

# Conectividad con Bases de Datos desde Java (JDBC).

**Antonio S. Cofiño**

**Marta Zorrilla**

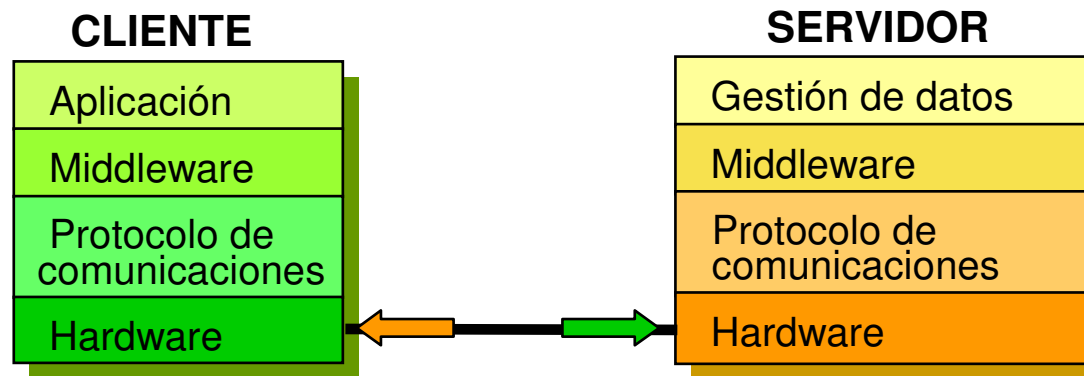
**Universidad de Cantabria**



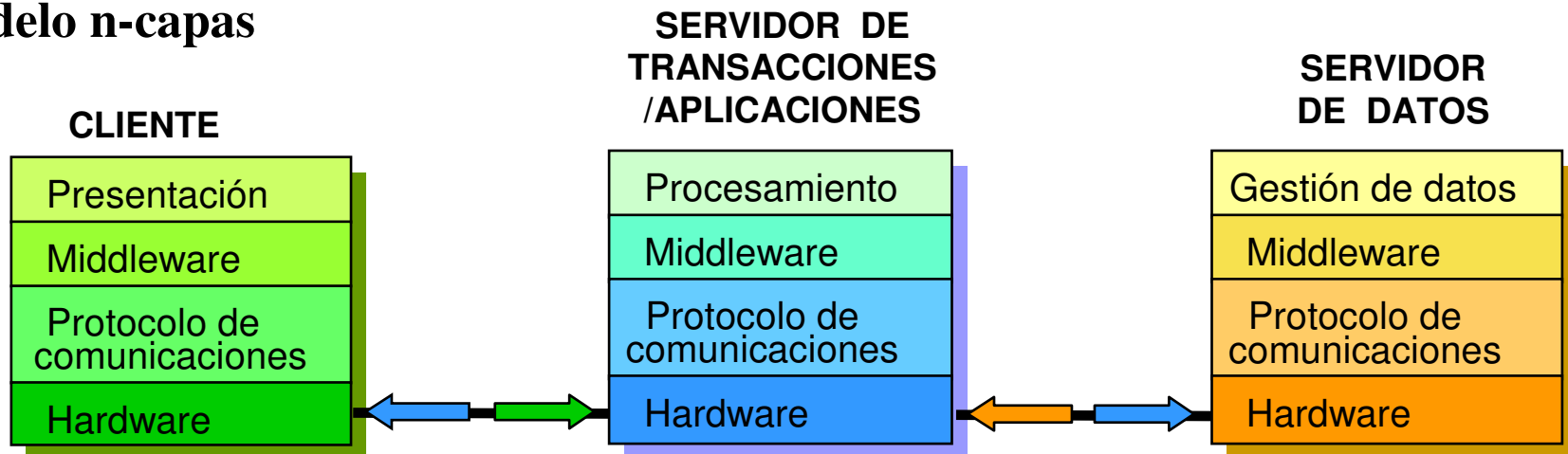
Java  Sun  
microsystems

# Aplicaciones de bases de datos

## ◆ Modelo dos capas



## ◆ Modelo n-capas



**Middleware:** Todo software distribuido necesario para la interacción entre clientes y servidores. Puede ser propietario del gestor de bases de datos que se utilice o estándar.

