

Complément 3

Procédures stockées et externes

Oracle propose deux mécanismes distincts mais complémentaires afin d'intégrer des programmes écrits en langages de troisième et quatrième générations (C, C++, Java, etc.) :

- Les procédures stockées (*stored procedures*) au niveau de la base de données.
- Les procédures externes (*external procedures*) à la base de données qui sont appelables.

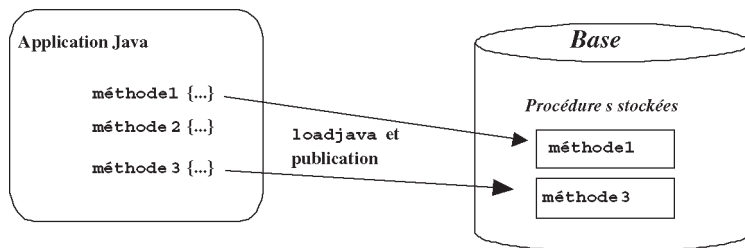
Ce chapitre présente ces deux mécanismes en utilisant le langage Java. Il est aisé de transposer les principes que nous présentons dans ce chapitre à d'autres langages évolués.

Procédures stockées Java

Une procédure stockée est une méthode d'une classe Java qui est compilée au niveau de la base de données. La machine virtuelle Java qui est présente dans le noyau exécute le fichier compilé (*byte-code*). Une procédure stockée peut accéder aux différents autres objets de la base (tables, vues, déclencheurs, etc.) par l'intermédiaire de la passerelle JDBC.

Les avantages à utiliser les procédures stockées sont multiples : le programmeur a la possibilité de récupérer son code qui peut avoir été développé il y a longtemps. D'autre part, l'optimisation et l'intégration de l'appel à ce code sont assurées par la base elle-même. La figure suivante illustre deux méthodes d'une classe Java qu'on transforme en procédure stockée de la base :

Figure C3-1 Procédures stockées Java





Seules les méthodes utilisant des directives des paquetages graphiques (*AWT* et *Swing*) ne peuvent pas être stockées dans la base.

Les trois contextes d'utilisation d'une procédure stockée sont les suivants :

- développement de fonctions ou procédures en relation avec les données de la base ;
- programmation de déclencheurs ;
- développement de méthodes associées à un type objet relationnel (extension objet du modèle relationnel d'Oracle non traitée ici).

Stockage d'une procédure

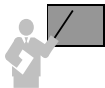
Différentes étapes doivent être respectées afin d'intégrer une procédure stockée Java.

Développement de la classe

Il faut bien sûr disposer d'une classe Java qu'on aura éventuellement compilée auparavant afin de s'assurer de sa syntaxe. Le code suivant décrit la classe `PremierExemple` contenant la méthode `affiche` qui compte le nombre de lettres contenues dans la chaîne passée en paramètre.

```
public class PremierExemple {
    public static String affiche(String message)
    { int nc=0;
      for (int i=0;i<message.length();i++)
          if ((message.charAt(i)>='a')&&(message.charAt(i)<='z'))
              || ((message.charAt(i)>='A')&&(message.charAt(i)<='Z'))
              nc++;
      return "Le message contient "+nc+" lettres!";}}

```



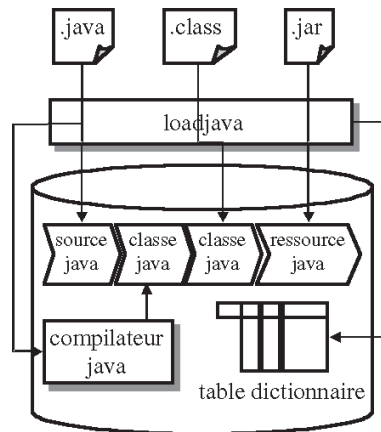
Seules les méthodes déclarées `public static` peuvent devenir des procédures stockées.

