

Appels de sous-programmes PL/SQL en JDBC

Le Langage de BLOC

PL/SQL

Le Langage de Bloc PL/SQL

- **PL/SQL:**
 - ‘Procédural Language’ : sur-couche procédurale à SQL, **boucles, contrôles, affectations, exceptions**,
 - Chaque **programme est un bloc (BEGIN – END)**
 - Programmation adaptée pour :
 - Transactions
 - Une **architecture Client - Serveur**

Bloc PL/SQL

- Le **bloc de requêtes** est envoyé **sur le serveur**: celui-ci exécute tout le bloc et renvoie un résultat final.



Format d'un bloc PL/SQL

- **Section DECLARE** : déclaration de
 - Variables locales simples
 - Variables tableaux
 - ...
- **Section BEGIN**
 - Section des ordres exécutables
 - Ordres SQL
 - Ordres PL
- **Section EXCEPTION**
 - Réception en cas d'erreur
 - Exceptions SQL ou utilisateur

```
DECLARE  
    --déclarations  
  
BEGIN  
    --exécutions  
  
EXCEPTION  
    --erreurs  
  
END;  
/
```

Structure d' un bloc PL/SQL

DECLARE *optionnelle*

Variables, curseurs, exceptions, ...

BEGIN *obligatoire*

Instructions SQL et PL/SQL

Possibilités de blocs fils (imbrication de blocs)

EXCEPTION *optionnelle*

Traitements des exceptions (gestion des erreurs)

END ; *obligatoire*

/

Les types de blocs

Anonyme

```
[ DECLARE ]
```

```
BEGIN
```

```
Instructions
```

```
[ EXCEPTION]
```

```
END;
```

```
/
```

Procédure

```
PROCEDURE nom  
IS
```

```
BEGIN
```

```
Instructions
```

```
[ EXCEPTION]
```

```
END;
```

```
/
```

Fonction

```
FUNCTION nom  
RETURN type_données  
IS
```

```
BEGIN
```

```
Instructions
```

```
RETURN valeur;
```

```
[ EXCEPTION]
```

```
END;
```

```
/
```

Exemple d'un bloc PL/SQL

```
DECLARE
    var_x      VARCHAR2(5);
BEGIN
    SELECT nom_colonne
    INTO var_x
    FROM nom_table
EXCEPTION
    WHEN nom_exception THEN
        .....
END ;
/
```

Déclaration des variables en PL/SQL

Syntaxe:

Identificateur [CONSTANT] type_données [NOT NULL] [:= expression];

- Variables de type SQL

```
nbr      NUMBER(2) ;
nom      VARCHAR(30) ;
minimum  CONSTANT INTEGER := 5 ;
salaire NUMBER(8,2) ;
debut    NUMBER NOT NULL ;
```

- Variables de type booléen (TRUE, FALSE, NULL)

```
fin      BOOLEAN ;
reponse BOOLEAN DEFAULT TRUE ;
ok      BOOLEAN := TRUE ;
```

La directive %TYPE:

- référence à un type existant qui est soit une colonne d'une table soit un type défini précédemment
- **%TYPE** : se lit de même type que

Syntaxe:

nom_variable1 nom_variable2%TYPE ;

Exemples:

- `vsalaire` est une variable de même type que la colonne **salaire** de la table **employe** :
`vsalaire employe.salaire%TYPE;`
- `vemploye` est une variable de même type qu'une ligne de la table **employe** :
`vemploye employe%ROWTYPE;`
- `vcomm` est une variable de même type que la variable **vsalaire**:
`vcomm vsalair%TYPE;`

Exemples de Variables faisant référence au dictionnaire de données

- Référence à une colonne de table (ou vue)

vslaire	employe.salaire%TYPE;
vnom	etudiant.nom%TYPE;
vcomm	vslaire%TYPE;

- Référence à une ligne de table (ou vue)

vemploye	employe%ROWTYPE;
vetudiant	etudiant%ROWTYPE;

- Contenu d' une variable : **variable.colonne**

vemploye.adresse

L' instruction SELECT dans PL/SQL

Récupérer une donnée de la BD avec SELECT.

