



Fabien Coelho
MINE S ParisTech
fabien.coelho@mines-paristech.fr

Composé avec BeTeX

Fabien Coelho

Systèmes de Gestion de Bases de Données DBMS : Data Base Management Systems

- données** volumineuses et structurées
- théorie** fondée sur l'algèbre relationnelle
- services** intégrité, cohérence, droits d'accès
- SQL** langage d'interrogation standard
- ODBC** interface standard pour C ou C++
- acteurs** Oracle, Sybase, Informix, IBM DB2, MS SQL Server...
- libres** PostgreSQL, MySQL, MSOL...

JDBC

JAVA :

3

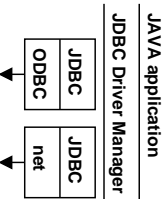
Fabien Coelho

les 2 API de JDBC

JAVA application

user API

driver API



API drivers

- ajout de nouveaux *drivers*
- driver* = un programme de connexion
- interaction avec le *driver manager*

API utilisateur

- connexion via un *driver* + URL
- requêtes SQL et résultats
- traductions types SQL ↔ JAVA
- metadatas* informations sur la base

JDBC

JAVA :

5

Fabien Coelho

Surviv d'une connexion à une base de données

- chargement du *drivers* à la main
`Class.forName("org.postgresql.Driver");`
- connexion à la base avec une URL
`Connection c = DriverManager.getConnection("jdbc:...", ...);`
- création d'un *statement* (plusieurs types de *statements*)
`Statement s = c.createStatement();`
- exécution d'une requête SQL (selon type de requête)
`ResultSet r = s.executeQuery("SELECT ...");`
- extraction du résultat
`while (r.next()) col = r.getString(1);`

JAVA :

7

JDBC

7

Fabien Coelho

JDBC : Java Data Base Connection

- présentation** du modèle JDBC
- driver** classe DriverManager et URL de connexion
- connexion** classes Connection Statement ResultSet
- avancé** classes PreparedStatement CallableStatement
- conversions** types SQL ↔ Java
- conseils** d'utilisation dans une application

Id: jdbc, eak 3799 2010-05-11 08:57:58Z Fabien

JDBC

2

Fabien Coelho

JAVA :

Contenu JDBC

API connexion de JAVA à DBMS

- interfaces et classes prédéfinies
- plusieurs versions !
JDBC 1.0 SQL 2 : java, sql,
JDBC 2.0 SQL 3 : java, sql plus extensions javax, sql
- standard et portable**
 - basé sur SQL comme langage de requête !
 - modèle offert très proche de ODBC
 - gestionnaire de *drivers*
 - nommage des bases via une URL

JDBC

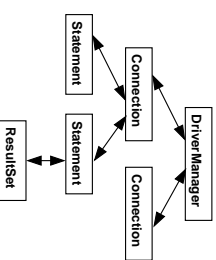
4

Fabien Coelho

JAVA :

Modèle de JDBC

- DriverManager** gestionnaire
 - enregistre les différents *drivers*
 - ouverture des *Connections*
- Connection** connexion ouverte
 - permet de créer des *Statements*
- Statement** une requête
 - exécution de requêtes SQL
 - recupération du *ResultSet*
- ResultSet** un résultat
 - énumération lignes et colonnes



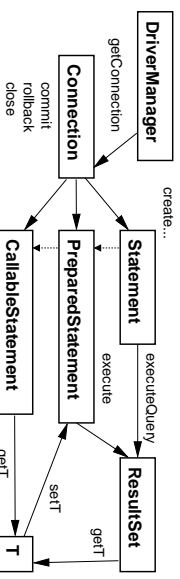
JDBC

6

Fabien Coelho

JAVA :

Relations principales



JAVA :

8

JDBC

8

Package java.sql

classes et interfaces implémentées par les *drivers*

principal DriverManager Connection Statement ResultSet
Informations ResultSetMetaData DatabaseMetaData
utils Date Time SQLException SQLWarning...
extensions Array Blob Clob Ref Struct...

Package javax.sql

– *connection pool*, événements, Row...

JDBC

9

Classe de fonctions DriverManager

– gestion des différents *drivers*, noms enregistrés auprès de Sun

4 types	binaire	réseau
générique	ODBC	protocole standard ?
spécifique	librairie du DBMS	protocole du DBMS

– enregistrement **automatique** des *drivers*

au démarrage `Class.forName(...)+CLASSPATH`

zone d'initialisation des classes `static {...}`

– `Property jdbc.drivers`: classes séparées par des :

`shell> java -Djdbc.drivers=org.postgresql.Driver ...`

```
Class.forName("org.postgresql.Driver"); // Postgres
```

```
Class.forName("org.gjt.mm.mysql.Driver"); // mysql
```

```
Class.forName("twz1.jdbc.mysql.jdbcMySqlDriver"); // z1MySQL
```