Chargement d'une méthode

La seconde étape consiste à faire appel à l'utilitaire `loadjava` qui permet de charger des ressources Java dans la base. Comme le montre la figure suivante, différents types de ressources Java peuvent être chargés :

- les fichiers source (.java) ;
- les classes (.class) ;
- les archives (.jar).

Figure C3-2 Chargement des méthodes



La commande suivante réalise l'opération de chargement. L'instruction est passée en ligne de commande dans le répertoire contenant la source ; l'utilisateur, le mot de passe et la chaîne de connexion désignent le schéma cible. L'option `-verbose` permet de tracer les différentes actions effectuées. Le chargeur effectue la connexion à la base de données, compile le code source, enregistre la source et la classe compilée.

```
loadjava -user soutou/ingres@cxbdsoutou -verbose PremierExemple.java
arguments: '-user' 'soutou/ingres@cxbdsoutou' '-verbose' 'PremierExemple.java'
created   : JAVA$CLASS$MD5$TABLE
creating  : source PremierExemple
created   : CREATE$JAVA$LOB$TABLE
loading   : source PremierExemple
creating  : PremierExemple
```

Vue du dictionnaire

La vue `USER_OBJECTS` permet d'extraire les différents objets Java entreposés dans la base (les types possibles sont : `JAVA CODE`, `JAVA CLASS` et `JAVA RESOURCE`). Nous retrouvons ainsi le nom de la classe chargée de notre exemple.

```
SQL> SELECT DBMS_JAVA.LONGNAME(OBJECT_NAME) FROM USER_OBJECTS
       WHERE OBJECT_TYPE = 'JAVA CLASS';

DBMS_JAVA.LONGNAME(OBJECT_NAME)
-----
PremierExemple
```

Publication de la méthode

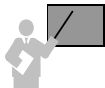
Cette étape se réalise sous l’interface de commande SQL*Plus. Chaque méthode Java qui est appelée doit être publiée. La publication attribue une signature PL/SQL à la méthode. La syntaxe SQL à utiliser pour publier une méthode Java est la suivante :

```
CREATE [OR REPLACE]
{ FUNCTION nomFonction [(paramètre1 [, paramètre2]...)] RETURN
typeSQL }
| PROCEDURE nomProcédure [(paramètre1 [, paramètre2]...)] }
{IS | AS} LANGUAGE JAVA
NAME 'nomClasse.nomMéthode([(paramètre1 [, paramètre2]...)] [return
typeJava]');
```

Les paramètres SQL sont définis par un nom, un mode et un type SQL (*nomparamètre* [IN | OUT | IN OUT] *typeSQL*). L’instruction suivante publie la méthode affiche de la classe PremierExemple renommée au niveau de la base PremierExemple_affiche :

Tableau C3-1 Publication d’une méthode

Instruction SQL	Commentaires
CREATE FUNCTION PremierExemple_affiche (mess VARCHAR2) RETURN VARCHAR2	Signature de la procédure stockée et type de retour.
AS LANGUAGE JAVA	Langage utilisé.
NAME 'PremierExemple.affiche'(java.lang.String) return java.lang.String'; /	Identification de la méthode Java.



- Le nom de la méthode publiée et les noms de ses paramètres ne sont pas nécessairement identiques à la méthode source Java.
- Il faut respecter la correspondance des types Java et SQL (String et VARCHAR2 par exemple).
- Java distingue les majuscules des minuscules, donc la casse doit être respectée (noms de classes, méthodes et paramètres Java). Par exemple, le mot return doit être en minuscules).

Appel de la méthode

L’appel d’une procédure stockée peut être réalisé :

- sous l’interface de commande SQL*Plus (*top level*) ;
- à partir d’une commande SQL (SELECT, INSERT, UPDATE ou DELETE) ;

- à partir d'un programme PL/SQL (bloc, fonction ou procédure) ;
- à partir d'un déclencheur.

