JDBC: Java DataBase Conectivity

Integrantes: Pablo Benaprés M.
Tomás Girardi J.
Roberto Vargas H.

Introducción

- Que es?
  - API
  - Driver
  - Java
  - Interactúa con bases de datos
  - Ejecuta consultas
  - Recibe resultados de consultas

API's Java

- Interfaz de Programación de Aplicaciones
- Herramientas para desarrollo de aplicaciones.
  - Conjunto de clases (en paquetes)
    - Metodos e Instancias para cumplir una serie de tareas relacionadas con la función de la API.
- Ejemplos:
  - JAXP -> XML
  - Swing -> GUI
  - AWT -> GUI

Drivers JDBC

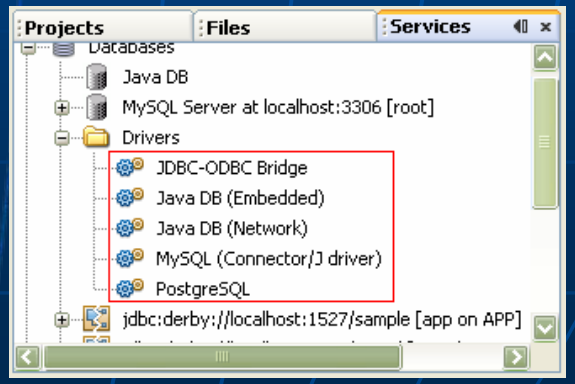
- Tipo1 - JDBC-DB Drivers
  - Para conexión con DB
  - Utilizan puente desarrollado por Sun e InterSolv.
  - Software no-Java en cliente
  - Implementado con código nativo.
- Tipo2 – Drivers API's nativas parcialmente en java
  - Código nativo con pequeña envoltura Java
  - Mayor rapidez
  - Menos seguridad: código nativo puede producir caída de la máquina virtual.
- Tipo 3 – Drivers de protocolo de red
  - Completamente en Java
  - Aplicación intermedia
  - Protocolo de red genérico que interactúa con aplicación intermedia.
- Tipo 4 – Drivers completamente Java
  - Completamente en Java
  - No requieren de aplicación intermedia
  - Interactúa directamente con Database

Historia

Año	Versión	Especificación JSR	Implementación JDK
2006	JDBC 4.0	JSR 221	Java SE 6
2001	JDBC 3.0	JSR 54	JDK 1.4
1999	JDBC 2.1		JDK 1.2
1997	JDBC 1.2		JDK 1.1

Instalación

- Instalar correctamente el JDK
  - JDBC viene incluido en el JDK por lo que no es necesario instalar nada adicional para el sistema base.
- Cada driver tiene requerimientos distintos como se verá más adelante. Los mas típicos vienen incluidos en algunas JDK (Eje: NetBeans 6.1)



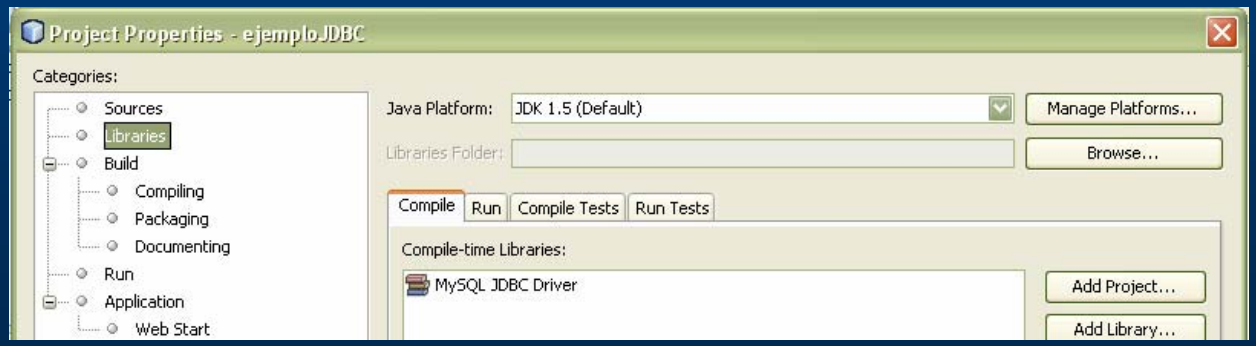
Bases de datos soportadas:

- IBM DB2
- ODBC
- Microsoft SQL Server
- Oracle
- Cloudscape
- Cloudscape RMI
- Firebird
- IDS Server
- MySQL
- PostgreSQL
- Sybase
- PointBase embedded server
- InstantDB
- Hypersonic SQL
- Informix Dynamin Server

