple)

<nom_variable_struc> <nom_table|nom_curseur>.**%ROWTYPE** ;

Un_Etudiant ETUDIANT%ROWTYPE ;

Accès à la valeur d'une composante : <nom_variable.struc>.Attribut

Un_Etudiant.Nom_Et

Instructions PL/SQL

Affectation

<nom_variable_scalaire> := <expression> ;
<nom_variable_struc>.<nom_attribut> := <expression> ;

Affectation par requête

```
SELECT <nom_attribut | liste_attributs | *>
INTO <variable_receptrice>
FROM ... WHERE ...
```

Affectation par FETCH

```
FETCH <nom_curseur>  INTO  <variable-receptrice>  ;
```

Instruction Conditionnelle

IF <condition1>
THEN [**BEGIN**] <liste1_instructions> [**END** ;]
[**ELSIF** <condition2>
THEN [**BEGIN**] <liste2_instructions> [**END** ;]]
[**ELSE** [**BEGIN**] <liste3_instructions> [**END** ;]]
END IF ;

Itérations

LOOP
    <liste_instructions>
END LOOP;

FOR <compteur> **IN** [**REVERSE**] <borne_inf> .. <borne_sup>
LOOP
    <liste_instructions>
END LOOP ;

WHILE <condition>
LOOP
    <liste_instructions>
END LOOP ;

Les curseurs

Définition de curseurs

```
CURSOR <nom_curseur>  IS  <requete_SQL> ;

CURSOR     <nom_curseur>     (<parametre1>     <type1>[,
<parametre2> <type2> …])  IS  <requete_SQL> ;
```

Propriété des curseurs :

```
<nom_curseur>%FOUND
<nom_curseur>%NOTFOUND
<nom_curseur>%ISOPEN
<nom_curseur>%ROWCOUNT
```

Gestion de curseurs

Manuelle :

- **OPEN** <nom_curseur> ;
  OPEN <nom_curseur>(<valeur1>[, <valeur2> …]) ;
- **CLOSE** <nom_curseur> ;
- **FETCH** <nom_curseur> **INTO** <variable-receptrice> ;

```
--Afficher la moyenne des notes de test par matière.
DECLARE
      CURSOR MoyMat IS
      SELECT Code, AVG(Moy_Test) Moy
      FROM NOTATION
      GROUP BY Code;

      Une_Ligne MoyMat%ROWTYPE ;
BEGIN
      OPEN MoyMat ;
      FETCH MoyMat INTO Une_Ligne ;
      WHILE MoyMat%FOUND
      LOOP
            DBMS_OUTPUT.PUT_LINE(Une_Ligne.Code    ||    ‘    ‘    ||
            Trunc(Une_Ligne.Moy, 2)) ;
            FETCH MoyMat INTO Une_Ligne ;
      END LOOP ;
      CLOSE MoyMat ;
END ;
```

Automatique :

```
FOR <nom_variable_parcours> IN <nom_curseur>
LOOP
        <liste_instructions>
END   LOOP ;
```

```
--Afficher la moyenne des notes de test par matière.
DECLARE
      CURSOR MoyMat IS
      SELECT Code, AVG(Moy_Test) Moy
      FROM NOTATION
      GROUP BY Code ;

BEGIN
      FOR Une_Ligne IN MoyMat
      LOOP
            DBMS_OUTPUT.PUT_LINE(Une_Ligne.Code    ||    '    '    ||
            Trunc(Une_Ligne.Moy, 2)) ;
      END LOOP ;
END ;
```

Automatique avec cursor non déclaré :

```
FOR <nom_variable_parcours> IN <(requete SQL)>
LOOP
        <liste_instructions>
END   LOOP ;
```

```
--Afficher la moyenne des notes de test par matière.
BEGIN
      FOR Une_Ligne IN  (SELECT Code, AVG(Moy_Test) Moy
                              FROM NOTATION
                              GROUP BY Code)
      LOOP
            DBMS_OUTPUT.PUT_LINE(Une_Ligne.Code    ||    '    '    ||
            Trunc(Une_Ligne.Moy, 2)) ;
      END LOOP ;
END ;
```