Sous SQL*Plus

Pour appeler la méthode publiée à partir de SQL*Plus, il faut utiliser la commande CALL. Dans le cas d'une fonction, il convient d'utiliser une variable globale de retour qu'il faudra définir auparavant (voir le chapitre 7).

```
CALL [nomPaquetage.] {nomProcédure([paramètre1[, paramètre2]...])
                        | nomFonction([paramètre1[,paramètre2]...]) INTO
                        :variable};
```

Le code suivant teste notre exemple :

Tableau C3-2 Appel d'une procédure stockée Java

Instruction SQL	Commentaires
VARIABLE g_résultat VARCHAR2(50);	Déclaration de la variable de retour.
CALL PremierExemple_affiche('SQL, un langage évolué???') INTO :g_résultat;	Appel de la méthode publiée.
PRINT g_résultat;	Affichage du résultat.
G_RÉSULTAT ----- Le message contient 16 lettres!	

Dans une commande SQL

La requête suivante invoque la procédure stockée. Nous utilisons la pseudo-table DUAL mais il est possible d'appeler une procédure stockée en passant en paramètres des valeurs de colonnes d'une table contenant des enregistrements.

```
SQL> SELECT PremierExemple_affiche('SQL, un langage évolué???')
FROM DUAL;

PREMIEREXEMPLE_AFFICHE('SQL,UNLANGUAGEÉVOLUÉ???')
-----
Le message contient 16 lettres!
```

Il est aussi possible d'appeler une procédure stockée dans une insertion, une modification ou une suppression.

Sous PL/SQL

L'appel de la même méthode sous PL/SQL se réalise classiquement comme s'il s'agissait d'une fonction cataloguée, elle-même écrite en PL/SQL.

```
DECLARE
    v_résultat VARCHAR2(50);
BEGIN
    v_résultat :=
    PremierExemple_affiche('SQL, un langage évolué???) ;
END;
```

Dans un déclencheur

Le déclencheur suivant invoque une procédure stockée que nous décrirons dans le paragraphe « Déclencheurs » de cette section. Dans cet exemple, chaque suppression d'une compagnie déclenchera l'appel de la méthode Java correspondante en passant en paramètre le nom de la compagnie supprimée.

```
CREATE TRIGGER Ex_trig_Java
    AFTER DELETE ON Compagnie FOR EACH ROW
BEGIN
    DeuxièmeExemple_affiche(:OLD.nomcomp) ;
END;
```

Partage de la méthode

Une fois que la méthode a été publiée, elle devient un objet à part entière du schéma auquel on peut attribuer des prérogatives d'exécution (GRANT EXECUTE ON PremierExemple_affiche TO Paul).

Suppression de la méthode

Pour supprimer une méthode au niveau de la base, Oracle propose la procédure dropjava (issue du paquetage DBMS_JAVA) qu'on appelle soit en ligne de commande, soit sous PL/SQL ou SQL*Plus. Ainsi, le script suivant permet de supprimer les deux premiers exemples en utilisant deux écritures différentes (la deuxième écriture nécessite d'être connectée dans le schéma qui contient la procédure stockée).

```
dropjava -user soutou/ingres@cxbd@soutou PremierExemple.java
CALL DBMS_JAVA.DROPJAVA('DeuxièmeExemple.java');
```

Il est possible de supprimer de la même manière des classes (.class) ou des ressources (.jar ou .zip).

Interactions avec la base

Le mécanisme de communication entre la procédure stockée Java et la base de données est fondé sur la technologie JDBC que nous avons étudiée en détail au chapitre 9.

Connexion par défaut

La connexion par défaut est celle qui concerne l'utilisateur appelant la procédure. Le pilote JDBC utilisé est dit « interne » (*server-side internal JDBC driver*). Il est aussi possible de connecter un autre schéma sur une base locale ou distante en utilisant explicitement un autre pilote (*server-side JDBC Thin driver*, *client-side JDBC Thin* ou *JDBC OCI driver*).

Le code suivant décrit la classe `Compagnies` qui inclut la méthode `retourneCode`. Celles-ci a pour but de retourner le code de la compagnie dont le nom est passé en paramètre de la requête. La connexion par défaut est réalisée en passant le paramètre « `jdbc:default:connection:` » à l'appel de la méthode `getConnection`.