# Conectividad con Bases de Datos

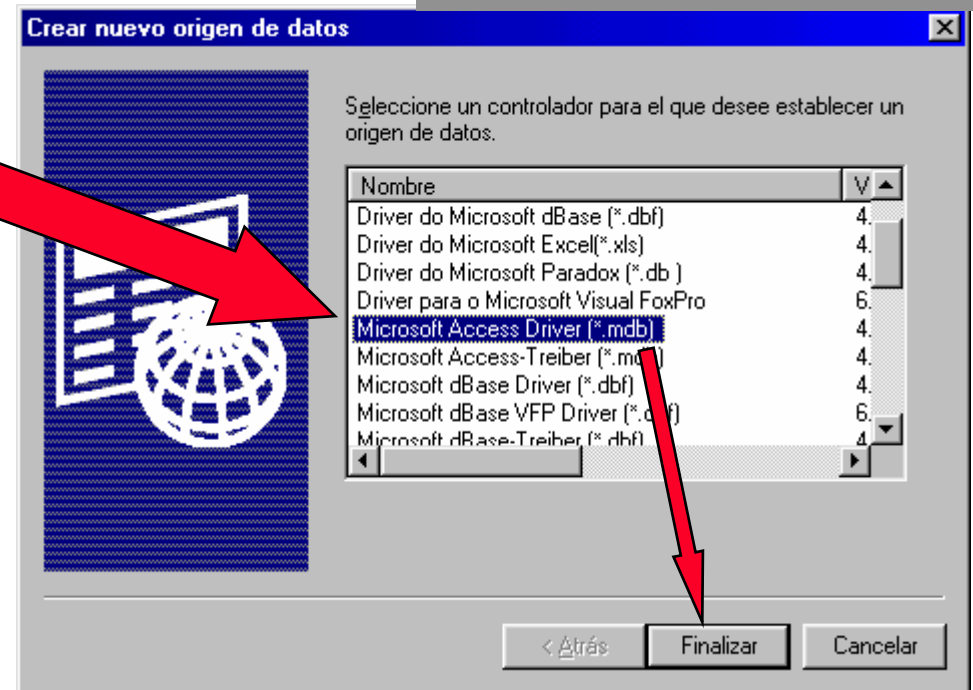
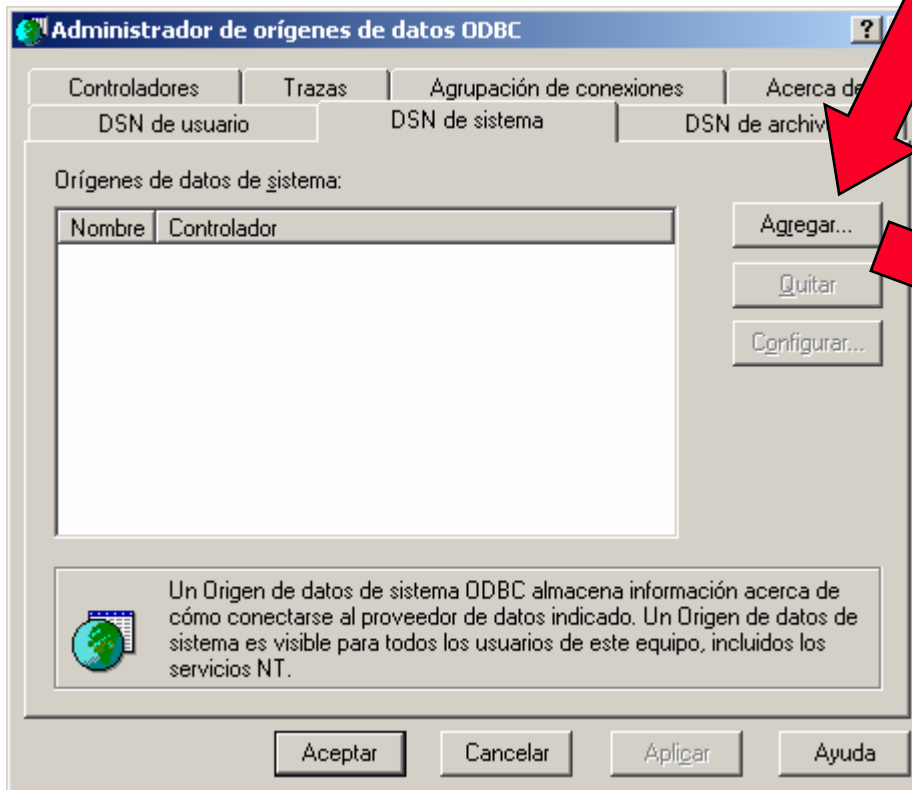
- **ODBC** (Open DataBase Connectivity): interfaz para comunicarse con SGBDR
- JAVA implementa **JDBC** debido a que **ODBC es una API en C**.
- Los Controladores **ODBC deben ser instalados en las máquinas clientes**, esto lleva conflictos con la seguridad implementada en JAVA (sobre todo en los Applets).
- Existen distintos **tipos de Controladores JDBC** que nos permiten una conectividad con los servidores de Bases de Datos:
  - Tipo 1: **puente JDBC-ODBC**
  - Tipo 2: traducen directamente el API JDBC en un API específico de la base de datos.
  - Tipo 3: traducen el API JDBC en un protocolo independiente de la base de datos. El driver JDBC no comunica directamente con la base de datos; comunica con un servidor de capa media, que a su vez comunica con la base de datos. Es el que nos permite de hacer uso del protocolo HTTP, y de esta forma poder integrar **soporte para Bases de Datos con aplicaciones WEB**.
  - Tipo 4: Los drivers se comunican directamente con la base de datos. Mejor rendimiento pero menos flexible (sobretudo si necesitamos cambiar de BD)

