Programación

Unidad 12: Conexión con bases de datos

Unidad 12

- Persistencia de datos
- Introducción a JDBC
- Componentes JDBC
- Drivers
- Conexión a la base de datos
- Interface Statement y Clase PreparedStatement
- ResultSet

Persistencia de datos

Unidad 12: Conexión con bases de datos

Persistencia de datos

Definición: La continuación de un efecto después de que desaparezca su causa. Por ejemplo: las palabras escritas en un papel persisten después de haberlas escrito

En **Computación:** La persistencia es la característica de los datos que **sobreviven** a la ejecución del programa que los ha creado

En **Programación Orientada a Objetos:** La persistencia es la capacidad de los objetos de **existir más allá del ciclo de vida** del programa en el que fueron creados

Introducción a JDBC

Unidad 12: Conexión con bases de datos

Introducción a JDBC

JDBC es el estándar para conseguir la conectividad entre bases de datos y Java

JDBC nos proporciona una API que da a la aplicación la capacidad de:

- Conectarse a las bases de datos
- Enviar sentencias SQL
- Procesar los resultados

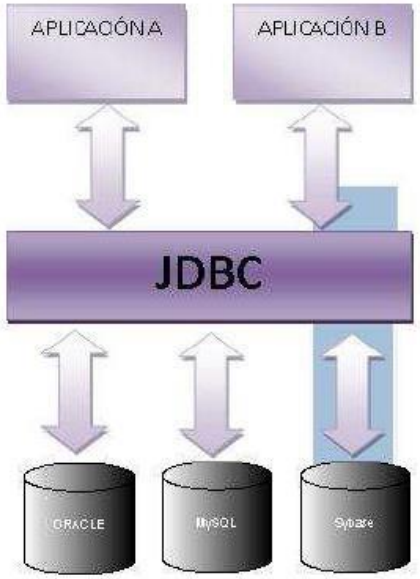
La conexión del código Java a la BD la realizaremos mediante la interfaz de programación de aplicaciones llamada JDBC (**Java Database Connectivity**) e incluye un conjunto de herramientas que permiten la ejecución de operaciones sobre DB desde Java.

A partir de ahí se puede realizar en la base de datos cualquier tipo de tareas sobre las que tenga permiso: consultas, actualizaciones, manipulación de tablas, etc.

Unidad 12: Conexión con bases de datos

Introducción a JDBC

Con JDBC abstraemos a las aplicaciones del SGBD que se use. JDBC proporciona una librería para acceder a distintas bases relacionales y “normaliza” la mayor parte de las operaciones (las hace independientes de la base de datos utilizada y por tanto portables). Es parte de las distribuciones estándar de Java.



JDBC no comprueba que las sentencias SQL sean correctas, sencillamente las envía a la base de datos.

Unidad 12: Conexión con bases de datos

Introducción a JDBC

JDBC consta de:

Un API genérico (escrito en java)

http://docs.oracle.com/javase/23/docs/api/java/sql/package-summary.html

Drivers específicos de cada base de datos que se encargan de las interacciones con la base de datos especificas de cada sistema que “habla" a la base con la base de datos.

Tanto el API como los drivers **se ejecutan en el cliente**.

Componentes JDBC

Unidad 12: Conexión con bases de datos

Componentes JDBC

Gestor de los Drivers

`java.sql.DriverManager`

Conexión con la BD

`java.sql.Connection`

Sentencia a ejecutar

`java.sql.Statement`

`java.sql.PreparedStatement`

Resultado

`java.sql.ResultSet`

Unidad 12: Conexión con bases de datos

Componentes JDBC

Pasos para programar con JDBC:

- Cargar el **driver** de la base de datos
- Definir la **URL de la conexión**
- Establecer la **conexión**
- **Crear** una sentencia **SQL**
- **Ejecutar** la sentencia **SQL**
- **Procesar** los resultados
- **Cerrar** la conexión

Drivers

Unidad 12: Conexión con bases de datos

Drivers

Para cargar el driver JDBC de la base de datos que queremos acceder utilizamos la clase **Driver**

```
Class.forName(nombreDriver)
```

- Son facilitados por el fabricante
  - MySQL **nombreDriver=com.mysql.cj.jdbc.Driver**
  - MySQL https://dev.mysql.com/downloads/connector/j/
  - PostgreSQL https://jdbc.postgresql.org/

