



TEMA 3: ACCESO A BBDD

Introducción ODBC - JDBC

ODBC → Open DataBase Connectivity [estándar de acceso a BBDD]

JDBC → Java DataBase Connectivity → API desde java

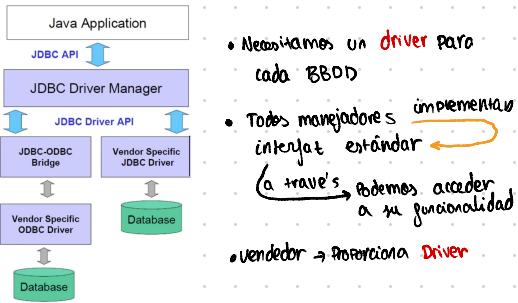
Paquetes: `java.sql.*`, `javax.sql.*`

objetivos: Establecer conexión con una BBDD

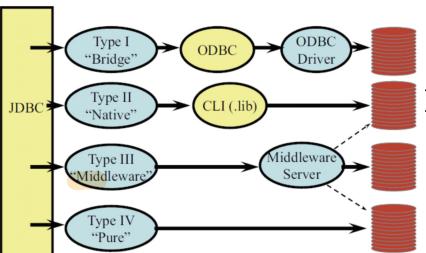
Enviar consultas (SQL) a BBDD

Procesamiento de los resultados de una consulta

Cada BBDD implementa su propio Protocolo binario para comunicación a través de la red



4 tipos de driver:



Ventajas: ya están donde ya están 3 controladores ODBC

Inconvenientes: no multiplataforma, bajo rendimiento

Ventajas: mejor rendimiento q el anterior

Inconvenientes: no escalabilidad, no multiplataforma

Ventajas: flexibilidad

Inconvenientes: se necesita servidor intermedio + config

Ventajas: mejor rendimiento, 100% portable

Inconvenientes: no aprovecha carac. del SO

usando la API JDBC

1) Cargar el driver

Ligarlo en memoria + tener driver instalado

↳ Solución → driver: `org.sqlite.JDBC`

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

2) Definir la URL de conexión

Protocolo: subprotocolo://servidor:puerto/subnombre

→ Protocolo → jdbc

→ Subprotocolo → indica mecanismo conexión a BBDD
Permite al **Driver Manager** saber q tipo de driver utilizar

→ Servidor → identifica máquina donde está BBDD

→ Puerto → indica puerto conexión a máquina

→ Subnombre → identifica BBDD concreta

```
String host = "dbhost.yoursite.com";
String dbName = "clientes";
int port = 1234;
String mysqlURL = "jdbc:mysql://" + host + ":" + port + "/" + dbName;

String otraURL = "jdbc:sqlite:/nopath/file.sqlite";
en sqlite → jdbc:sqlite:nopath/file.sqlite
```

3) Establecer conexión con la BD

↳ Se utiliza clase **DriverManager**

→ Generar las URLs utilizadas
abrir solo una conexión abierta y reutilizarla no saturar el servidor

4) Métodos más imp:

[devuelven obj Connection → representan conexión]

```
static Connection getConnection(String url)
static Connection getConnection(String url, String user, String password);

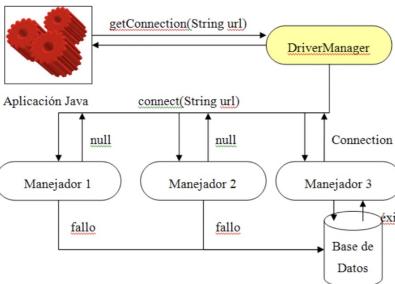
String user = "usuario";
String password = "pass";
DriverManager.getConnection(mysqlURL, user, password);

DriverManager.getConnection(otraURL, "usuario", "pass");
```

• **Driver Manager** invoca `connect()` de la interface Driver, devuelve conexión correspondiente

• Va probando todos drivers registrados hasta q uno establece conexión

sino establece conexión → **SQLException**



TEMA 3: ACCESO A BBDD

(USANDO LA API JDBC)

4) PREPARAR LA CONSULTA (CON OBJ Statement)

Permiten ejecutar comandos SQL contra la BD

```
Statement myStatement = conn.createStatement();
```

callable Statement: permite ejecución de instrucciones no-SQL

PreparedStatement: instrucción SQL → Permite ejecución más eficiente

- Necesaria para sentencias SQL q involucran datos no básicos (enteros, reales y cadena caracteres)
- seguridad

CONSULTAS PARAMETRIZADAS

Consulta

```
String sqlString = "SELECT * FROM tracks WHERE name LIKE ?";  
PreparedStatement prep = conn.prepareStatement(sqlString);  
prep.setString(1, "%love%");
```

INserción

```
String sqlString = "INSERT INTO empleado (nombre, telefono, salario, fechaContrato, foto)"  
+ "VALUES (?, ?, ?, ?, ?);"  
PreparedStatement prep = conn.prepareStatement(sqlString);  
prep.setString(1, "Lucía Sánchez");  
prep.setInt(2, 33333333);  
prep.setDouble(3, 2500.30);  
prep.setDate(4, unObjetoSQLDate);  
prep.setBytes(5, unArrayBytes);
```

5) EJECUTAR LA CONSULTA (Query)

```
ResultSet result = myStatement.executeQuery("SELECT * FROM Clientes");  
  
int result2 = myStatement.executeUpdate("UPDATE Clientes SET codigoProvincia=28 WHERE provincia='Madrid'");
```

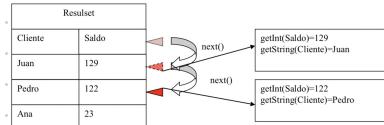
Statement

```
ResultSet result = prep.executeQuery();  
prep.executeUpdate();
```

