Cours Bases de données 2ème année IUT

Cours 6 : JDBC : ou comment lier ORACLE avec Java 1ère partie

Anne Vilnat
http://www.limsi.fr/Individu/anne/cours

Plan

Introduction

Usage

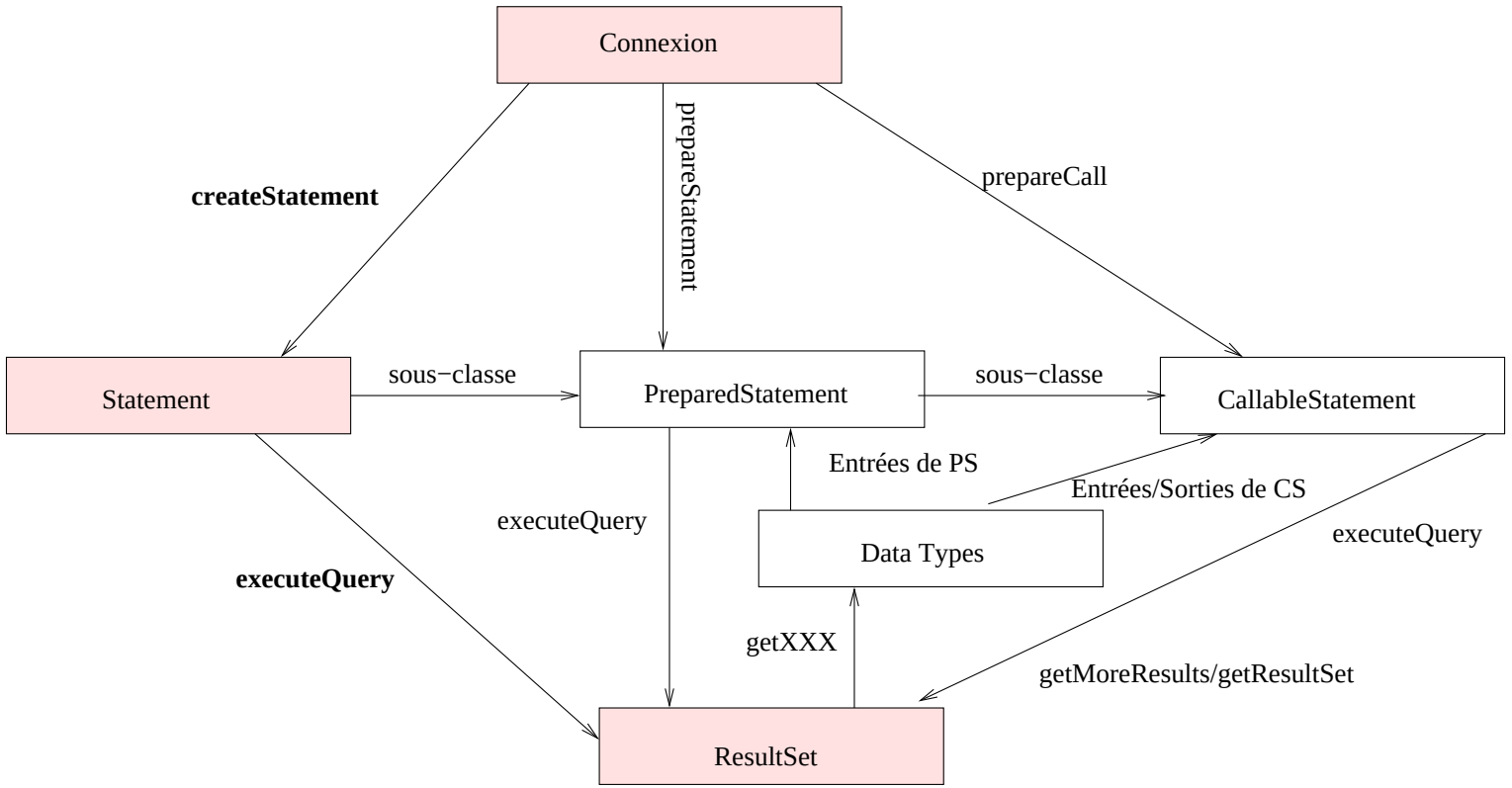
JDBC pour exécuter, depuis un programme Java, l'ensemble des ordres SQL reconnus par la base de données cible.
La base de données doit reconnaître le langage ANSI SQL-2.

Définition

JDBC (*Java DataBase Connectivity*) est une API (*Application Programming Interface*) qui permet d'exécuter des instructions SQL.
JDBC fait partie du JDK (*Java Development Kit*).
Paquetage **java.sql** :

```
import java.sql.*;
```

Les classes et interfaces du package java.sql



Etapes

Fonctionnement

Etapes d'un programme utilisant JDBC :

1. mettre en place le pilote ou *driver*.
2. établir une connexion avec une source de données.
3. effectuer les requêtes.
4. utiliser les données obtenues pour des affichages, des traitements statistiques, etc.
5. mettre à jour les informations de la source des données.
6. terminer la connexion.
7. éventuellement, recommencer en 1.

