Base de Datos JDBC

Unidad: 1
Laboratorio de Programación

Universidad Nacional de la Patagonia Austral
Unidad Académica Río Gallegos

Indice

- Repaso clase anterior
- Sentencias preparadas
- PreparedStatement Vs. Statement
- Transacciones. Autocommit. Commit y Rollback.

Repaso…

- JDBC es un API que permite realizar consultas a una base de datos relacional. Su diseño está inspirado en dos conocidas APIs:
  - ODBC
  - X/OPEN SQL CLI (Call Level Interface)

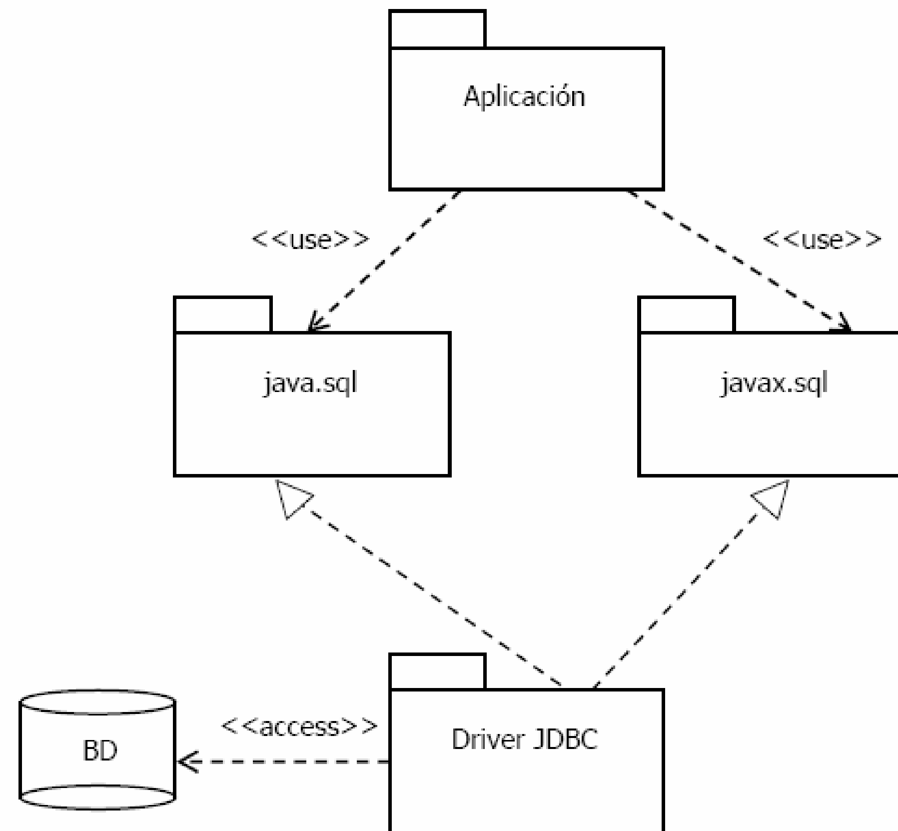
- La aplicación de Java debe tener acceso a un driver JDBC adecuado:
  - **Un driver suele ser un fichero** .jar **que contiene una implementación de todos los interfaces del API de JDBC**
  - **Nuestro código trabaja en conjunto con los paquetes** java.sql **y** javax.sql

- Este driver proporciona la comunicación entre el API JDBC y la base de datos real.

Lista proveedores de drivers JDBC

Agave Software Design	3	Oracle, Sybase, Informix, ODBC supported databases
Asgard Software	3	Unisys A series DMSII database
Borland	4	InterBase 4.0
Caribou Lake Software	3	CA-Ingres
Center for Imaginary Environments	4	mSQL
Connect Software	4	Sybase, MS SQL Server
DataRamp	3	ODBC supported databases
IBM	2 / 3	IBM DB2 Version 2
IDS Software	3	Oracle, Sybase, MS SQL Server, MS Access, Informix, Watcom, ODBC supported databases

Drivers JDBC



Tipo de clases

TIPO	Clase JDBC
Implementación	java.sql.Driver java.sql.DriverManager java.sql.DriverPropertyInfo
Conexión a base de datos	java.sql.Connection
Sentencias SQL	java.sql.Statement java.sql.PreparedStatement java.sql.CallableStatement
Datos	java.sql.ResulSet
Errores	java.sql.SQLException java.sql.SQLWarning