Tableau C3-3 Connexion par défaut

Code Java	Commentaires
<pre>import java.sql.*; import oracle.jdbc.*; public class Compagnies {public static String retourneCode() throws SQLExcep- tion {String codeComp = null; try {Connection cx = DriverManager.getConnection ("jdbc:default:connection:"); String sql = "SELECT comp FROM Compagnie WHERE nomComp ='Air France'"; Statement stmt = cx.createStatement(); ResultSet rs = stmt.executeQuery(sql); rs.next(); codeComp = rs.getString(1); rs.close(); stmt.close(); } catch (SQLException e) { System.err.print(e.getMessage());} return codeComp; } }</pre>	<p>Importation des paquetages.</p> <p>Établissement de la connexion par défaut.</p> <p>Extraction du code de la compagnie.</p> <p>Gestion des erreurs</p> <p>Retour du résultat.</p>

Le tableau suivant synthétise les étapes à suivre pour pouvoir exploiter la méthode `retourneCode` en tant qu'objet de la base.

Tableau C3-4 Étapes à suivre

Ligne de commande	Commentaires
<code>javac Compagnies.java -classpath D:\oracle\ora92\jdbc\lib\ojdbc14.jar</code>	Compilation de la classe.
<code>loadjava -user soutou/ingres@cxbdsoutou -verbose Compagnies.class</code>	Chargement de la classe. Trace : arguments: '-user' 'soutou/ingres@cxbdsoutou' '-verbose' 'Compagnies.class' creating : class Compagnies loading : class Compagnies
<code>CREATE FUNCTION JavaRetourneCodeComp RETURN VARCHAR2 AS LANGUAGE JAVA NAME 'Compagnies.retourneCode()' return java.lang.String; /</code>	Publication de la méthode (sous SQL*Plus).
<code>SQL> VARIABLE le_code VARCHAR2(10); SQL> CALL JavaRetourneCodeComp() INTO :le_code; Appel terminé. SQL> PRINT le_code; LE_CODE ----- AF</code>	Test de la procédure stockée (sous SQL*Plus).

Passage de paramètres

Chaque méthode d’une classe Java (même la méthode main) peut être publiée. Le code suivant présente une classe Java hébergeant la méthode `insereEtCompteComp` qui nécessite deux paramètres d’entrée. La classe contient aussi une méthode main qui affiche la valeur des paramètres d’entrée (quel que soit le nombre de ces derniers).

Nous allons à présent stocker ces deux méthodes en procédures au niveau de la base.

- la méthode main est publiée en spécifiant explicitement trois paramètres, par exemple ;
- la méthode `insereEtCompteComp` est publiée en respectant la correspondance des types SQL et Java.

Le tableau C3-6 synthétise les étapes à suivre pour pouvoir exploiter ces deux méthodes. Dans notre jeu d’exemple, trois compagnies sont déjà stockées dans la base.

Tableau C3-5 Classe Java ayant des paramètres en entrée

Code Java	Commentaires
<pre>import java.sql.*; import oracle.jdbc.*; public class ExempleComplet { public static void main (String[] args) {System.out.println("Valeurs des paramètres : "); int i; for (i=0; i < args.length; i++) System.out.println("paramètre "+i+" = "+args[i]);}</pre>	Affichage des paramètres du main.
<pre>public static int insereEtCompteComp(String param1, String param2) throws SQLException {int retour = 0; try {Connection cx = DriverManager.getConnection ("jdbc:default:connection:"); String instruction = "INSERT INTO Compagnie VALUES (?,?)"; PreparedStatement étatPréparé = cx.prepareStatement(instruction); étatPréparé.setString(1, param1); étatPréparé.setString(2, param2); étatPréparé.executeUpdate(); instruction = "SELECT COUNT(comp) FROM Compagnie"; étatPréparé = cx.prepareStatement(instruction); ResultSet rs = étatPréparé.executeQuery(instruction); rs.next(); retour = rs.getInt(1); étatPréparé.close(); cx.close();}</pre>	Méthode insereEtCompteComp. Insertion paramétrée. Comptage des compagnies.
<pre>catch (SQLException e) { System.err.println(e.getMessage());} return retour;}</pre>	Gestion des erreurs



