Interface SQL-Langages hôtes Java DataBases Connectivity JDBC

Hala Skaf-Molli
Skaf@loria.fr
www.loria.fr/~skaf

Interface SQL-Langages hôtes

- Pourquoi ?
- Comment ?

Interface SQL-Langages hôtes

- Pourquoi?
  - SQL tout seul est insuffisant pour écrire des applications complètes.
  - Ex: application pour produire les factures, gérer les clients d'une entreprise..
- Problèmes:
  - *Impedance mismatch*
  - «Mariage» des paradigmes

Problèmes

- «Impedance mismatch»:
  - SQL a ses propres types de données;
  - le langage de programmation considéré (C, COBOL, …) a ses propres types;
  - faire correspondre les deux ensembles de types.
- «Mariage» des paradigmes:
  - SQL est un langage déclaratif orienté ensemble (le résultat d'une requête SQL est un ensemble)
  - les langages de programmation classiques sont de nature procédurale c.à.d. Orientés «tuples» ou «un à la fois»
  - faire appel à des structures itératives pour manipuler les ensembles résultants d'une évaluation de requête.

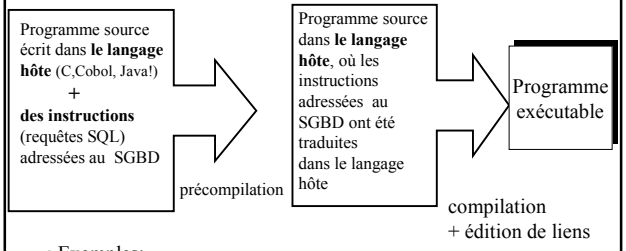
Interface SQL-Langages hôtes

- Deux approches:
  - Approche pré-compilée
  - Approche compilée

Approche pré-compilée

- Embedded SQL,
  - En C ESQL/C,
  - En Java SQLJ..

Approche pré-compilée



- Exemples:
  - dans Oracle V6, PRO*C/C++
  - dans Sybase V11, Embedded SQL (ESQL)

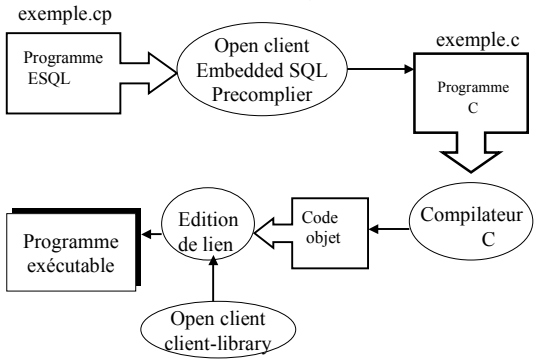
Approche pré-compilée
Embedded SQL de sybase

- Sybase Open Client Embedded SQL est un produit permettant à un programmeur d'utiliser de commandes Transact SQL dans un programme écrit dans un langage hôte tel que C ou Cobol.
- Chaque commande Transact SQL est précédée par le mot clé exec sql et termine par «;»
  - Exemple: la table Livre(titre_id,titre,prix)
    - exec sql update Livre set prix=prix*1.10; // ESQL/C

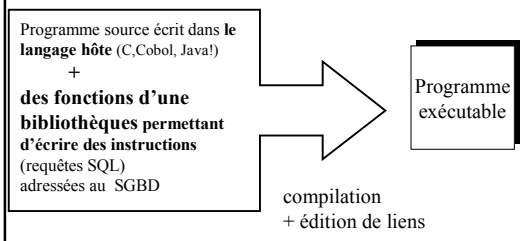
// **SQLJ**

```
int n;
#sql { INSERT INTO emp VALUES (:n)};
```