Requisitos para trabajar con JDBC

UNPA
Universidad Nacional de la Patagonia Austral

- Tener instalado Java y el driver JDBC en nuestra máquina.
  - Instalando una máquina virtual:
    - Descargar el último JDK de Sun e instalar, incluye los paquetes java.sql y javax.sql.
- Instalar el Sistema Gestor de Bases de Datos (MySQL):
  - Descargar e instalar MySQL
    - http://dev.mysql.com/downloads/mysql/5.0.html
  - Descargar e instalar herramientas gráficas para MySQL
    - http://dev.mysql.com/downloads/gui-tools/5.0.html
- En función del Sistema Gestor de Bases de Datos, instalar el driver apropiado (MySQL):
  - Descargar el driver JDBC para MySQL.
    - http://dev.mysql.com/downloads/connector/j/5.0.html

Pasos de JDBC

- Siete pasos básicos en el uso de JDBC

1.- Cargar el controlador (driver)
2.- Definir el URL de la Conexión
3.- Establecer la conexión
4.- Crear un objeto Statement
5.- Ejecutar una consulta (query)
6.- Procesar los resultados
7.- Cerrar la conexión

UNPA
Universidad Nacional de la Patagonia Austral

JDBC: Detalles de proceso

1. Cargar el driver

```
try {
    Class.forName("com.mysql.jdbc.Driver");
} catch(ClassNotFoundException e){
    System.out.println("Error cargando el driver: "+e);
}
```

2. Definir el URL de la Conexión

```
String dbName="Ejemplo";
String mysqlURL="jdbc:mysql:"+dbName;
```

JDBC: Detalles de proceso…

3. Establecer la conexión:

```
String username="root";
String password="";
Connection
connection=DriverManager.getConnection(mysqlURL,
username, password);
```

4. Crear un objeto Statement:

```
Statement statement = conexion.createStatement();
```

UNPA
Universidad Nacional de la Patagonia Austral

JDBC: Detalles de proceso…

5. Ejecutar una consulta (query):

```
String query="SELECT col1, col2, col3 FROM sometable";
ResultSet resultSet = statement.executeQuery(query);
```

-Para modificar la base de datos, usar executeUpdate, suministrando un string que contenga sentencias del tipo UPDATE, INSERT, o DELETE.

6. Procesar los resultados:

```
while(resultSet.next())
   System.out.println(resultSet.getString(1) + " " +
                      resultSet.getInt(2) + " " +
                      resultSet.getDouble(3));
}
```

-Primer columna tiene indice 1, NO 0

-ResultSet provee de varios métodos getXXX que toma un índice de columna o nombre y retorna el dato.

5. Cerrar la conexión: `connection.close();`

Ejemplo

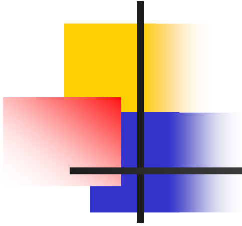
```
import java.sql.*;

public class EjemploJDBC {

    public static void main(String args[]){
        try {
        //Cargar driver especifico de base de datos
        Class.forName("com.mysql.jdbc.Driver");
        //Crear el objeto de conexion a la base de datos
        Connection conexión=
    DriverManager.getConnection("jdbc:mysql:Ejemplo");
```

Ejemplo

```
        //Crear objeto Statement para realizar queries a la base de datos
        Statement instruccion = conexion.createStatement();
        //Un objeto ResultSet, almacena los datos de resultados de una
consulta
        ResultSet tabla = instruccion.executeQuery("SELECT cod , nombre
FROM datos");
        System.out.println("Codigo\tNombre");
        while(tabla.next())
                System.out.println(tabla.getInt(1)+"\t"+tabla.getString(2));
        conexion.close();
        }
        catch(ClassNotFoundException e){ System.out.println(e); }
        catch(SQLException e){ System.out.println(e); }
        catch(Exception e){ System.out.println(e); }
    }
}
```

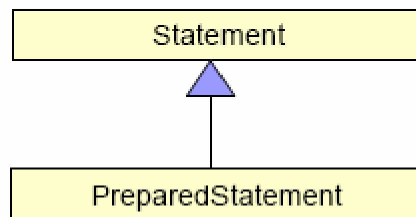


UNPA
Universidad Nacional de la Patagonia Austral

Sentencias preparadas y Transacciones

Sentencias Preparadas (1)

- Hasta ahora, para ejecutar sentencias hemos usado Statement.
- Sin embargo hay otra clase llamada PreparedStatement que hereda de la primera.