PreparedStatement

6) PROCESAR LOS RESULTADOS (ResultSet)

- Esta interfaz representa conjunto de datos del resultado de una consulta SQL
- Posse métodos → Permiten acceder a los datos de la consulta
 - get (int columna)
 - get (String columna)
- Acceder a registros utilizamos cursor
 - initialmente apunta justo antes de la 1^a fila, column 1, 2, 3, ... No 0, 1, 2, ...
- Desplazar cursor → next()



```
int id;  
String nombre;  
while(result.next()) {  
    id = result.getInt("Id");  
    nombre = result.getString("Nombre");  
    System.out.println("Cliente " + id + " -- Nombre: " + nombre);  
}
```

```
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.sql.Statement;  
  
public class Ejemplo1 {  
  
    public static void main(String args[]) {  
        String driver = "com.mysql.jdbc.Driver";  
        String url = "jdbc:mysql://localhost:3306/amigos";  
        String usuario = "root";  
        String clave = "rootPass";  
  
        Statement statement = null;  
        Connection conexion = null;  
        ResultSet resultados = null;  
  
        try {  
            Class.forName(driver);  
            conexion = DriverManager.getConnection(url, usuario, clave);  
            statement = conexion.createStatement();  
  
            statement.executeUpdate("CREATE TABLE usuarios (nombre VARCHAR(25), login VARCHAR(15), edad INT, nivelParticipacion FLOAT)");  
  
            statement.executeUpdate("INSERT INTO usuarios VALUES ('Pepe', 'pepe', 25, 0.64)");  
            statement.executeUpdate("INSERT INTO usuarios VALUES ('Juan', 'juan', 20, 0.23)");  
            statement.executeUpdate("INSERT INTO usuarios VALUES ('Maria', 'mar', 22, 0.84)");  
            statement.executeUpdate("INSERT INTO usuarios VALUES ('Natalia', 'Natali', 35, 0.33)");  
            statement.executeUpdate("INSERT INTO usuarios VALUES ('Raúl', 'paco', 42, 0.22)");  
  
            resultados = statement.executeQuery("SELECT NOMBRE FROM usuarios WHERE nivelParticipacion > 0.5");  
  
            while (resultados.next()) {  
                System.out.println(resultados.getString("NOMBRE")  
                    + " gracias por ser un usuario tan activo");  
            }  
        } catch (Exception e) {  
            e.printStackTrace();  
        } finally {  
        }
```

7) CERRAR LA CONEXIÓN

- Mantener abiertas conexiones es costoso
- Conviene cerrar cuando no se vaya a utilizar más `conexion.close();`

Algunos métodos de java.sql.Connection

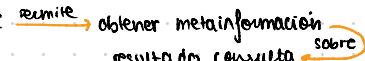
Método	Funcionalidad
Statement createStatement()	Devuelve un objeto del tipo Statement que se utiliza para ejecutar las consultas SQL a la base de datos
PreparedStatement prepareStatement(String sql)	Crea un objeto del tipo PreparedStatement, que permite parametrizar una sentencia SQL
DatabaseMetaData getMetaData()	Obtiene un objeto con (meta) información sobre la configuración de la base de datos
void commit()	Hace permanentes todos los cambios hechos después del último commit/rollback
void rollback()	Deshace todos los cambios hechos después del último commit/rollback
void setAutoCommit(boolean yn)	Establece si se efectuarán commit automáticos después de cada comando SQL. Por defecto, AutoCommit = true
boolean getAutoCommit()	Devuelve si está activado el AutoCommit
void close()	Cierra la conexión con la base de datos
boolean isClosed()	Devuelve true si la conexión con la base de datos está cerrada

Driver: jar meteo → DatabaseDriver → Properties
! → ModulePath y classpath

Query → devuelve
Update → no devuelve nada

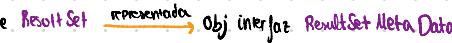
Tema 3: Acceso a BBDD

Metainformación

- API JDBC  obtiene metainformación resultado consulta sobre
- Metainfo. de la BD  representada Obj. interfaz java.sql.DatabaseMetaData

Algunos métodos:

- ↳ getDriverName()
- ↳ getDatabaseProductName() [Nombre BD]
- ↳ getDriverVersion()
- ↳ getURL()

- La metainfo. de ResultSet  representada Obj. interfaz ResultSetMetaData

accesible mediante getMetaData() → de la interfaz ResultSet

Métodos (lomo)

- ↳ getColumnCount() [Nº columnas del ResultSet]
- ↳ getColumnLabel(int columna)
- ↳ isWritable(int columna)
- ↳ getColumnClassName(int columna)

```
run:  
MySQL  
5.6.21  
MySQL Connector Java  
mysql-connector-java-5.1.38 ( Revision: fe541c166cec739c74cc727c5da96c1028b4834a )  
jdbc:mysql://localhost:3306/amigos  
mtp@localhost  
| nombre (VARCHAR) | login (VARCHAR) | edad (INT) | nivelParticipacion (FLOAT)  
-----  
| Pepe | pepe | 23 | 0.64  
-----  
| Juan | juan | 38 | 0.23  
-----  
| Antonio | anton | 28 | 0.82  
-----  
| Maria | mar | 22 | 0.84  
-----  
| Natalia | Nati | 35 | 0.33  
-----  
| Paco | paco | 42 | 0.22  
-----  
  
BUILD SUCCESSFUL (total time: 1 second)
```

ORMs (Object-Relational Mapping)

- Frameworks de persistencia para operaciones CRUD [Create, Read, Update, Delete] en una BD
- Mapado entre objetos y BD relacional
- objetivo/Ventaja → liberar desarrollador escribir código SQL
- Problema → menos rendimiento

? JPA (Java Persistence API) 