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

L' 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;
```

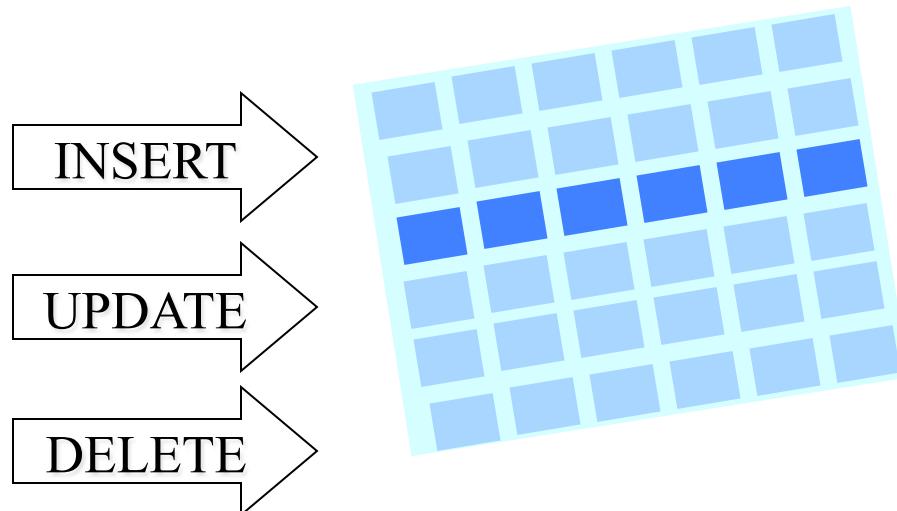
L' instruction SELECT dans PL/SQL

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 d' agrégat
    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 BD
utilisant les commandes du LMD:



Insertion de données

Ajouter les informations d'un nouvel employé à la table *emp*

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

Mise à jour de données

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

```
DECLARE
    v_augm_sal    emp.sal%TYPE := 2000;
BEGIN
    UPDATE      emp
    SET      sal := sal + v_augm_sal
    WHERE      poste= 'analyste';
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;
```

Les sous-Programmes

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

Les sous-Programmes

- ***Une procédure*** : est un sous programme qui ne retourne des résultats seulement dans ses paramètres.
- ***Une fonction*** : est un sous programme qui retourne des résultats dans :
 - Le nom de la fonction
 - Les paramètres de la fonction .

Les procédures

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

Les procédures

P1,...,Pn suivent la syntaxe :

<Nom_Arg> [IN | OUT | IN OUT] <Type> Où :

IN : Paramètre d' entrée

OUT : Paramètre de sortie

IN OUT : Paramètre d' entrée/Sortie

Par défaut le paramètre est IN

Les fonctions

```
DECLARE
    [Déclarations globales]
FUNCTION <Nom_fonc>[(P1,...,Pn)] RETURN Type IS
    [Déclarations locales]
    BEGIN
        ...
        RETURN valeur;
    EXCEPTION
        ...
    END ;
BEGIN
    -- Appel a la fonction
    ...
EXCEPTION
    ....
END ;
/
```

Les procédures et les fonctions stockées

- Sont des blocs PL/SQL qui possèdent des noms.
- Consistent à ranger le bloc PL/SQL compilé dans la base de données (CREATE).
- Peuvent être appelées de n'importe quel bloc PL/SQL.

Les procédures stockées

```
CREATE [ OR REPLACE ] PROCEDURE <Nom_Proc>[(P1,...,Pn)] IS  
[Déclarations des variables locales]  
BEGIN  
...  
EXCEPTION  
...  
END ;  
/
```

→ Procedure Created → *La procédure est correcte*

Ou

→ Procedure Created with compilation errors → *Corriger les erreurs* →
SHOW ERRORS;

Les procédures stockées

```
CREATE [ OR REPLACE ] PROCEDURE <Nom_Proc>[(P1,...,Pn)] IS  
[Déclarations des variables locales]  
BEGIN  
...  
EXCEPTION  
...  
END ;  
/
```

```
CREATE PROCEDURE AugmenteCapacité(p_immat IN VARCHAR,  
p_n IN NUMBER) IS  
BEGIN  
    UPDATE Avion SET cap = cap + p_n WHERE immat = p_immat;  
END;
```

Les fonctions stockées

```
CREATE [ OR REPLACE ] FUNCTION <Nom_Fonc>[(P1,...,Pn)]  
  RETURN Type IS  
    [Déclarations des variables locales]  
  BEGIN  
    Instructions SQL et PL/Sql  
    RETURN(Valeur)  
  EXCEPTION  
    Traitement des exceptions  
  END ;  
 /
```

→ function Created → *La fonction est correcte*

Ou

→ function Created with compilation errors → *Corriger les erreurs* →
SHOW ERRORS;

Exemple de Fonction PL/SQL