- ¿Cuál es la diferencia?
  - Creación de objeto PreparedStatement>Recibe consulta SQL.
  - Esta consulta es enviada al SGBD y compilada.
  - Cuando ejecutamos la consulta, ya ha sido previamente compilada y no necesita compilarse, sólo ejecutarse.
  - Esto es útil cuando se necesita ejecutar una misma consulta varias veces.

Sentencias Preparadas…

- Pero sobre todo es útil para utilizar sentencias SQL con parámetros:
- Es decir, podemos ejecutar la misma sentencia suministrando diferentes valores cada vez que la ejecutamos.
- Para utilizar una sentencia preparada:

```
PreparedStatement pstm = con.prepareStatement ("UPDATE COFFEES
                        SET SALES = ? WHERE COF_NAME LIKE ?")
```

- ¿A la hora de ejecutar esa sentencia, cómo pasamos los parámetros o valores a la misma?
- Utilizamos los métodos setXXX de la clase PreparedStatement

UNPA
Universidad Nacional de la Patagonia Austral

Sentencias Preparadas…

- Utilización de los métodos setXXX:
  - Dependiendo del tipo de dato correspondiente a cada marca de ?, vamos a utilizar un método distinto.
  - Por ejemplo, en el caso anterior:
  - Si la primera marca corresponde a un int, usamos:
    - pstm.setInt(1,75);
      - 1 = primera marca.
      - 75 = valor del parámetro int que estamos pasando.
  - Si queremos que la segunda marca sea un String, usamos:
    - pstm.setString(2,"Colombian");
      - 2 = segunda marca.
      - "Colombian" = parámetro string pasado

Sentencias Preparadas…

- A continuación ya podemos ejecutar la sentencia, como si se tratase de una sentencia normal:
  - **pstm.executeUpdate();**
- Pero sin argumentos…

```
Statement stm = con.createStatement();
String updateString = "UPDATE COFFEES SET SALES = 75 " +
                      "WHERE COF_NAME LIKE 'Colombian'";
stm.executeUpdate(updateString);
```

```
PreparedStatement updateSales = con.prepareStatement(
        "UPDATE COFFEES SET SALES = ? WHERE COF_NAME LIKE ? ");
updateSales.setInt(1, 75);
updateSales.setString(2, "Colombian");
updateSales.executeUpdate();
```

Realizan lo mismo

Sentencias Preparadas…

- La utilidad clara de las sentencias preparadas, es la de repetir sentencias con diferentes valores…
  - Por ejemplo cuando realizamos sentencias utilizando como parámetro el índice de un bucle.
- Una vez se establece un parámetro, éste permanece hasta que se asigne otro valor al mismo parámetro o se invoque el método clearParameters().

UNPA
Universidad Nacional de la Patagonia Austral

Sentencias Preparadas…

- Supongamos ahora que queremos actualizar los valores de la columna SALES para todos los cafés:
  - Esto podemos realizarlo con un simple bucle:

```
String updateString = "update COFFEES " +
                       "set SALES = ? where COF_NAME like ?";
PreparedStatement pstm = con.prepareStatement(updateString);
int [] salesForWeek = {175, 150, 60, 155, 90};
String [] coffees = {"Colombian", "French_Roast", "Espresso",
                      "Colombian_Decaf", "French_Roast_Decaf"};
int len = coffees.length;
for(int i = 0; i < len; i++) {
          pstm.setInt(1, salesForWeek[i]);
          pstm.setString(2, coffees[i]);
          pstm.executeUpdate();
}
```

Sentencias Preparadas…

- Mientras que el método executeQuery, devuelve el conjunto de registros seleccionados, es decir, un ResultSet.
- Cuando ejecutamos un método executeUpdate, lo que se devuelve es el número de registros que se han visto afectados o actualizados.

```
PreparedStatement updateSales = con.prepareStatement(
          "UPDATE COFFEES SET SALES = ? WHERE COF_NAME LIKE ? ");
updateSales.setInt(1, 75);
updateSales.setString(2, "Colombian");
int n = updateSales.executeUpdate();  // n = 1
```

UNPA
Universidad Nacional de la Patagonia Austral

Sentencias Preparadas…

- Cuando utilizamos el método executeUpdate para una sentencia de tipo DDL (como la creación de una tabla), en este caso, el método devuelve un valor de 0:

```
String stCreacionCoffees = "CREATE TABLE COFFEES " +
"(COF_NAME VARCHAR(32), SUP_ID INTEGER, PRICE FLOAT, " +
"SALES INTEGER, TOTAL INTEGER)";

int n = updateSales.executeUpdate(stCreacionCoffees);  // n = 0
```