La sortie par défaut d'une procédure cataloguée n'est pas l'écran. Pour rendre opérationnels vos affichages via les interfaces `System.out` et `System.err`, utilisez conjointement les procédures `SET SERVEROUTPUT ON SIZE n` de `SQL*Plus` et `SET_OUTPUT(n)` du paquetage `DBMS_JAVA`. La taille du buffer Java est au minimum (par défaut) de 2 000 octets ; le maximum vaut 1 mégaoctet.

Les paramètres de sortie (déclarés en `OUT` or `IN OUT` au niveau de `PL/SQL`) doivent être déclarés au niveau de Java par une table d'un élément. Par exemple un paramètre `OUT` de type `NUMBER` devra être associé à un paramètre Java déclaré en tant que `float[] tab`. L'affectation dans Java se fera au premier indice du tableau, soit `tab [0]`.

Paquetages

De manière analogue, il est possible de publier des méthodes Java en paquetages. La syntaxe est la suivante :

```
CREATE [OR REPLACE] PACKAGE nomPaquetage {IS | AS}
    Spécifications
END nomPaquetage;
CREATE [OR REPLACE] PACKAGE BODY nomPaquetage {IS | AS}
    Implémentations
END nomPaquetage ;
```

L'encapsulation dans un paquetage des deux méthodes de la précédente classe Java est réalisée de la manière suivante :

```
CREATE PACKAGE PaquetageJava IS
    PROCEDURE ProgPrincipal (p1 IN VARCHAR2, p2 IN VARCHAR2, p3 IN
        VARCHAR2);
    FUNCTION procédureDoubleEmploi (p1 VARCHAR2, p2 VARCHAR2)
        RETURN NUMBER;
END PaquetageJava;
/
CREATE PACKAGE BODY PaquetageJava IS
    PROCEDURE ProgPrincipal (p1 IN VARCHAR2, p2 IN VARCHAR2, p3 IN
        VARCHAR2)
        AS LANGUAGE JAVA NAME 'ExempleCompleet.main(java.lang.String[])';
    FUNCTION procédureDoubleEmploi (p1 VARCHAR2, p2 VARCHAR2)
        RETURN NUMBER
        AS LANGUAGE JAVA
        NAME 'ExempleCompleet.insereEtCompteComp(java.lang.String,java.
            lang.String)
            return int';
END PaquetageJava ;
/
```

L'appel de la méthode main est réalisé de la manière suivante :

```
SET SERVEROUTPUT ON SIZE 5000
CALL DBMS_JAVA.SET_OUTPUT(5000);
CALL PaquetageJava.ProgPrincipal('SQL', 'Oracle', 'Super!');
```

Déclencheurs

Oracle permet de programmer un déclencheur à l'aide d'une méthode Java stockée en tant que procédure. Ce mécanisme est aussi valable pour les déclencheurs de type `INSTEAD OF`.

L'exemple suivant décrit la méthode `affiche` de la classe Java `DeuxièmeExemple` qui affiche à l'écran le paramètre d'entrée dans un message prédéfini :

```
public class DeuxièmeExemple
{ public static void affiche(String param1)
  {System.out.println("La compagnie "+param1+" a été détruite."); } }
```

Le code du déclencheur a été décrit dans le paragraphe « Dans un déclencheur » de la section « Appel de la méthode ». L'appel de la procédure est déclenché lors de la suppression d'une compagnie. Le message affiché inclut le nom de la compagnie supprimée :