# Uso del puente JDBC-ODBC (Tipo 1)

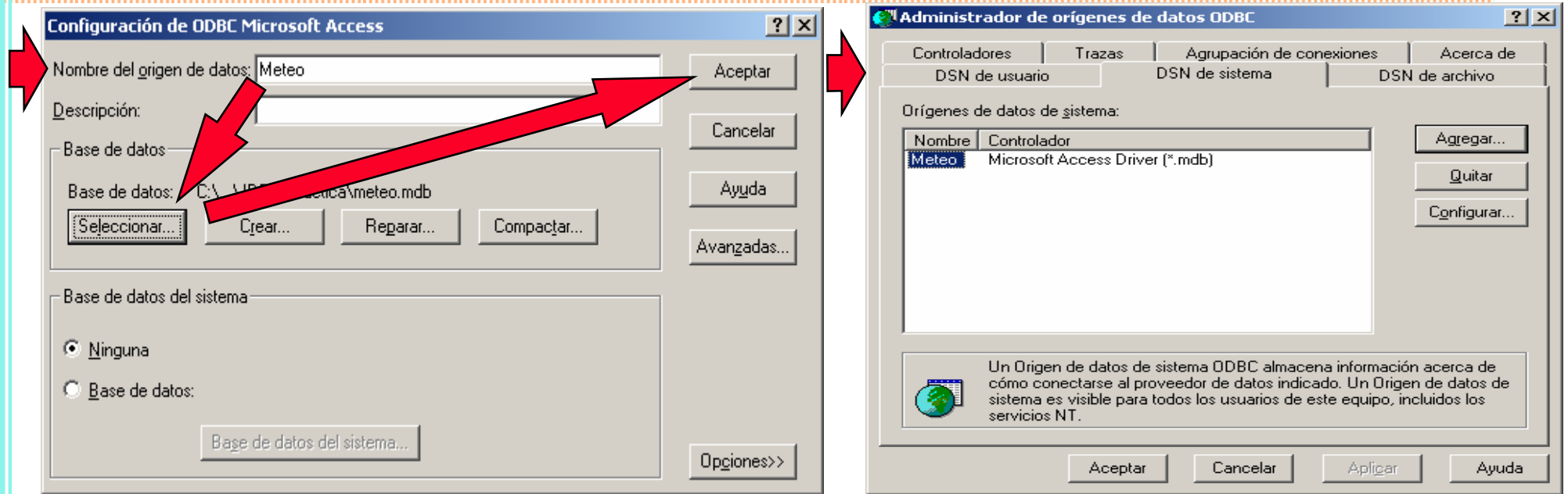
- Usar una base de Datos de MS Access mediante el origen de datos ODBC de Windows.
- En esta base de datos crearemos una tabla que luego manipularemos.



- Necesitamos configurar nuestro Archivo .mdb como origen de Datos.
- Seleccionamos el controlador para nuestro origen de datos. Microsoft Access Driver (\*.mdb)



# Uso del puente JDBC-ODBC (Tipo 1) cont.



- Damos un nombre a nuestro origen de datos (**Meteo**)
- Seleccionamos el fichero \*.mdb (la Base de Datos).
- Ya tenemos definido un origen de datos ODBC.

- Nuestro objetivo es acceder a la base de datos **Meteo** mediante una aplicación JAVA usando JDBC.

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();  
String url = "jdbc:odbc:Meteo";  
Connection c = DriverManager.getConnection(url);
```

# La Clase "DriverManager"

## Method Summary

static void	<b>deregisterDriver</b> ( <a href="#">Driver</a> driver) Drops a Driver from the DriverManager's list.
static <a href="#">Connection</a>	<b>getConnection</b> ( <a href="#">String</a> url) Attempts to establish a connection to the given database URL.
static <a href="#">Connection</a>	<b>getConnection</b> ( <a href="#">String</a> url, <a href="#">Properties</a> info) Attempts to establish a connection to the given database URL.
static <a href="#">Connection</a>	<b>getConnection</b> ( <a href="#">String</a> url, <a href="#">String</a> user, <a href="#">String</a> password) Attempts to establish a connection to the given database URL.
static <a href="#">Driver</a>	<b>getDriver</b> ( <a href="#">String</a> url) Attempts to locate a driver that understands the given URL.
static <a href="#">Enumeration</a>	<b>getDrivers</b> () Retrieves an Enumeration with all of the currently loaded JDBC drivers to which the current caller has access.
static int	<b>getLoginTimeout</b> () Gets the maximum time that a driver can wait when attempting to log in to a database.
static <a href="#">PrintWriter</a>	<b>getLogWriter</b> () Gets the log writer.
static void	<b>println</b> ( <a href="#">String</a> message) Prints a message to the current JDBC log stream.
static void	<b>registerDriver</b> ( <a href="#">Driver</a> driver) Registers the given driver with the DriverManager.
static void	<b>setLoginTimeout</b> (int seconds) Sets the maximum time in seconds that a driver will wait while attempting to connect to a DB.
static void	<b>setLogWriter</b> ( <a href="#">PrintWriter</a> out) Sets the logging/tracing <a href="#">PrintWriter</a> object that is used by the DriverManager and all drivers.

# La Interface "Connection"

Method Summary	
void	<b><a href="#">close()</a></b> Releases a Connection's database and JDBC resources immediately instead of waiting for them to be automatically released.
void	<b><a href="#">commit()</a></b> Makes all changes made since the previous commit/rollback permanent and releases any database locks currently held by the Connection.
<a href="#">Statement</a>	<b><a href="#">createStatement()</a></b> Creates a <b>Statement</b> object for sending SQL statements to the database.
<a href="#">DatabaseMeta Data</a>	<b><a href="#">getMetaData()</a></b> Gets the metadata regarding this connection's database.
int	<b><a href="#">getTransactionIsolation()</a></b> Gets this Connection's current transaction isolation level.
<a href="#">Map</a>	<b><a href="#">getTypeMap()</a></b> Gets the type map object associated with this connection.
<a href="#">CallableState ment</a>	<b><a href="#">prepareCall(String sql)</a></b> Creates a <b>CallableStatement</b> object for calling database stored procedures.
<a href="#">PreparedState ment</a>	<b><a href="#">prepareStatement(String sql)</a></b> Creates a <b>PreparedStatement</b> object for sending parameterized SQL statements to the database.
void	<b><a href="#">rollback()</a></b> Drops all changes made since the previous commit/rollback and releases any database locks currently held by this Connection.
void	<b><a href="#">setAutoCommit(boolean autoCommit)</a></b> Sets this connection's auto-commit mode.
void	<b><a href="#">setReadOnly(boolean readOnly)</a></b> Puts this connection in read-only mode as a hint to enable database optimizations.
void	<b><a href="#">setTypeMap(Map map)</a></b> Installs the given type map as the type map for this connection.

# La Interfase "Statement"

Para acceder a la base de datos necesitamos poder ejecutar sentencias SQL. Para ello podemos llamar al método `createStatement()` de *Connection* para obtener un objeto que implementa la interfase *Statement*.

## Method Summary

boolean	<code>execute(String sql)</code> Executes an SQL statement, that may return multiple results.
<code>ResultSet</code>	<code>executeQuery(String sql)</code> Executes an SQL statement that returns a single <code>ResultSet</code> object.
int	<code>executeUpdate(String sql)</code> Executes an SQL INSERT, UPDATE or DELETE statement or an SQL statement that returns nothing, such as an SQL DDL statement.