ESQL/C



Approche compilée



- Exemples:
  - JDBC, dans Sybase V11, DB_library\C

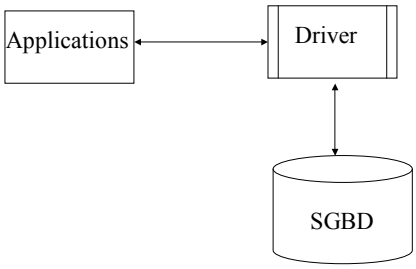
Approche compilée

- JDBC: Java DataBase Connectivity.
- API Java: accéder à n'importe quelle BD à partir d'un programme Java
- Avantages
  - Portabilité
  - Indépendance des constructeurs de SGBD

JDBC

- Etablir une connexions avec une base de données.
- Envoyer des requêtes SQL à cette base
- Traiter les résultats des requêtes

Programming With Database



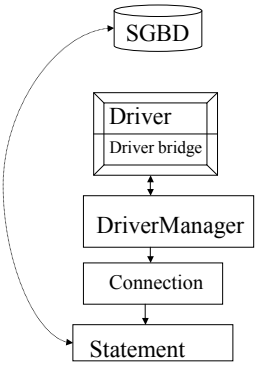
JDBC API

- Package: java.SQL
- java.SQL contient 8 interfaces permettent:
  - donner le nom de pilote jdbc utilisé: **Diver**
  - se connecter à une base de données: **Connection**
  - Création et exécution des requêtes**: Statment**, **PreparedStatement**, **CallableStatement**
  - Traitement des résultats: **ResultStatement**
- Meta-données sur la connexion et sur les résultats **DatabaseMetaData**, **ResultSetMetaData**

Interface	Description
java.sql.Driver	Locates the driver for a database URL
java.sql.Connection	Connection to a specific database
java.sql.Statement	Executes SQL statements
java.sql.PreparedStatement	Handles parameterized SQL statements
java.sql.CallableStatement	Handles database stored procedure calls
java.sql.ResultSet	Gets the results of SQL statements
java.sql.DatabaseMetaData	Used to access a variety of information about connection's DBMS and database
java.sql.ResultSetMetaData	Used to access a variety of information describing a ResultSet's attributes

JDBC API

- **java.sql package**
- **java.sql.DriverManager**
  - Obtain a DB connectio
- **java.sql.Connection**
  - A connection with one DB
- **java.sql.Statement**
  - A SQL query



Connexion : URL

<protocol>:<subprotocol>:<datasource>

- <protocol> : jdbc
- <subprotocol> : le nom de driver ou le nom de mécanisme de connexion utilisé par la base de données..
- <datasource>: le nom de la base de données ou l'adresse pour identifier la base de données...

Exemple

- URL Sybase

jdbc:sybase:Tds:serpent.atela.uhp-nancy.fr:4100/rnc3

- URL Oracle

jdbc:oracle:thin:@panoramix:1521:depinfo

- Le driver de sybase:

com.sybase.jdbc.SybDriver

- Le driver d'oracle

oracle.jdbc.driver.OracleDriver"

// **Creation d'une table**

```
import java.sql.*;
    public class Create {
      public static void main (String args[]) {
      String url ="jdbc:oracle:thin:@panoramix:1521:depinfo" ;
       Connection con;
       Statement stmt;
        String login="",  passwd="";
        try {
          login=args[0];
          passwd=args[1];
        } catch (Exception e) {
          System.err.println("usage : Create login password");
          System.exit(1);}
        String  query ="create table client(nom varchar(20),prenom varchar(20), age int)";
            try {// enregistrement du driver
          Class.forName("oracle.jdbc.driver.OracleDriver");
        } catch(ClassNotFoundException e){
          System.err.print("ClassNotFoundException");
          System.err.println(e.getMessage());
        } try { // connexion et execution de la requete
          con = DriverManager.getConnection (url,login,passwd);
          stmt = con.createStatement ();
          stmt.executeUpdate(query);
          stmt.close();
          con.close();
            } catch (SQLException ex) {
          System.err.println (" SQLException caught: "+ ex.getMessage());}  }}
```