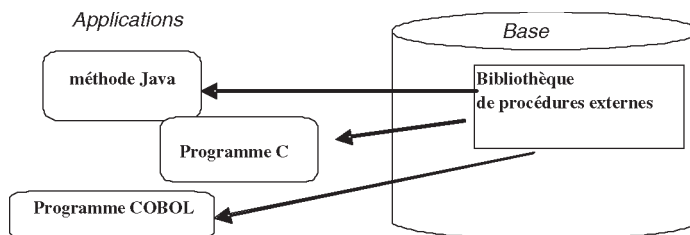
```
SQL> SET SERVEROUTPUT ON SIZE 5000
SQL> CALL DBMS_JAVA.SET_OUTPUT(5000);
Appel terminé.

SQL> DELETE FROM Compagnie WHERE comp = 'SQL5';
La compagnie Oracle AirLines5 a été détruite.
1 ligne supprimée.
```

Procédures externes Java

Le principe des procédures externes (*external procedures*) existe depuis la version 8 d'Oracle. Ce mécanisme offre la possibilité de faire appel à des programmes écrits dans des langages divers (C, C++, COBOL, Java, etc.). Ces programmes ne sont pas stockés dans l'environnement d'Oracle comme l'illustre la figure suivante :

Figure C3-3 Procédures externes



Comme dans le cas des procédures stockées, l'exploitation d'une procédure externe Java nécessite plusieurs phases :

- compilation du programme externe (ici de la classe Java) ;
- création d'une librairie ;
- publication de la procédure externe au travers d'une spécification PL/SQL ;
- appel de la procédure externe au travers de sa spécification.

Compilation de la classe

Pour illustrer les mécanismes à mettre en œuvre, exploitons la méthode récursive `fib` de la classe Java `Fibonacci` qui calcule le résultat de la célèbre suite (1, 1, 2, 3, 5, 8, 13, 21... : chaque terme de la suite, à partir du deuxième, est la somme des deux termes qui le précèdent) :

```
public class Fibonacci
{ public static int fib (int n)
  {if (n == 1 || n == 2)
    return 1;
   else
    return fib(n - 1) + fib(n - 2);} }
```

La compilation de cette classe produit le fichier `Fibonacci.class` qui doit être placé dans un répertoire externe à Oracle (dans l'exemple `C:\WINDOWS\Temp`).

Création d'une librairie

Le chargement d'une procédure externe consiste à créer (si elle n'existe pas déjà) une librairie qui contiendra les exécutables par la commande `CREATE DIRECTORY`. La commande suivante définit la librairie de nom `répertoireProcExternes` qui référence le répertoire contenant les exécutables. Le privilège `CREATE ANY DIRECTORY` doit être acquis pour pouvoir exécuter cette commande.

```
CREATE DIRECTORY répertoireProcExternes AS 'C:\WINDOWS\Temp';
```

Le chargement de la classe est réalisé à l'aide de la commande SQL `CREATE JAVA`.

```
CREATE JAVA CLASS USING BFILE(répertoireProcExternes, 'Fibonacci.
class');
/
```

Publication d'une procédure externe

La publication d'une procédure externe Java est similaire à celle des procédures stockées. L'instruction suivante définit un point d'entrée de la méthode `fib` dans Oracle sous la forme de la signature de la fonction PL/SQL `fibonacciExterne` :

```
CREATE FUNCTION fibonacciExterne (n NUMBER) RETURN NUMBER
      AS LANGUAGE JAVA NAME 'Fibonacci.fib(int) return int';
/
```

Appel d'une procédure externe

Comme dans le cas des procédures stockées, l'appel d'une procédure externe peut être réalisé sous SQL*Plus, à partir d'une commande SQL, d'un programme PL/SQL (bloc, fonction ou procédure) ou d'un déclencheur.

Dans notre exemple, appelons la fonction sous SQL*Plus en demandant la somme des sept premiers termes, comme l'illustre la trace suivante :

```
SQL> VARIABLE n NUMBER
SQL> VARIABLE résultat NUMBER
SQL> EXECUTE :n := 7;
Procédure PL/SQL terminée avec succès.

SQL> CALL fibonacciExterne (:n) INTO :résultat ;
Appel terminé.
SQL> PRINT résultat
      RÉSULTAT
-----
             13
```