```
Statement stmt=con.createStatement();  
stmt.execute("CREATE TABLE cafes (nombre char(20), precio NUMERIC);");  
stmt.executeUpdate("INSERT INTO cafes VALUES (Colombiano,9);");  
stmt.executeQuery("SELECT * FROM cafes WHERE precio<10;");
```

# Primer Ejemplo. Creando una Tabla Nueva

```
import java.sql.*;
public class CrearTabla{
    public static void main(String args[]) throws Exception {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();
        String url = "jdbc:odbc:Meteo";

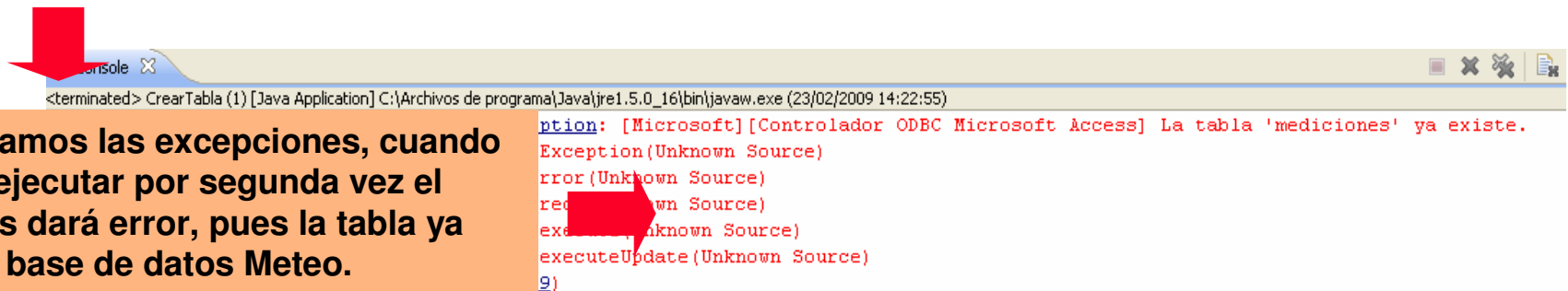
        Connection c= DriverManager.getConnection(url);
        Statement stmt=c.createStatement();
        stmt.executeUpdate("CREATE TABLE mediciones ("+
            "est_id INTEGER NOT NULL,"+
            "med_fecha DATE NOT NULL,"+
            "var_id VARCHAR(8) NOT NULL,"+
            "med_valor FLOAT,"+
            "unidmed_id VARCHAR(8),"+
            "PRIMARY KEY (est_id,med_fecha,var_id));");
        stmt.close();
        c.close();
    }
}
```

- Cargamos el **Driver JDBC-ODBC**.

- Definimos el Origen de Datos, y realizamos la conexión.

- Creamos y Ejecutamos estamentos SQL para crear la Tabla **predicciones** y Añadir elementos a esta.

- Además realizamos una solicitud para obtener todos los elementos de la tabla.



# Excepciones SQL. (SQLException)