JDBC

10

Établissement d'une connexion

– trois **fonctions** de `DriverManager`
 – utilisation d'URL pour désigner le serveur
 – délais maximum d'attente, flux de messages d'erreurs...

```
public class DriverManager {
    static Connection getConnection(String url)
        throws SQLException;
    static Connection getConnection(String url, String login, String p
        throws SQLException;
    static Connection getConnection(String url, Properties info)
        throws SQLException;
    static void setLogIntimout(int seconds);
    static void setLogWriter(PrintWriter pw);
}
```

JDBC

11

```
import java.sql.*;
public class SQL01 {
    static public final String
        URL = "jdbc:postgresql://sablons.ensmp.fr/cinema";
    static public void main(String[] args) {
        try {
            String pass = new java.util.Scanner(System.in).nextLine();
            Class.forName("org.postgresql.Driver");
            Connection c = DriverManager.getConnection
                (URL, "coelho", pass);
            // ...
            c.close();
        } catch (ClassNotFoundException e) {System.err.println(e);}
        catch (SQLException e) {System.err.println(e);}
    }
}
```

JDBC

13

La Connection

– Il peut y en avoir plusieurs
 – Y compris vers la même base de données
 – Changement de base/catalogue, similaire à cd
`String cat = c.getCatalog(); // courant`
`c.setCatalog("vinsdepays"); // changement`
 – examen des données : tables, structures, procédures...
`DatabaseMetaData m = c.getMetaData();`

JDBC

15

Exécution de SQL01

```
shell> java SQL01
j'entre un mot de passe au hasard
A connection error has occurred: FATAL I :
Password authentication failed for user "coelho"
```

```
shell> java SQL01
```

Le bon mot de passe

JDBC

14

Utilisation de Connection

– Statement `PreparedStatement` `CallableStatement`
 plusieurs statements en parallèle...
 // avec une **connexion**
`Statement s = c.createStatement();`
`PreparedStatement ps = c.prepareStatement(requete_sql);`
`CallableStatement cs = c.prepareCall(requete_sql);`
 // utilisation de `s`, `ps`, `cs`...
 // fermeture de la **connexion**
`c.close();`

JDBC

16

```

- support des transactions, automatiques ou manuelles
c.setAutoCommit(false); // manuel
c.setTransactionIsolation(TRANSACTION_SERIALIZABLE);
// requêtes INSERT/UPDATE/DELETE...
c.commit(); // ok !           OU BINN
c.rollback(); // pas ok :-(-
- divers options : consultation des warnings, lecture seule...
SQLWarning w = c.getWarnings(); // donne le premier
c.clearWarnings(); // on efface !

```

Utilisation d'un Statement

- exécution de requêtes **en SQL** selon leur type

```

ResultSet r = s.executeQuery("SELECT ...");
int n = s.executeUpdate("UPDATE ...");
boolean ok = s.execute("DROP ...");

```

- suite de commandes SQL *batch* (sauf select)

```

s.clearBatch(); s.addBatch(sql); s.executeBatch();

```

- **options** temps max, tailles max (lignes, attribut), direction d'énumération, échappements, taille des transferts...

```

// connexion
Class.forName("twz1.jdbc.mysql.jdbcMySqlDriver");
Connection c = DriverManager.getConnection(
    "jdbc:mysql://palo-alt02/mysql", "coelho", "pass");

// requête
Statement s = c.createStatement();
s.setFetchSize(1000);
ResultSet r = s.executeQuery("SELECT * FROM user");

// exploitation du résultat...

// on range tout !
r.close();
s.close();
c.close();

```

Utilisation d'un ResultSet

- ensemble de lignes résultat, *en partie* sur le client...
- transfert géré par le *driver*
- passer à la ligne suivante : `r.next()` vrai si ok
- extraction d'un attribut : selon **type** et **nom** ou **numéro**

Correspondances entre les types JAVA et SQL

conversion standard des types définie dans JDBC
souple tout peut être une String
extraction méthode `getT(...)` où T est le nom du type

Correspondances types SQL - Java

```

entiers java boolean short int long
SQL BIT SMALLINT INTEGER BIGINT

réels java float double double BigDecimal
SQL REAL FLOAT DOUBLE DECIMAL NUMERIC

chaînes java String String AsciiStream
SQL CHAR VARCHAR LONGVARCHAR

binaires java Byte[] Byte[] BinaryStream
SQL BINARY VARBINARY LONGVARBINARY

tempus java java.sql.Date java.sql.Time
SQL DATE TIME TIMESTAMP

objet java sérialisés dans un attribut binaire ?