- Ejemplo:

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

Conexión a la base de datos

Unidad 12: Conexión con bases de datos

Conexión a la base de datos

Una vez que tenemos el driver cargado en memoria, creamos la conexión

Objeto clase Connection

```
Connection nombre;
nombre=DriverManager.getConnection(urlbd,usuario,clave);
```

urlbd ➡ driver://maquina/bd

¡Cuidado! Las consultas SQL solo se podrán enviar cuanto tengamos una conexión

Unidad 12: Conexión con bases de datos

Conexión a la base de datos - Ejemplo

```
Connection  connection = DriverManager.getConnection( url: "jdbc:mysql://localhost/biblioteca?"+
        "useSSL=false&zeroDateTimeBehavior=CONVERT_TO_NULL&serverTimezone=UTC"+
        "&allowPublicKeyRetrieval=true&user=usuario&password=clavesecreta!");
```

A partir de ahora, podremos comunicarnos con la BD enviándoles sentencias SQL

Una vez que hayamos terminado de trabajar con una conexión deberemos **liberarla**.

```
nombreConexion.close();
```

Interface Statement y clase PreparedStatement

Unidad 12: Conexión con bases de datos

Interface Statement y clase PreparedStatement

Existen dos tipos de **statements** (que será la forma de comunicarnos con la base de datos)

`java.sql.Statement`

`java.sql.PreparedStatement`

Interface Statement y clase PreparedStatement

Interface Statement

Se usa para ejecutar una sentencia SQL contra una BD. Siempre lleva asociada una conexión que sirvió como origen para su creación.

Se crean con el siguiente método, de la clase java. sql.Connection

```
Statement identif = objetoConnection.createStatement();
```

Se cierran mediante el método **`close()`**

No hace falta crear un objeto statement para cada consulta

Unidad 12: Conexión con bases de datos

Interface Statement y clase PreparedStatement

El método para ejecutarla depende del tipo de sentencia SQL que contenga.

Sentencias **SELECT**:

Se usa el método **`executeQuery(String sql)`**

Devuelve una instancia de java.sql.ResultSet

Sentencias **INSERT, UPDATE y DELETE**:

Se usa el método **`executeUpdate(String sql)`**

Devuelve un int con el número de filas afectadas

Unidad 12: Conexión con bases de datos

Interface Statement y clase PreparedStatement

Sentencias **CREATE TABLE y DROP TABLE**:

Se usa el método **`executeUpdate(String sql)`**

Devuelve un int que siempre vale 0

```
public static Connection conectarBD() throws ClassNotFoundException, SQLException{

    conn = null;
        System.out.println("Conectando...");
            Class.forName( className: "com.mysql.cj.jdbc.Driver");
            Connection conn =
                DriverManager.getConnection( url: "jdbc:mysql://localhost/biblioteca?" +
                                        "useSSL=false&zeroDateTimeBehavior=CONVERT_TO_NULL&serverTimezone=UTC&"+
                                         "user=root&password=mipass");
            if (conn==null)
                System.out.println("No hay conexión");
    return conn;
}
```

Unidad 12: Conexión con bases de datos

Interface Statement y clase PreparedStatement

```
public static Connection conectarBD() throws ClassNotFoundException, SQLException{

    conn = null;
        System.out.println("Conectando...");
           Class.forName( className: "com.mysql.cj.jdbc.Driver");
           Connection conn =
              DriverManager.getConnection( url: "jdbc:mysql://localhost/biblioteca?" +
                                          "useSSL=false&zeroDateTimeBehavior=CONVERT_TO_NULL&serverTimezone=UTC&"+
                                           "user=root&password=mipass");
           if (conn==null)
               System.out.println("No hay conexión");
    return conn;
}
```

```
public static ResultSet consultaBD(String sql) throws SQLException{
    Statement stmt=conn.createStatement();
    return stmt.executeQuery(sql);
}
```

Unidad 12: Conexión con bases de datos

Interface Statement y clase PreparedStatement

```
public List<Usuario> mostrar() throws SQLException{
    String cadenaSQL = "SELECT * FROM usuarios";
    ResultSet rs;
    List<Usuario> lista = new ArrayList<>();
    rs=AccesoBD.consultaBD(cadenaSQL);
    while (rs.next()){
        Usuario nuevo = new Usuario();
        nuevo.setId(rs.getInt( columnLabel: "idusuarios"));
        nuevo.setLogin(rs.getString( columnLabel: "nombre"));
        nuevo.setClave(rs.getString( columnLabel: "clave"));
        lista.add(nuevo);
    }
    rs.close();
    return lista;
}
```