```
public class SQLException  
extends Exception
```

An exception that provides information on a database access error or other errors. Each SQLException provides several kinds of information:

- A **string describing the error**. This is used as the Java Exception message, available via the method `getMessage`.
- A **"SQLstate"** string, which follows the XOPEN SQLstate conventions. The values of the SQLState string are described in the XOPEN SQL spec.
- An integer **error code** that is **specific to each vendor**. Normally this will be the actual error code returned by the underlying database.
- A **chain to a next Exception**. This can be used to provide additional error information.

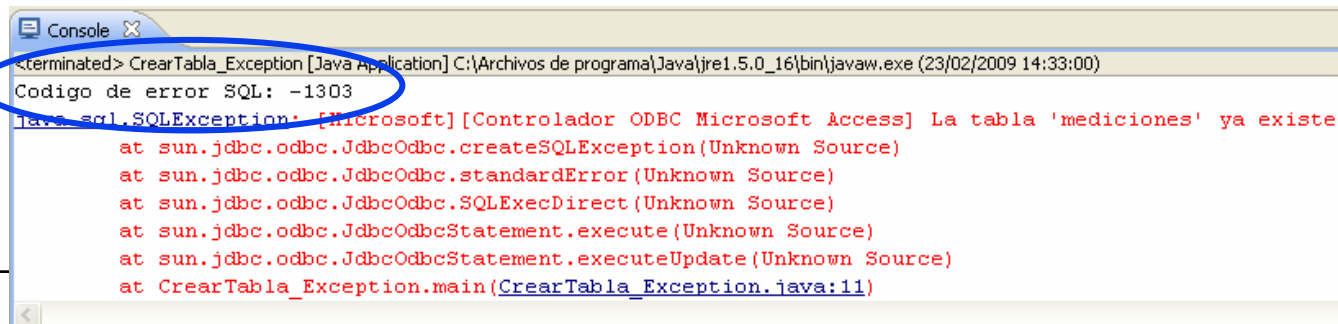
int	<a href="#">getErrorCode()</a> Retrieves the vendor-specific exception code for this SQLException object.
<a href="#">SQLException</a>	<a href="#">getNextException()</a> Retrieves the exception chained to this SQLException object.
<a href="#">String</a>	<a href="#">getSQLState()</a> Retrieves the SQLState for this SQLException object.
void	<a href="#">setNextException(SQLException ex)</a> Adds an SQLException object to the end of the chain.

# Controlando una Excepción SQL ...

```
import java.sql.*;
public class CrearTabla{
    public static void main(String args[]) {

        try{
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();
            String url = "jdbc:odbc:Meteo";
            Connection c= DriverManager.getConnection(url);
            Statement stmt=c.createStatement();
            stmt.executeUpdate("CREATE TABLE mediciones ("+
                "est_id INTEGER NOT NULL,"+
                "med_fecha DATE NOT NULL,"+
                "var_id VARCHAR(8) NOT NULL,"+
                "med_valor FLOAT,"+
                "unidmed_id VARCHAR(8),"
                "PRIMARY KEY (est_id,med_fecha,var_id));");

            stmt.close();
            c.close();
        }catch(SQLException e){
            System.out.println("Codigo de error SQL: "+e.getErrorCode());
            e.printStackTrace();
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```



```
Console X
<terminated> CrearTabla_Exception [Java Application] C:\Archivos de programa\Java\jre1.5.0_16\bin\javaw.exe (23/02/2009 14:33:00)
Codigo de error SQL: -1303
java.sql.SQLException: [Microsoft][Controlador ODBC Microsoft Access] La tabla 'mediciones' ya existe.
    at sun.jdbc.odbc.JdbcOdbc.createSQLException(Unknown Source)
    at sun.jdbc.odbc.JdbcOdbc.standardError(Unknown Source)
    at sun.jdbc.odbc.JdbcOdbc.SQLExecDirect(Unknown Source)
    at sun.jdbc.odbc.JdbcOdbcStatement.execute(Unknown Source)
    at sun.jdbc.odbc.JdbcOdbcStatement.executeUpdate(Unknown Source)
    at CrearTabla_Exception.main(CrearTabla_Exception.java:11)
```

**El código de error específico depende del gestor de base de datos. Para "Access" la existencia de una tabla es -1303**

# Controlando una Excepción SQL

*Podemos modificar el código para que intente crear una tabla y, en caso de que exista, borre la existente y defina la nueva tabla.*

```
import java.sql.*;
public class CrearTabla_Exception2{
    public static void main(String args[]) {
        String creaTabla="CREATE TABLE mediciones ("+
            "est_id INTEGER NOT NULL,"+
            "med_fecha DATE NOT NULL,"+
            "var_id VARCHAR(8) NOT NULL,"+
            "med_valor FLOAT,"+
            "unidmed_id VARCHAR(8),"
            "PRIMARY KEY (est_id,med_fecha,var_id));";

        Connection c;
        Statement stmt;

        try{
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();
            String url = "jdbc:odbc:Meteo";
            c= DriverManager.getConnection(url);
            stmt=c.createStatement();
            try{
                stmt.executeUpdate(creaTabla);
                System.out.println("Creada tabla\n");
            }catch(SQLException e) {
                if (e.getErrorCode()==-1303) {
                    System.out.println("Borrada y creada tabla\n");
                    stmt.executeUpdate("DROP TABLE mediciones");
                    stmt.executeUpdate(creaTabla);
                }
            }
        }catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

# Sentencias SQL Preparadas

```
public interface PreparedStatement  
extends Statement
```

¿Cuándo se utilizar? Cuando se quiera ejecutar varias veces la misma sentencia, ya que al estar precompilada, la ejecución es más rápida.

¿Cómo pasar los parámetros? usando los métodos **setXXX** definidos en la clase **PreparedStatement**. Si el valor que queremos sustituir por una marca de interrogación es un **int** de Java, podemos llamar al método **setInt**. Si el valor que queremos sustituir es un **String** de Java, podemos llamar al método **setString**, etc. En general, hay un método **setXXX** para cada tipo Java.