Statement vs PreparedStatement

- **PreparedStatement** es más eficiente en bucles que lanzan la misma query con distintos parámetros

- **Statement** obliga a formatear los datos de la query porque ésta se envía a la BD tal cual la escribimos
  - Las cadenas de caracteres tienen que ir entre '' => código menos legible y más propenso a errores
  - Existen algunos tipos de datos (ej.: fechas) que se especifican de manera distinta según la BD

- Con **PreparedStatement** el formateo de los datos lo hace el driver

- Conclusión
  - En general, usar siempre **PreparedStatement**

Transacciones

- Hay casos en los que nos puede interesar que siempre que se realice una acción en una base de datos, se realice otra asociada…
  - Por ejemplo, si actualizamos la cantidad de café vendida en una semana, queremos también actualizar el total de café vendido.
  - No nos interesa que se realice una acción y no la otra, sino que queremos que se realicen las dos acciones como si fueran una unidad, como si fueran una única sentencia.
  - A ESTO SE LE DENOMINA UNA TRANSACCIÓN…

Transacciones…

- Por defecto, cuando realizamos una conexión con una base de datos, ésta está en modo AUTO-ENVIO O AUTOENTREGA.
  - Esto quiere decir, que después de haber sido ejecutada o completada la sentencia SQL en JAVA, se entrega inmediatamente al sistema gestor de bases de datos, que la ejecuta.

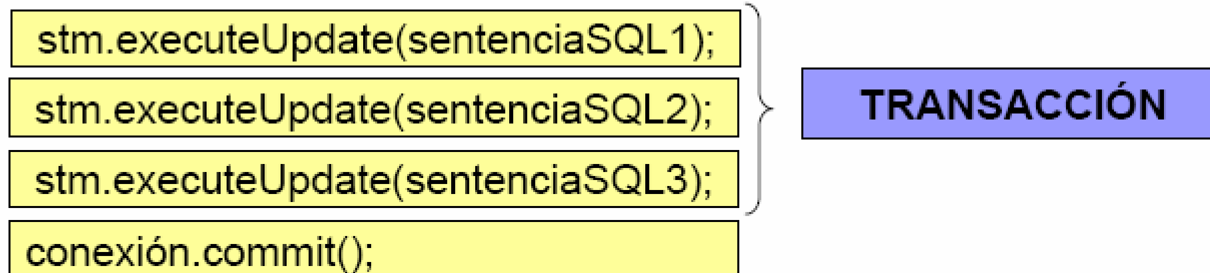
- Si queremos que dos o más sentencias se ejecuten como una única transacción, debemos desactivar el modo Auto-Envío. Método setAutoCommit(boolean b) de Connection.

```
conexion.setAutoCommit(false);
```

Transacciones…

- Lo que hace el compilador en modo Auto-Envío es llamar de manera automática al método **commit()** del objeto Connection después de cada ejecución de sentencia SQL.

- Desactivando el modo Auto-Envío, somos nosotros los que debemos llamar al método commit() y se entregarán todas las sentencias SQL previas a esa llamada como una unidad.

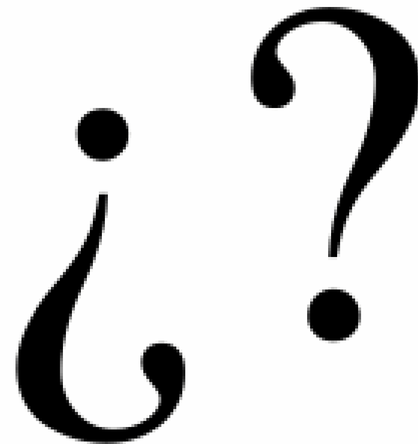


Transacciones…

- Abortar una transacción.
  - Existe una forma de deshacer los cambios realizados por una transacción.
    - Mediante el método rollback() de la clase Connection.
- ¿Cuándo utilizar este método?
- Imaginemos que durante la ejecución de una transacción que contiene varias sentencias SQL se lanza una excepción de tipo SQLException-> Ahí podemos ejecutar rollback() para dejar la BD en el estado anterior a la transacción.

UNPA
Universidad Nacional de la Patagonia Austral

Consultas…



Próxima clase

- Interfaz gráfica de Usuario (GUI)
- Ejercitación