// **Insertion des données dans une table**

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

public class InsertTable {
  public static void main (String args[]) {
       String url ="jdbc:oracle:thin:@panoramix:1521:depinfo" ;
       Connection con;
       Statement stmt;
       String login="skaf";
      String password="halaskaf";
       try { // enregistrement du driver
            Class.forName("oracle.jdbc.driver.OracleDriver");
          } catch(ClassNotFoundException e) {
               System.err.print("ClassNotFoundException");
               System.err.println(e.getMessage());  }
        try {
          con = DriverManager.getConnection (url, login, password);
          stmt = con.createStatement ();
          stmt.executeUpdate("insert into client values('Titi', 'lala',20)");
          stmt.executeUpdate("insert into client values('Dupond', 'Paul',30)");
          stmt.executeUpdate("insert into client values('Momo', 'toto',10)");
          stmt.close();
          con.close();
             } catch (SQLException ex) {
              System.err.println (" SQLException caught: "+ ex.getMessage());  }}}
```


// **selection**

```
import java.sql.*;
  public class Select{
  public static void main (String args[]) {
        String url ="jdbc:oracle:thin:@panoramix:1521:depinfo" ;
       Connection con;
       Statement stmt;
       String query= "select nom, prenom, age from client";
       String login, password;
       String n,p;
       Int a;
            try { // enregistrement du driver
                Class.forName("oracle.jdbc.driver.OracleDriver");
                 } catch(ClassNotFoundException e){
                    System.err.print("ClassNotFoundException");
                    System.err.println(e.getMessage());  }
            try { // connexion et execution de la requete
                  con = DriverManager.getConnection (url,login,password);
                    // Create a Statement object so we can submit SQL statements to the driver
                     stmt = con.createStatement ();
                    ResultSet rs= stmt.executeQuery(query);  // Submit a query
              while ( rs.next()) {
                       n = rs.getString(1); // nom
                       p = rs.getString(2); // prenom
                       a = rs.getInt(3); // age
                       System.out.println("nom: "+n+ " prenom: "+p+ "ge: "+a);}
              stmt.close();    // Close the statement
              con.close();  // Close the connection
              } catch (SQLException ex) { System.err.println (" SQLException caught: "+
    ex.getMessage());  } }}
```


Java.sql.ResultSet

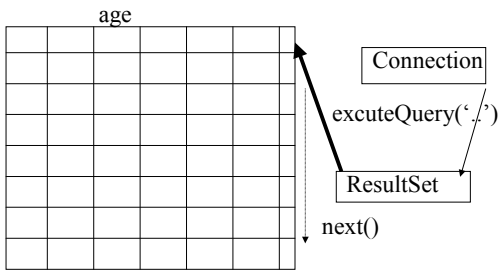


Table 2: Standard mapping from SQL types to Java types.

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

Table 3: Standard mapping from Java types to SQL types.

The mapping for String will normally be VARCHAR but will turn into LONGVARCHAR if the given value exceeds the drivers limit on VARCHAR values. Similarly for byte[] and VARBINARY and LONGVARBINARY.

Java Type	SQL type
String	VARCHAR or LONGVARCHAR
java.math.BigDecimal	NUMERIC
boolean	BIT
byte	TINYINT
short	SMALLINT
int	INTEGER
long	BIGINT
float	REAL
double	DOUBLE
byte[]	VARBINARY or LONGVARBINARY
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP

		SMALLINT	INTEGER	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC	BIT	CHAR	VARCHAR	LONGVARCHAR	BINARY	VARBINARY	LONGVARBINARY	DATE	TIME	TIMESTAMP
getByte	X	x	x	x	x	x	x	x	x	x	x	x	x						
getShort	x	X	x	x	x	x	x	x	x	x	x	x	x						
getInt	x	x	X	x	x	x	x	x	x	x	x	x	x						
getLong	x	x	x	X	x	x	x	x	x	x	x	x	x						
getFloat	x	x	x	x	X	x	x	x	x	x	x	x	x						
getDouble	x	x	x	x	x	X	x	x	x	x	x	x	x						
getBigDecimal	x	x	x	x	x	x	X	x	x	x	x	x	x						
getBoolean	x	x	x	x	x	x	x	X	x	X	x	x	x						
getString	x	x	x	x	x	x	x	x	x	x	X	X	x	x	x	x	x	x	x
getBytes														X	X	x			
getDate											x	x	x				X		x
getTime											x	x	x					X	x
getTimestamp											x	x	x				x		X
getAsciiStream											x	x	X	x	x	x			
getUnicodeStream											x	x	X	x	x	x			
getBinaryStream														x	x	X			
getObject	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Table 1: Use of ResultSet getXXX methods to retrieve common SQL data types
An "x" means that the given getXXX method can be used to retrieve the given SQL type.
An "X" means that the given getXXX method is recommended for retrieving the given SQL type.