Les étapes...

exemples

1. charger un pilote *driver*
   Class.forName(“oracle.jdbc.driver.OracleDriver”);
2. créer un objet *Connection*
   Connection.maConnection=DriverManager.getConnection(url);
   url : String contenant l’adresse de la base de données
3. créer un objet *Statement*
   Statement.maRequeteSQL=maConnection();
4. envoyer la requête et récupérer le résultat dans un *ResultSet*
   ResultSet.monResultat=
   maRequeteSQL.executeQuery(texteRequeteSQL);
   texteRequeteSQL : String contenant le texte de la requête, par exemple :
   “SELECT * FROM Client”

Mise en place du pilote

2 méthodes :

Chargement statique

- enregistrer le ou les drivers(s) à utiliser
- à chaque connexion, passer comme argument l'url correspondante
- utiliser l'interface java.sql.Driver : écrire une classe Driver, pour créer une instance d'elle-même et l'enregistrer avec la méthode DriverManager.registerDriver()

pas la plus simple, ni la plus usitée...

Mise en place du pilote

Chargement dynamique

A la demande, sans noter explicitement le nom des classes :

```
try {
        Class.forName("oracle.jdbc.driver.OracleDriver");
}
catch (Exception e){
        System.out.println("Impossible de charger le driver");
        return;
}
```

Des pilotes existent pour mySQL, postGresSQL, ACCESS,...

Nommage des bases de données

Dérivée des url d'internet.

Schéma général

jdbc:<sous-protocole>:<compléments>
jdbc = protocole
sous-protocole : pour distinguer le type de pilote jdbc oracle:thin à l'IUT
complements : la base de données. Syntaxe : login/motDePasse@ordinateur:port:base.
Exemple : toto/mdpToto@orasrv1.ens.iut-orsay.fr:1521:etudom

Connexion

par la méthode getConnection de DriverManager :

Exemple

```
import java.net.*;
import java.sql.*;
String url=
 "jdbc:oracle:thin:toto/mdpToto@srv1.ens.iut-orsay.fr:1521:etudom
try {
        Class.forName("oracle.jdbc.driver.OracleDriver");
}
catch (Exception e){
        System.out.println("Impossible de charger le driver");
        return;
Connection maConnexion=DriverManager.getConnection(url); }
```

Connexion

ou en utilisant les objets DataSource

Exemple

```
import java.sql.*;
import oracle.jdbc.pool.*;
public class TestDataSource {
  public static void main(String args[])
        throws ClassNotFoundException, SQLException {
    OracleDataSource ds = new OracleDataSource();
    ds.setDriverType('thin");
    ds.setServerName'("srv1");
    ds.setPortNumber(1521);
    ds.setDataBaseName("etudom");
    ds.setUser("toto");
    ds.setPassword("mdpToto");

Connection maConnexion=DriverManager.getConnection(url); }
```

Pour dialoguer : Statement

Exemple

```
Statement monInstruction = maConnexion.createStatement();
```

suivant l'instruction SQL

Instructions SQL	Méthode	Type retourné	Valeur retournée
SELECT	executeQuery	ResultSet	Lignes de résultat
UPDATE, INSERT,DELETE	executeUpdate	int	Nb lignes modifiées
Autres	execute	boolean	Faux si erreur

Consultation et récupération de données

Exemple

```
ResultSet monRésultat = monInstruction.executeQuery(
    "SELECT login, nomClient FROM toto.Client");
```

ResultsSet et ses méthodes

Résultat dans un ResultSet

Parcours analogue à celui d'un curseur avec la méthode next, et accès aux colonnes avec getXXX

Exemple de parcours

```
while (monResultat.next()) {
  String nom = monResultat.getString("nomClient");
  int login=monResultat.getInt ("login");
  // traitement des données récupérées
}
```

Consultation et récupération de données

ResultsSet et ses méthodes

Le premier next positionne sur la première ligne.
Paramètres de getXXX : nom de l'attribut ou rang dans la requête (sous forme d'entier). Obligatoire pour les attributs calculés (MAX(...) ou quand les noms ne sont pas connus (SELECT *...)

Exemple de parcours

```
while (monResultat.next()) {
  String nom = monResultat.getString(2);
  int login=monResultat.getInt (1);
  // traitement des données récupérées
}
```

Les correspondances de types

Type SQL	Type Java
CHAR, VARCHAR2,	String
NUMERIC, DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT, DOUBLE	double
BINARY, VARBINARY, LONGVARBINARY	byte [ ]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

Les valeurs NULL

Problème : reconnaître dans Java le cas d'une valeur NULL.
Conventions :

- Pour les méthodes getString(), getObject(), getDate(), ... : Null Java (il existe)
- Pour les méthodes getInt(), getByte(), getShort(), ... : la valeur 0 est renvoyée
- Pour la méthode getBoolean(), la valeur Faux est renvoyée.

MAIS pas correct pour reconnaître des valeurs non renseignées dans la base...
D'où la méthode wasNull() de ResultSet.
Fonctionnement :

- lire la donnée,
- tester avec wasNull() si elle vaut NULL au sens SQL

Accès et mise à jour

Pour INSERT, DELETE et UPDATE...

La classe Statement a : executeUpdate()
Elle retourne un int qui contient le nombre de lignes affectées par l'instruction.

Exemple

```
int nbLignes = monInstruction.executeUpdate(
   "INSERT INTO toto.Client(login, nomClient)
            VALUES (" + numero + "," + nom + ")");
System.out.println(nbLignes + " ligne(s) insérée(s)") ;
```

Modification de la définition des données

Pour modifier la structure de la base

La classe Statement a : execute(ordreSQL)
L'ordre SQL correspond à la chaîne de caractères contenant l'ordre à éxécuter
Elle retourne un boolean qui est vrai si il n'y a pas eu d'erreur à l'éxécution..

Déconnexion

Libérer ResultSet et Statement, fermer la Connection

Exemple

```
monResultat.close() ;
monInstruction.close() ;
maConnexion.close() ;
```