```

Informations sur un ResultSet

- à travers la classe `ResultSetMetaData`
- Informations sur les colonnes : `m.getColumn...`

```

ResultSetMetaData m = r.getMetaData();
// nombre d'attributs
int nc = m.getColumnCount();
// nom d'un attribut
String n = m.columnName(1);
// type d'un attribut (voir java.sql.Types)
int ti = m.getColumnType(1);
String ts = m.getColumnTypeName(1);
- etc.

```

```

Class.forName("org.postgresql.Driver");
Connection c = DriverManager.getConnection(
    "jdbc:postgresql://sablons/cinema", "coelho", "pass");
Statement s = c.createStatement();
ResultSet r = s.executeQuery("SELECT * FROM films");
ResultSetMetaData m = r.getMetaData();
int ncols = m.getColumnCount();

while (r.next()) {
    for (int i=1; i<ncols; i++)
        System.out.print(r.getString(i) + "\t");
    System.out.println();
}
r.close();
s.close();
c.close();

```

Exécution de SQL03

```

shell> java SQL03
mot de passe
1 Les lumières de la ville 1925-01-01 2 1
2 La rose poumpre du Caire 1985-01-01 1 2

```

Optimisation du cas simple SELECT ... ?

- à partir du DriverManager
- création d'une Connection
- création d'un Statement
- exécution d'une requête executeQuery(...)
- obtention d'un ResultSet
- extraction, conversion des colonnes...
- informations complémentaires : ResultSetMetaData
- ou!!

JDBC

25

Exécution d'un update

- requête SQL de type UPDATE INSERT DELETE...
 - retourne le nombre de lignes modifiées
 - int** n = s.executeUpdate(sql);
 - ou 0 si non approprié (CREATE...)
- ```

Class.forName("twz1.jdbc.mysql.jdbcMySqlDriver");
Connection c = DriverManager.getConnection(
 "jdbc:mysql://palo-alto2/mysql", "coelho", "pass");
Statement s = c.createStatement();
int d = s.executeUpdate("DELETE FROM user WHERE name='root'");
System.out.println("nombre de lignes effacées : " + d);
s.close();
c.close();

```

JDBC

26

### Exécution d'une requête générale

- plusieurs résultats : ResultSet ou comptes...
  - navigation parmi les résultats.. si implémenté!
- ```

boolean res = s.execute(sql_generale);
int count = 0;
while (res || count!=-1) {
    if (res) {
        ResultSet r = s.getResultSet();
        // ...
    } else
        count = s.getUpdateCount();
    res = s.getMoreResult(); // set or count
}

```

JDBC

27

Lot de requêtes (batch)

- requête avec plusieurs commandes SQL (JDBC 2.0)
 - commandes courantes : efface, ajoute, exécute
 - retourne le comptage pour chaque commande
 - pas de ResultSet par contre!
- ```

s.clearBatch();
s.addBatch("UPDATE ...");
s.addBatch("INSERT ...");
int[] r = s.executeBatch();

```

JDBC

28

### Utilisation d'un PreparedStatement

- classe similaire à Statement dont elle hérite
  - PreparedStatement ps = c.prepareStatement(...);
  - précisez une requête SQL générique (avec jokers ?)
  - SELECT \* FROM user WHERE name LIKE ? AND age>?
  - intérêt: précompilation ? mais optimisations selon valeurs...
  - fixation des paramètres comme ResultSet
- ```

ps.clearParameters();
ps.setString(1, "C%");
ps.setInt(2, 10);
ps.executeQuery(); // ou .executeUpdate() ou .execute()
ps.close();

```

JDBC

29

- ```

Class.forName("twz1.jdbc.mysql.jdbcMySqlDriver");
Connection c = DriverManager.getConnection(
 URL, "coelho", "mon mot de pass");
PreparedStatement ps =
 c.prepareStatement("select * from user where name=?");
String name = in.nextLine();

ps.clearParameters();
ps.setString(1, name);
ResultSet r = ps.executeQuery();

while (r.next())
 System.out.println(r.getString(1) + " " +
 r.getString(2) + " " +
 r.getString(3));
r.close(); ps.close(); c.close();

```

JDBC

30

### Exécution de SQL05

```

shell> java SQL05
root
localhost root 566f3e6d598e0c9b
palo-alto root 566f3e6d598e0c9b
palo-alto.ensmp.fr root 566f3e6d598e0c9b

```

JDBC

31

### Les CallableStatement

- appel une procédure stockée stored procedure
- hérite de PreparedStatement
- CallableStatement** cs =
- entées : comme PreparedStatement
- sorties : enregistrement préalable
- cs.registerOutParameter(1, Types.VARCHAR);
- extraction avec méthode getT(n) où T est un type

JDBC

32

### Classe DatabaseMetaData

- informations sur la base de données
- retourné par `c.getMetaData()` de `Connection`
- liste des tables : `getCatalogs()`
- quelques centaines de routines d'interrogation...