```
PreparedStatement pstmt = con.prepareStatement("UPDATE EMPLOYEES  
SET SALARY = ? WHERE ID = ?");  
pstmt.setBigDecimal(1, 153833.00);  
pstmt.setInt(2, 110592);  
pstmt.executeUpdate();
```

# Ejemplo. Rellenando Mediciones

```
import java.io.*;import java.sql.*;import java.util.*;
public class RellenaMedidas{
    public static void main(String args[]){
        double data; String str,s; Connection c;
        PreparedStatement pstmt_med;
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();
        }catch(Exception e){
            e.printStackTrace();
        }
        String url = "jdbc:odbc:Meteo";
        try{
            c= DriverManager.getConnection(url);
            pstmt_med = c.prepareStatement("INSERT INTO mediciones"+
            "(est_id,med_fecha,var_id,med_valor,unidmed_id) VALUES (?, ?, ?, ?, ?);");
            BufferedReader br=new BufferedReader(new FileReader("medidas.dat"));
            str=br.readLine();
            while(str!=null ){
                pstmt_med.clearParameters();
                StringTokenizer st=new StringTokenizer(str, ", ");
                s=st.nextToken();
                pstmt_med.setInt(1,Integer.parseInt(s.trim()));
                s=st.nextToken();
                s=s.substring(0,4)+"-"+s.substring(4,6)+"-"+s.substring(6,8);
                pstmt_med.setDate(2,java.sql.Date.valueOf(s));
                pstmt_med.setString(3,"TEMP_MIN");
                pstmt_med.setString(5,"° C");
                s=st.nextToken();
                data=Double.parseDouble(s.trim());
                pstmt_med.setDouble(4,data);
                if(data!=-9999) pstmt_med.executeUpdate();
                str=br.readLine();
            }
            pstmt_med.close();
            c.close()
        }catch(IOException e){
            e.printStackTrace();
        }catch(SQLException e){
            e.printStackTrace();
        }
    }
}
```

- Cargamos el **Driver JDBC-ODBC**. Nos conectamos a nuestro origen de datos.
- Creamos y Ejecutamos **sentencias SQL preparadas** para rellenar la tabla de medidas.
- Abrimos el fichero de texto medidas.dat, del cuál leeremos la información, usando **tokens**
- Asignamos cada uno de los **parámetros** de entrada de la **sentencia preparada**

# Resultado de una consulta

```
public interface ResultSet
```

Conjunto de datos devuelto por un SELECT.

<a href="#">ResultSetMetaData</a>	<a href="#">getMetaData()</a> Retrieves the number, types and properties of this <code>ResultSet</code> object's columns.
boolean	<a href="#">next()</a> Moves the cursor down one row from its current position.

```
public interface ResultSetMetaData
```

Objeto que permite obtener info sobre tipo de dato y propiedades de las columnas de un `ResultSet`

int	<a href="#">getColumnCount()</a> Returns the number of columns in this <code>ResultSet</code> object.
<a href="#">String</a>	<a href="#">getColumnName(int column)</a> Get the designated column's name.
int	<a href="#">getColumnType(int column)</a> Retrieves the designated column's SQL type.
<a href="#">String</a>	<a href="#">getColumnTypeName(int column)</a> Retrieves the designated column's database-specific type name.

# Ejemplo. Resultado de una consulta

```
import java.sql.*;
public class ViewData{
public static void main(String[] args) {
    try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();
        String url = "jdbc:odbc:Meteo";
        Connection c = DriverManager.getConnection(url, "", "");
        Statement st = c.createStatement
            (ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
        ResultSet rs = st.executeQuery("SELECT est_id,est_nombre FROM
            estaciones WHERE pais_id='SPAIN'");