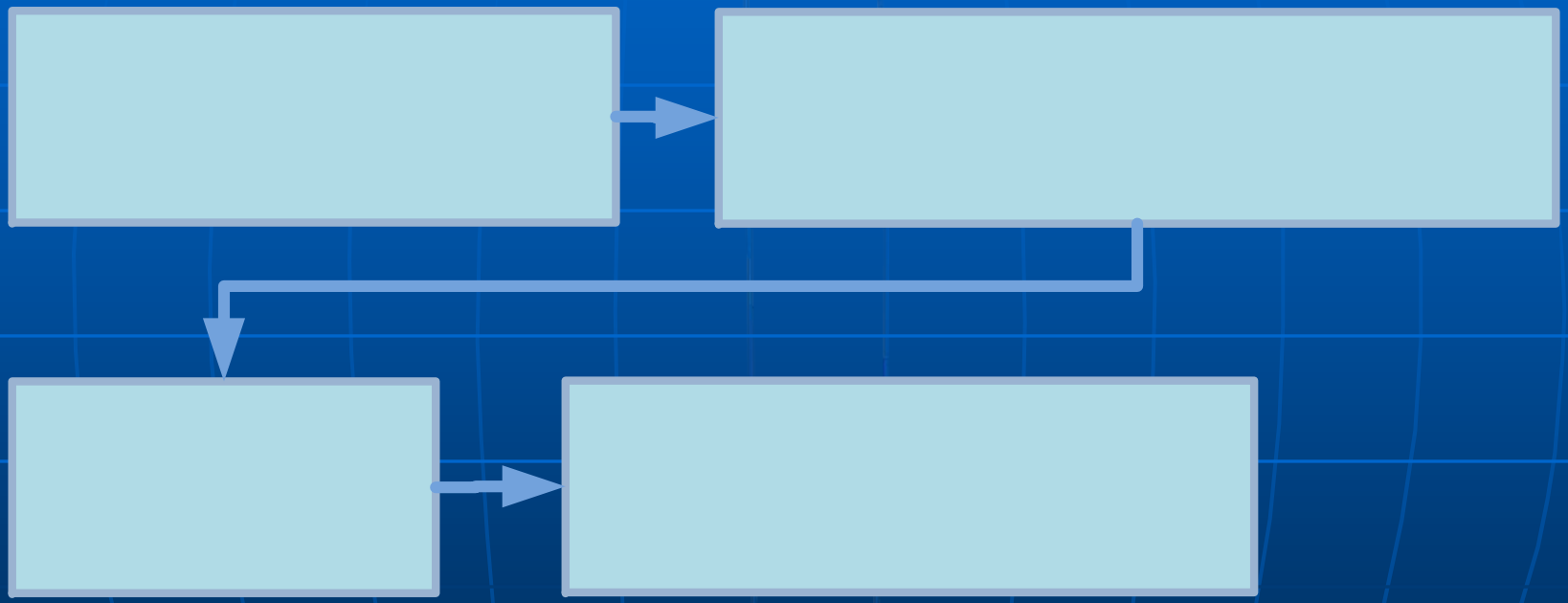
Fuente: http://www.devx.com/tips/Tip/28818

Programación Java de JDBC:

- Incluir librerías:
  - Propias la Base de Datos usada.
    - Ej: MySQL - > mysql-connector-java-5.1.5-bin
  - ¿Como?
    - Jar
      - Incluir librería (xxx.jar) dentro del jar.
      - Setear “Class-Path” en archivo “MANIFEST.MF”
        - Ej: Class-Path: mysql-connector-java-5.1.5-bin.jar
    - Ejecutando .class directamente
      - Java ejemploJDBC –classpath lib/mysql-connector-java-5.1.5-bin.jar
    - Ej: Netbeans



Funcionamiento



Cargar Driver Correspondiente

Class.forName("com.mysql.jdbc.Driver");

Programación Java de JDBC:

- Cargar los drivers:
  - Ej: JDBC-MySQL
    - Crea Instancia en DriverManager.

```
Import java.sql.*;

Class.forName(“com.mysql.jdbc.Driver”);
```

Programación Java de JDBC:

- Establecer la conexión:

```
   Connection con =
DriverManager.getConnection(
                    String url,
                    String usuario,
                    String password};
```

  - Ejemplo de URL

```
   jdbc:mysql:://<host>:<puerto>/<base de
datos>
```

Programación Java de JDBC:

- Otras bases de datos:
  - PostgreSQL (>7.0):

    ```
    Class.forName(“org.postgresql.Driver”)
      url = jdbc:postresql://<host>:<port>/<DB>
    ```

  - ODBC:

    ```
    Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”)
      url = jdbc.odbc:<DB>
    ```

Programación Java de JDBC:

- Ejecución DDL (Data Definition Language)
  - Creación de tablas:

```
String sentenciaDDL =
     “CREATE TABLE EJEMPLO(
                campo1 FLOAT,
                campo2 VARCHAR(40),
                campo3 DATE)”;
```

  - ‘;’ No necesario

Programación Java de JDBC:

- Ejecución DDL (Data Definition Language)
  - Modificación de Tablas:
    - Creación statement (solo necesario si no existe).

```
Statement st =
            con.createStatement();


st.executeUpdate(sentenciaDDL);
```

Programación Java de JDBC:

- Creación de tablas e inserción:

```
String query = “INSERT INTO EJEMPLO VALUES(
                        164836789,
                         “Roberto Vargas”,
                         31-12-86”)”;

st.executeStatement(query);
```

Programación Java de JDBC:

- Sentencias Query:
  - Ver Contenido de Tabla:

```
String sentenciaQuery = “SELECT * FROM EJEMPLO”;
ResultSet rs = st.executeQuery(sentenciaQuery)
rs.next();
float flt = rs.getFloat(“campo1”);
```

Programación Java de JDBC:

- ResultSet:
  - Multiples tipos de get:
    - Enteros:
      - `(int) getInt(String nombreColumna)`
      - `(int) getInt(int numeroColumna)`
    - Strings:
      - `(String) getString(String nombreColumna)`
      - `(String) getString(int numeroColumna)`
    - getByte(...), getBoolean(...), getDouble(...), getDate(...), etc.

(ver http://java.sun.com/j2se/1.5.0/docs/api/java/sql/ResultSet.html)

Programación Java de JDBC:

- Ahorro de código: preparedStatement!

  - Sentencias repetitivas con campos variables:

    ```
    PreparesdStatement ps =
      con.preparedStatement((String) Query)
    ```

    - String Es una sentencia normal, reemplazando los valores de los campos con '?':

    ```
    Query = "INSERT INTO LISTA VALUES(?, ?)";
    Ps.setString(1,"Fideos");
    Ps.setDouble(2,"150.0");
    (int) N = Ps.executeUpdate(void);
    ```

    executeUpdate retorna 0 como DDL, el numero de filas modificadas como Query. Útil para un "UPDATE... WHERE..."

Programación Java de JDBC:

- Integridad de datos: transacciones

Por defecto, una query (o DDL) se ejecuta en el momento que se hace un llamado a executeUpdate() o executeStatement(). Pero esto puede ser manipulado:

```
con.setAutoCommit(false);
```

De esta forma no se efectua la transacción hasta que se denote explícitamente:

```
con.setAutoCommit(false);
… varias sentencias…
con.commit();
…con.setAutoCommit(true);
```

JDBC 2.0:

- Nuevas implementaciones: cursores

```
Statement stmt = con.createStatement(
                 ResultSet.TYPE_SCROLL_(IN)SENSITIVE,
                 ResultSet.CONCUR_(READ_ONLY)UPDATABLE);
```

  - `rs.next();`
  - `rs.previous(); (JDBC 2.0)`
  - `rs.absolute(int);`
  - `rs.relative(int);`
  - `(boolean) rs.isFirst();`
  - `(boolean) rs.inLast();`
  - `(boolean) rs.isAfterLast();`

  Ver: http://java.sun.com/j2se/1.5.0/docs/api/java/sql/ResultSet.html

JDBC 2.0:

- Nuevas implementaciones: modificaciones a tiempo real:

```
Statement stmt = con.createStatement(
                 ResultSet.TYPE_SCROLL_(IN)SENSITIVE,
                 ResultSet.CONCUR_(READ_ONLY)UPDATABLE);
```

  - Manipulación directa del ResultSet:
    - `rs.updateXXX((String),(XXX));`
    - `rs.updateRow();`
    - `rs.moveToInsertRow();`
    - `rs.deleteRow();`

    Ver: http://java.sun.com/j2se/1.5.0/docs/api/java/sql/ResultSet.html

JDBC 2.0:

- Nuevas implementaciones: Actualización por lotes:

Parecido a las transacciones, pero como un batch:

```
st.addBatch((String)…);
…
(int[]) st.executeBatch();
```

Retorna el numero de modificaciones hechas en cada fila correspondiente a la posicion del arreglo.

```
st.clearBatch();
```

Referencias:

- http://www.unix.com.ua/orelly/java-ent/jenut/ch02_03.htm
- http://java.sun.com/j2se/1.5.0/docs/api/java/sql/ResultSet.html
- http://www.devx.com/tips/Tip/28818

- "Sugerencia" para estudio:
  - http://www.programacion.com/tutorial/jdbc/