### Classe Types

- ensemble de constantes pour décrire les types SQL
- `BIT`, `FLOAT`, `INTEGER`, `TIME`, `DECIMAL`, `VARCHAR`...
- utilisé pour identifier les types : `registerOutParameter(...)`
- et pour l'examen de tables...

JDBC

33

### Conseils sur l'utilisation de JDBC

- encapsuler (cacher) tout dans une classe spécifique
- requêtes diverses
- fonctions utilitaires (resultat vers HTML ? vers AVVT ?)
- gestion des erreurs ? pas forcément...
- prévoir l'utilisation par des `Threads` : synchronisation!
- tester dans un contexte simple `main`
- plus difficile avec `Servlet/JSP/EJB`

JDBC

34

### Modèle d'utilisation

- réalisation d'applications client-serveur
- serveur lourd (DBMS) stocke les données
- *sécurité, standard...*
- client léger (JNA) propose une interface
- *portabilité, apparences...*
- orienté utilisateur, présentation
- composition de plusieurs services

JDBC

35

```
import java.sql.*;
import java.util.*;
/** acces a la base Cinema... */
public class CinemaDB
{
 // ATTRIBUTS
 protected Connection dbc;
 protected Statement sta;

 // CONSTRUCTOR
 public CinemaDB(Connection dbc)
 throws SQLException
 {
 this.dbc = dbc;
 this.sta = dbc.createStatement();
 }
}
```

JDBC

36

```
// METHODS
protected synchronized String[] getList(String sql)
 throws SQLException
{
 List l = new ArrayList();
 ResultSet r = sta.executeQuery(sql);
 while (r.next())
 l.add(r.getString(1));
 return (String[]) l.toArray(new String[l.size()]);
}

public String[] getGenres() throws SQLException
{
 return getList("SELECT nom FROM genres ORDER BY nom");
}

public String[] getTitres() throws SQLException
{
 return getList("SELECT titre FROM films ORDER BY titre");
}
```

JDBC

37

```
// TESTS !
static public void main(String[] args) throws Exception
{
 Scanner in = new Scanner(System.in);
 System.out.print("Login: "); String login = in.nextLine();
 System.out.print("pass: "); String pass = in.nextLine();
 Class.forName("org.postgresql.Driver");
 Connection dbc = DriverManager.getConnection(
 "jdbc:postgresql://sablons/cinema", login, pass);
 CinemaDB cine = new CinemaDB(dbc);
 String[] genres = cine.getGenres();
 String[] titres = cine.getTitres();
 System.out.println(genres[2]);
}
}
```

JDBC

38

- ### Pooling : mutualisation des coûts
- partage de la connexion (longue et coûteuse à établir ?)
  - une seule suffit! important pour certaines licences de DB
  - partage des `Statement` entre `Threads` ?
  - contexte `Servlet/JSP/EJB`
  - classe `DataSource` + `JNDI`...
  - serveur `PGPOOL` externe qui cache les connexions
  - implémente la séquence d'authentification

JDBC

39

- ### Conclusion
- connexion `JAVA-DBMS` via `JDBC`
  - conforme aux standards `SQL`, proche de `ODBC`
  - modèle bien construit `Connection`, `Statement`, `ResultSet`...
  - autre ? **`XML`** `Torque` et générations automatique de classes!
  - **`ORM`** `Object-Relational Mapping`
  - se méfier : contraintes ? typage ? fait le design de la base...

JDBC

40

## List of Slides

### Titre

- 2 JDBC : Java Data Base Connection
- 3 Systèmes de Gestion de Bases de Données JDBC
- 4 Contenu JDBC
- 5 Les 2 API de JDBC
- 6 Modèle de JDBC
- 7 Survol d'une connexion à une base de données
- 8 Relations principales
- 9 Package `java.sql`
- 9 Package `javax.sql`
- 10 Classe de fondions `DriverManager`
- 11 Etablissement d'une connexion

- 12 URL de JDBC
- 14 Exécution de SQL01
- 15 La Connection
- 16 Utilisation de `Connection`
- 18 Utilisation d'un `Statement`
- 20 Utilisation d'un `ResultSet`
- 20 Correspondances entre les types JAVA et SQL
- 21 Correspondances types SQL - Java
- 22 Informations sur un `ResultSet`
- 24 Exécution de SQL03
- 25 Optimisation du cas simple `SELECT ... ?`
- 26 Exécution d'un `update`
- 27 Exécution d'une requête générale
- 28 Lot de requêtes (*batch*)
- 29 Utilisation d'un `PreparedStatement`

- 31 Exécution de SQL05
- 32 Les `CallableStatement`
- 33 Classe `DatabaseMetaData`
- 33 Classes Types
- 34 Conseils sur l'utilisation de JDBC
- 35 Modèle d'utilisation
- 39 *Profiling* : mutualisation des codes
- 40 Conclusion