        ResultSetMetaData md = rs.getMetaData();
        while(rs.next()) {
            System.out.print("\nEstación: | ");
            for(int i=1; i<= md.getColumnCount(); i++)
                System.out.print(rs.getString(i) + " | ");
        }
    } catch(Exception e) {
        e.printStackTrace();
    }
}}
```

# La Interfase "Statement", parámetros para consultas

<u>Statement</u>	<u>createStatement()</u>  Creates a <code>Statement</code> object for sending SQL statements to the database.
<u>Statement</u>	<u>createStatement(int resultSetType, int resultSetConcurrency)</u>  Creates a <code>Statement</code> object that will generate <code>ResultSet</code> objects with the given type and concurrency.
<u>Statement</u>	<u>createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability)</u>  Creates a <code>Statement</code> object that will generate <code>ResultSet</code> objects with the given type, concurrency, and holdability.

## Parámetros:

`resultSetType`: `ResultSet.TYPE_FORWARD_ONLY`, `ResultSet.TYPE_SCROLL_INSENSITIVE`,  
or `ResultSet.TYPE_SCROLL_SENSITIVE`

`resultSetConcurrency` : `ResultSet.CONCUR_READ_ONLY` or  
`ResultSet.CONCUR_UPDATABLE`

`resultSetHoldability` : `ResultSet.HOLD_CURSORS_OVER_COMMIT` or  
`ResultSet.CLOSE_CURSORS_AT_COMMIT`

# Conversión de Tipos

## Common SQL Types--Standard Retrieval Methods

SQL Type	Java Method
BIGINT	getLong()
BINARY	getBytes()
BIT	getBoolean()
CHAR	getString()
DATE	getDate()
DECIMAL	getBigDecimal()
DOUBLE	getDouble()
FLOAT	getDouble()
INTEGER	getInt()
LONGVARBINARY	getBytes()
LONGVARCHAR	getString()
NUMERIC	getBigDecimal()
OTHER	getObject()
REAL	getFloat()
SMALLINT	getShort()
TIME	getTime()
TIMESTAMP	getTimestamp()
TINYINT	getByte()
VARBINARY	getBytes()
VARCHAR	getString()

## SQL3 Types--Retrieval Methods

SQL Type	Java Method
ARRAY	getArray()
BLOB	getBlob()
CLOB	getClob()
DISTINCT	getType()
REF	getRef()
STRUCT	(cast)getObject()
JAVA OBJECT	(cast)getObject()

# JDBC y SQL Server 2005

- ◆ Cargar el Driver de la base de datos (tipo 4) . Tiene su propio conjunto de clases (<http://www.microsoft.com/spain/sql/technologies/jdbc/default.mspx> del SQL 2005)
- ◆ Después especificar driver para el SQL 2005

```
Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver").newInstance();  
String url = "jdbc:sqlserver://localhost;databaseName=Cine;  
            user=MyUserName; password=****";  
Connection c = DriverManager.getConnection(url);
```

- ◆ Si usas SQL 2005 fíjate que acepte transacciones TCP/IP. Este es un error común ya que el servidor lo trae deshabilitado por defecto. Además se necesita que el modo de conexión sea mixto

# JDBC y Oracle

- ◆ Cargar el Driver de la base de datos  
([http://www.oracle.com/technology/software/tech/java/sqlj\\_jdbc/index.html](http://www.oracle.com/technology/software/tech/java/sqlj_jdbc/index.html))
- ◆ Después especificar driver para el Oracle

```
Class.forName("oracle.jdbc.driver.OracleDriver");
String serverName = "127.0.0.1";
String portNumber = "1521";
String sid = "mydatabase";
String url = "jdbc:oracle:thin:@" + serverName + ":" + portNumber + ":" + sid;
String username = "username";
String password = "password";
connection = DriverManager.getConnection(url, username, password);
```

# Ejecución de procedimientos almacenados

```
public interface CallableStatement  
extends PreparedStatement
```

Interfaz para ejecutar procedimientos almacenados

void	<b>registerOutParameter</b> (int parameterIndex, int sqlType) Registers the OUT parameter in ordinal position parameterIndex to the JDBC type sqlType.
boolean	<b>wasNull()</b> Retrieves whether the last OUT parameter read had the value of SQL NULL.
int	<b>getInt</b> (int parameterIndex) Retrieves the value of the designated JDBC INTEGER parameter as an int in the Java programming language.
void	<b>setInt</b> (String parameterName, int x) Sets the designated parameter to the given Java int value.

# Ejemplo. Ejecución procedimiento con parámetros de entrada y devolución de cjto de resultados

```
import java.sql.*;
public class ViewDataProc{
public static void main(String[] args) {
try {
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();
String url = "jdbc:odbc:Cine";
Connection c = DriverManager.getConnection(url, "", "");
int idPer=2; //Id de la persona
try
{ CallableStatement stmt
= c.prepareCall("{ call dbo.PersonaPremios (?) }");
stmt.setInt (1,idPer);
ResultSet rs = stmt.executeQuery();
while (rs.next())
{ String texto = rs.getString("texto");
System.out.println( " Salida: " + texto);
}
rs.close();
stmt.close();
}
catch (SQLException e)
{ e.printStackTrace(); }
} catch (Exception e)
{ e.printStackTrace(); }
}
```

```
CREATE PROCEDURE [dbo].[PersonaPremios]
@idPersona int AS
SELECT Premios.Nombre + ' por ' +
Peliculas.TituloOriginal as texto
FROM Nominaciones N INNER JOIN Peliculas P
ON N.IdPelicula = P.IdPelicula INNER JOIN
Premios Pr ON N.IdPremio = Pr.IdPremio INNER
JOIN Personas Pe ON N.IdPer = Pe.IdPer WHERE
(N.FineDN = 1) AND (N.IdPer = @idPersona)
```

## Ejemplo. Ejecución procedimiento con parámetros de entrada y salida