Curseur paramétré :

```
FOR  <variable_parcours>  IN  <nom_curseur>(<valeur1>[,
<valeur2> …]) …
LOOP
        <liste_instructions>
END   LOOP ;
```

```
--Afficher la moyenne des notes de test par matière en
--utilisant un curseur paramétré (sans Group BY).
DECLARE
      CURSOR MoyMat(Mat NOTATION.Code%Type) IS
      SELECT AVG(Moy_Test) Moy
      FROM NOTATION
      WHERE Code = Mat ;

BEGIN
      FOR Une_Mat IN (SELECT DISTINCT Code FROM NOTATION)
      LOOP
            FOR Une_Ligne IN MoyMat(Une_Mat.Code)
            LOOP
                  DBMS_OUTPUT.PUT_LINE(Une_Mat.Code   ||  ‘  ‘  ||
                  Trunc(Une_Ligne.Moy, 2)) ;
            END LOOP ;
      END LOOP ;
END ;
```

Exceptions

Déclaration d'exceptions

nom_exception **EXCEPTION** ;

Déclenchement d'exceptions utilisateur nommées

```
IF  <condition>  THEN  RAISE <nom_exception> ;
END IF  ;
```

Principales Exceptions système

NO_DATA_FOUND ; TOO_MANY_ROWS ; ZERO_DIVIDE, OTHERS.

Traitements des exceptions

WHEN <nom_exception> **THEN** [**BEGIN**] <liste_instructions> [**END**] ;

```
--Exception système NO_DATA_FOUND
DECLARE
        Nom  ETUDIANT.Nom_Et%Type ;
        Prenom      ETUDIANT.Prenom_Et%Type ;
BEGIN
        SELECT Nom_Et, Prenom_Et INTO  Nom, Prenom
        FROM ETUDIANT
        WHERE Num_ET = 22 ;
        DBMS_OUTPUT.PUT_LINE(Nom || ' ' ||  Prenom) ;
EXCEPTION
        WHEN NO_DATA_FOUND
        THEN DBMS_OUTPUT.PUT_LINE('Numéro étudiant erroné') ;
END ;

--Exception utilisateur
DECLARE
        MoyenneBD       NOTATION.Moy_Test%Type ;
        Pas_Note    EXCEPTION ;
BEGIN
        SELECT AVG(Moy_Test) INTO  MoyenneBD
        FROM NOTATION WHERE Code = 'ABD' ;
        IF MoyenneBD IS NULL THEN RAISE Pas_Note ;
        END IF ;
        DBMS_OUTPUT.PUT_LINE(MoyenneBD) ;
EXCEPTION
        WHEN Pas_Note
        THEN DBMS_OUTPUT.PUT_LINE('Pas de notes BD') ;
END ;
```

Procédures, Fonctions

Spécifications de procédures

PROCEDURE <nom_procedure>[(<specification_parametre>)]
IS
    [<liste_declarations>]
BEGIN
    <liste_instructions>
[**EXCEPTION**
    <gestion_exception>]
END [<nom_procedure>] ;

<specification_parametre> :=
<nom_parametre> <mode> <nom_type> [:= <valeur_defaut>]

<mode> peut prendre les valeurs `IN`, `OUT` ou `IN OUT`.

Procédures stockées

CREATE [OR REPLACE] PROCEDURE <nom_procedure> <specification_procedure> ;

Spécifications de fonctions

FUNCTION <nom_fonction>[(<specification_parametre>)]
RETURN <type_donnee>
IS
    [<liste_declarations>]
BEGIN
    <liste_instructions>
[**EXCEPTION**
    <gestion_exception>]
END [<nom_fonction>] ;

<type_donnee> est un des types scalaires ou composés valides en PL/SQL et ne peut être ni un curseur ni une exception.

Fonctions stockées

```
CREATE [OR REPLACE] FUNCTION <nom_fonction>
<specification_fonction> ;
```

--Calcul de l'âge à partir de la date de naissance

```
CREATE OR REPLACE FUNCTION calcul_age(datenais DATE)
RETURN NUMBER
IS
age NUMBER(3,0);
nb_mois CONSTANT NUMBER(2,0) := 12;

BEGIN
age := floor(months_between(sysdate, datenais)/nb_mois);
RETURN(age);
END;
```

Appels de la fonction calcul_age :

Dans SQL :

```
SELECT calcul_age('12/10/2003')
FROM DUAL ;
```

Dans un Bloc Anonyme :

```
DECLARE
     Date_Naissance Date := '12/10/2003' ;
     Age NUMBER(3) ;
BEGIN
     Age := calcul_age(Date_Naissance') ;
     DBMS_OUTPUT.PUT_LINE(Age) ;
END ;
```