```
// une requête dynamique ...
import java.lang.*;
import java.sql.*;
public class Dynamique {
  public static void main (String args[]) {
       String url ="jdbc:oracle:thin:@panoramix:1521:depinfo" ;
       Connection con;


        try { // enregistrement du driver
        Class.forName("oracle.jdbc.driver.OracleDriver");
           } catch(ClassNotFoundException e) {
                 System.err.print("ClassNotFoundException"+e.getMessage()); }
       try {   con = DriverManager.getConnection (url, login, password);
          // Precompile SQL statement
        PreparedStatement stmt = con.prepareStatement("select from client
  where  nom = ? and prenom = ?");
           stmt.setString(1,"Titi");
           stmt.setString(2,"Lala");
        ResultSet rs= stmt.executeQuery();  // Submit a query
           while ( rs.next())  {
                  String n = rs.getString(1); // nom
                   String p = rs.getString(2); // prenom
                    int  a = rs.getInt(3); // age
                    System.out.println("nom: "+n+ " prenom: "+p+ "ge: "+a);         }
                stmt.close();
               con.close();
             } catch (SQLException ex) {
            System.err.println (" SQLException caught: "+ ex.getMessage());  }
      }}
```


SQLJ et JDBC

// **SQLJ**

```
int n;
#sql { INSERT INTO emp VALUES (:n)};
```

// **JDBC**

```
int n;
Statement stmt = conn.prepareStatement  ("INSERT INTO emp VALUES (?)");
stmt.setInt(1,n);
stmt.execute ();
stmt.close();
```

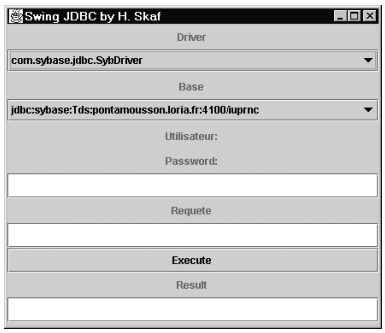

Accès aux Méta données

- **ResultSetMetaData** contient des informations sur le nombre, le type et certaines propriétés des colonnes d'un **ResultSet**

- Consultation des méta-données par les méthodes
  - **getColumnCount** ( ) : nombre de colonnes
  - **getColumnTypeName** (num_col) : nom du type de la colonne
  - **getColumnName** (num_col) : nom de la colonne
  - **getColumnLabel** (num_col) : label de la colonne
  - ...