Unidad 12: Conexión con bases de datos

Interface Statement y clase PreparedStatement

```
List<Usuario> listaUsuarios;
UsuarioDAO usrDAO = new UsuarioDAO();
try {
    //Intentamos conectar desde el método estático conectarBD()
    if ((miconn == null) || (miconn.isClosed()))
        miconn = AccesoBD.conectarBD();
    System.out.println("CONECTADO!!!");
    if ((miconn != null) && (!miconn.isClosed())) {
        //Obtenemos la lista de usuarios que está en la tabla
        //usuarios de la base de datos
        listaUsuarios = usrDAO.mostrar();
        System.out.println("MOSTRAR!!!");
        //Recorremos la lista para mostrar los datos recuperados de la base de datos
        for (Usuario listaUsuario : listaUsuarios) {
            System.out.println(listaUsuario);
        }
    }
} catch (SQLException ex) {
    System.out.println("Error en SQL");
} catch (ClassNotFoundException ex) {
    System.out.println("Clase no encontrada");
} finally {
    try {
        if ((miconn != null) && (!miconn.isClosed())) {
            //Cerramos la conexión con la base de datos
            AccesoBD.cerrarBD();
            System.out.println("DESCONECTADO!!!");
        }
    } catch (SQLException ex) {
        System.out.println("Error en SQL");
    }
}
```

Unidad 12: Conexión con bases de datos

Interface Statement y clase PreparedStatement

Clase PreparedStatement

Si se van a ejecutar varias consultas similares el uso de consultas parametrizadas puede ser más eficiente. PreparedStatement se usa para ejecutar estas **sentencias parametrizadas**. Se diferencia de java sql Statement en dos aspectos:

- es compilada por el SGBD previamente
- la sentencia puede contener variables marcadas con ?

Se crean con el siguiente método de la clase java.sql.Connection

```
PreparedStatement ps = objetoConnection.prepareStatement
```

Unidad 12: Conexión con bases de datos

Interface Statement y clase PreparedStatement

Una **PreparedStatement** es útil cuando se va a ejecutar una sentencia SQL **muchas veces**. Una **PreparedStatement** es **precompilada** por lo que es ejecutada inmediatamente por la base de datos.

Una Statement **debe ser compilada**.

Para ejecutar:

`executeQuery()` y **`executeUpdate()`**

En el caso de que la sentencia contenga alguna variable se le deberá dar un valor antes de ejecutarla:

```
public void setXxxx(int index,xxxx value)
```

Las sentencias se cierran mediante el método

```
public void close()
```

Unidad 12: Conexión con bases de datos

Interface Statement y clase PreparedStatement

Ejemplo:

```
…
PreparedStatement pstmt=con.prepareStatement("SELECT * FROM Empleados
   WHERE trabajo=? AND level=?");
……
pstm.setString(1,"MANAGER");
pstm.setInt(2,16);
ResultSet rs=pstmt.executeQuery();
……
rs.close();
pstmt.close();
con.close();
……
```

ResultSet

Unidad 12: Conexión con bases de datos

ResultSet

Un ResultSet es el resultado de ejecutar un SELECT. Lleva asociadas todas las filas que cumplen las condiciones de la sentencia SQL.

Implementa métodos para:

- Acceder a las filas que componen el resultado.
- Acceder al valor de cada columna de la fila seleccionada.

Unidad 12: Conexión con bases de datos

ResultSet

Método para acceder a la fila siguiente del resultado:

```
public boolean next()
```

Métodos para acceder al valor de una columna, dependerá del tipo de dato:

- `public xxxx getXxxx(int col)`
- `public xxxx getXxxx(String col)`

Los ResultSet se cierran mediante el método **`close()`**

Unidad 12: Conexión con bases de datos

ResultSet

```
……
    ResultSet rs=stmt.executeQuery("SELECT * FROM empleados");
    while(rs.next()){
                System.out.println("Nombre: "+rs.getString(2)+
                "Nivel: "+rs.getInt("nivel"));
    }
    rs.close();
……
```