```
CREATE FUNCTION LeNomCompagnieEst(p_immat IN VARCHAR) RETURN VARCHAR IS
    résultat Compagnie.nomComp%TYPE;
BEGIN
    SELECT nomComp INTO résultat
    FROM Compagnie WHERE comp = (SELECT comp FROM Avion WHERE immat = p_immat);
    RETURN résultat;
EXCEPTION
    WHEN NO_DATA_FOUND THEN RETURN NULL;
END;
```

Appel d'un sous-programme PL/SQL en JDBC

L'interface ***CallableStatement*** permet d'appeler des sous-programmes (fonctions ou procédures PLSQL...), en passant d'éventuels paramètres en entrée et en récupérant en sortie.

L'interface `callableStatement` spécialise l'interface ***PreparedStatement***.

Les **paramètres d'entrée** sont affectés par les méthodes `setxxx()`.

Les **paramètres de sortie** (définis ***OUT*** au niveau du sous-programme) sont extraits à l'aide des méthodes `getxxx()`.

Ces états qui permettent d'appeler des sous-programmes sont créés par la méthode `prepareCall` de l'interface `Connection`: ***CallableStatement prepareCall (String)***

Type du sous-programme	Paramètre
Fonction	<code>{? = call nomFonction(?, ?, ...) }</code>
Procédure	<code>(call nomProcédure(?, ?, ...))</code>

Appel d'une procédure PL/SQL

```
CREATE PROCEDURE AugmenteCapacité(p_immat IN VARCHAR,  
p_n IN NUMBER) IS  
BEGIN  
    UPDATE Avion SET cap = cap + p_n WHERE immat = p_immat;  
END;
```

Code Java

```
try { ...  
    String ordreSQL = "{call AugmenteCapacité(?, ?)}";  
    CallableStatement étatAppelable =  
        cx.prepareCall(ordreSQL);  
  
    étatAppelable.setString(1, "F-GLFS");  
    étatAppelable.setInt(2, 50);  
  
    étatAppelable.execute();  
  
    étatAppelable.close();  
}  
} catch(SQLException ex) { ... }
```

Commentaires

Création d'un état appelable.

Passage des paramètres d'entrée.

Exécution de la procédure.

Fermeture de l'état.

Gestion des erreurs.

Appel d'une fonction PL/SQL

Code Java

```
try { ...  
    String ordreSQL =  
        " (? = call LeNomCompagnieEst (?) ) ";  
    CallableStatement étatAppelable =  
        cx.prepareCall(ordreSQL);
```

Commentaires

Création d'un état appelable.

```
étatAppelable.registerOutParameter  
    (1, java.sql.Types.VARCHAR);
```

Déclaration du paramètre de sortie.

```
étatAppelable.setString(2, "F-GLFS");
```

Passage du paramètre d'entrée.

```
étatAppelable.execute();
```

Exécution de la fonction.

```
System.out.print ("Compagnie de F-GLFS : "+  
    étatAppelable.getString(1));
```

Extraction du résultat.

```
étatAppelable.close();
```

Fermeture de l'état.

```
} catch (SQLException ex) { ... }
```

Gestion des erreurs.

Transactions et Points de validation

Code Java

```
try { ...  
    cx.setAutoCommit(false);  
    String ordreSQL =  
        "INSERT INTO Avion VALUES (?, ?, ?, ?, ?)";  
    PreparedStatement étatPréparé =  
        cx.prepareStatement(ordreSQL);  
  
    Savepoint p1 = cx.setSavepoint("P1");  
    étatPréparé.setString(1, "F-NEW2");  
    ...  
    if (! étatPréparé.execute())  
        System.out.println("F-NEW2 inséré");  
  
    Savepoint p2 = cx.setSavepoint("P2");  
    étatPréparé.setString(1, "F-NEW3");  
    ...  
    if (! étatPréparé.execute())  
        System.out.println("F-NEW3 inséré");  
  
    cx.rollback(p2);  
    cx.commit();  
    cx.close();  
} catch (SQLException ex) { ... }
```

Commentaires

Désactivation de l'*autocommit*.
Création d'un état appelable.

Création du point de validation P1.

Passage de paramètres et première
insertion.

Création du point de validation P2.

Passage de paramètres et
deuxième insertion.

Annulation de la deuxième partie.

Validation de la première partie.

Fermeture de la connexion.

Gestion des erreurs.