```
public static void executeStoredProcedure(Connection con) {
    try {
        CallableStatement cstmt = con.prepareCall("{call
dbo.GetImmediateManager(?, ?)}");
        cstmt.setInt(1, 5);
        cstmt.registerOutParameter(2, java.sql.Types.INTEGER);
        cstmt.execute();
        System.out.println("MANAGER ID: " + cstmt.getInt(2));
    }
    catch (Exception e) {
        e.printStackTrace();
    }
}
```

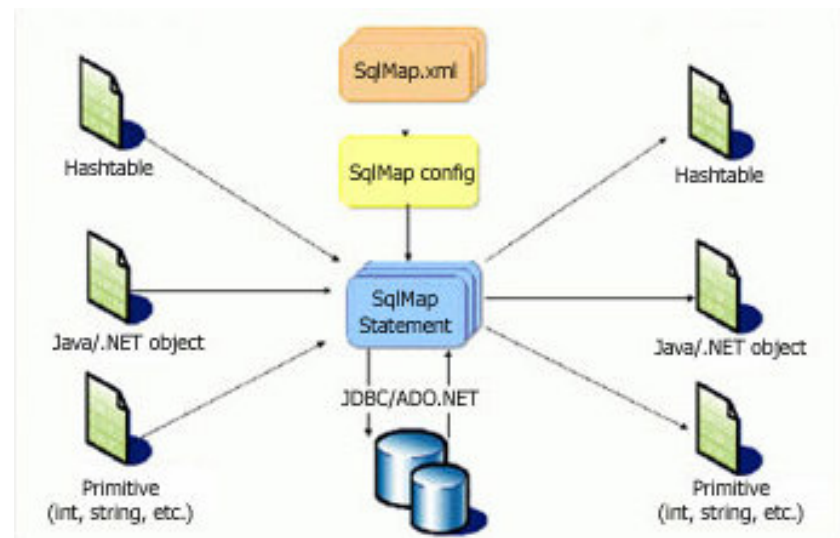
```
CREATE PROCEDURE GetImmediateManager @employeeID INT,
                                     @managerID INT OUTPUT AS
BEGIN
    SELECT @managerID = ManagerID FROM HumanResources.Employee WHERE
    EmployeeID = @employeeID
END
```

## Ejemplo. Uso de transacciones

```
import java.sql.*;
public class EjecutarTransacciones{
    public static void main(String[] args) {
        try {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();
            String url = "jdbc:odbc:Cine";
            Connection con = DriverManager.getConnection(url, "", "");
            try {
                //Switch to manual transaction mode by setting autocommit to false and starts manual tran.
                con.setAutoCommit(false);
                Statement stmt = con.createStatement();
                stmt.executeUpdate("INSERT INTO tribunal(Idper,idfest,numedic)
                    VALUES(1,2,56)");
                stmt.executeUpdate("INSERT INTO tribunal(Idper,idfest,numedic)
                    VALUES(1,1,80)");
                con.commit(); //it commits the transaction and starts a new one.
                stmt.close(); //This turns off the transaction.
                System.out.println("Transaction succeeded.Both records were written to DB.");
            }
            catch (SQLException ex) {
                ex.printStackTrace();
                try {
                    con.rollback();
                    System.out.println("Transaction failed.No records were written to DB.");
                }
                catch (SQLException se) {
                    se.printStackTrace();
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

# iBatis

- ◆ **iBatis** es un framework que facilita el diseño de la capa de persistencia utilizada en las aplicaciones Java para acceder a nuestro repositorio de datos.
- ◆ iBatis une los *Objetos* con las *sentencias SQL* mediante un *descriptor XML*.
- ◆ iBatis necesita uno o más ficheros con las sentencias SQL que usará el programa
- ◆ iBatis necesita un fichero de configuración en XML donde se indiquen los parámetros de conexión a la base de datos y los ficheros de mapeo XML



# Hibernate

- ◆ **Hibernate** es una herramienta de Mapeo objeto-relacional para la plataforma Java (y disponible también para .Net con el nombre de NHibernate) que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) que permiten establecer estas relaciones.
- ◆ Hibernate no requiere conocer el modelo de datos al cual se accede. Se tiene una aplicación Orientada a objetos y se mapea el diseño OO en una capa de persistencia.
- ◆ La mayor diferencia entre Hibernate e iBATIS proviene del hecho de que el último basa su funcionamiento en el mapeo de sentencias SQL que se incluyen en ficheros XML. Eso significa que, al contrario que Hibernate, requiere conocimiento de SQL por parte del programador. Por otra parte, permite la optimización de las consultas, ya sea con lenguaje estándar o con SQL propietario del motor de base de datos utilizado. Con iBATIS, siempre se sabe lo que se está ejecutando en la base de datos y permite generar consultas dinámicas muy potentes.

# Referencias

## ◆ Referencias

- Tutorial JDBC: <http://java.sun.com/docs/books/tutorial/jdbc/index.html>
- SQL Server y JDBC  
<http://msdn.microsoft.com/es-es/library/bb418503.aspx>
- Oracle y JDBC  
<http://w2.syronex.com/jmr/edu/db/oracle-and-java>  
[http://www.oracle.com/technology/tech/java/sqlj\\_jdbc/htdocs/jdbc\\_faq.html](http://www.oracle.com/technology/tech/java/sqlj_jdbc/htdocs/jdbc_faq.html)
- iBatis  
<http://ibatis.apache.org/>
- Hibernate  
<http://www.hibernate.org/>