```
import java.lang.*;
import java.sql.*;
  public class Meta{
  public static void main (String args[]) {
  String url ="jdbc:oracle:thin:@panoramix:1521:depinfo" ;

        Connection con;
        Statement stmt;
        String colnom;
        String query ="select nom, prenom, age from client";
        String login="",  passwd="";
        try {
          login=args[0];
          passwd=args[1];
        } catch (Exception e) {
          System.err.println("usage : Meta login password");
          System.exit(1);
        }
        try { // enregistrement du driver
                Class.forName("oracle.jdbc.driver.OracleDriver");

           //   Class.forName ("com.sybase.jdbc.SybDriver");
              } catch(ClassNotFoundException e){
                System.err.print("ClassNotFoundException");
                System.err.println(e.getMessage());
                }
          try { // connexion et execution de la requete
                con = DriverManager.getConnection (url,login,passwd);
                // Create a Statement object so we can submit
                // SQL statements to the driver
                stmt = con.createStatement ();

                ResultSet rs= stmt.executeQuery(query);  // Submit a query
              ResultSetMetaData rsmd = rs.getMetaData();
              int nbcol = rsmd.getColumnCount();
              for (int i=1; i <= nbcol; i++)

               System.out.println("Colonne : "+rsmd.getColumnName(i));
                System.out.println("Type : "+rsmd.getColumnTypeName(i) );

              System.out.println(" les donnees");
              while ( rs.next())
                {
                    String n = rs.getString(1); // nom
                     String p = rs.getString(2); // prenom
                      int  a = rs.getInt(3); // age
                      System.out.println(" "+n+ " "+p+ "  "+a);
                }
              stmt.close();    // Close the statement
                con.close();  // Close the connection
              } catch (SQLException ex) {
              System.err.println (" SQLException caught: "+ ex.getMessage());

                }
        }
}
```

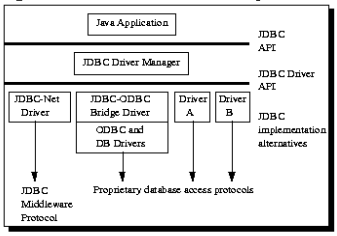
Une application avec Swing et JDBC



JDBC drivers

Selon JavaSoft: 4 types…

Figure 1.3: shows various driver implementation possibilities.



Source: http://java.sun.com/products/jdk/1.3/docs/guide/jdbc/

Type 1 JDBC-ODBC bridge

Translates JDBC calls into ODBC calls and passes them to an ODBC driver. Some ODBC software must be resident on the client machine. Some client database code may also reside on the client machine.

ODBC: Open DataBase Connection (accès aux bases de données dans le monde Microsoft..)

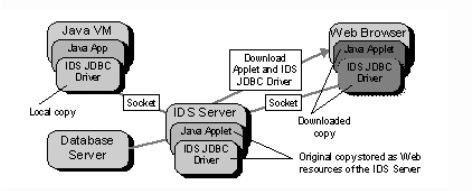
Type 2 native-API partly-Java driver

Converts JDBC calls into database-specific calls. This driver, which communicates directly with the database server, also requires some binary code on the client machine...

Type 3 net-protocol all-Java driver

Communicates to a middle-tier server using a DBMS-independent net protocol. A middle-tier gateway then converts the request to a vendor-specific protocol.

Type 3 : Exemple



Source http://www.idssoftware.com/jdriver.htm

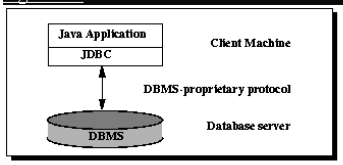
Type 4 native-protocol all-Java driver

Converts JDBC calls to the vendor-specific DBMS protocol, allowing client applications direct communication with the database server.

Le driver utilisé dans les exemples est de ce type...

Type 4 native-protocol all-Java driver

Figure 1.1: illustrates a two-tier architecture for data access.



Source:
http://java.sun.com/products/jdk/1.3/docs/guide/jdbc/

Conclusion

- JDBC offre une couche d'abstraction indépendante de:
  - base de données...
  - plate-forme système…
  - une base pour une utilisation plus 'user-friendly'
    - embedded SQL for Java : SQLJ
    - "Object/relational" mapping: JavaBlend