UT9

ACCESO A BASES DE DATOS RELACIONALES

1. INTRODUCCIÓN

- Java tiene una API que permite interactuar con fuentes de datos de manera que podemos:
  - Conectarnos a una base de datos (BD)
  - Enviar consultas de selección, inserción y actualización de la BD
  - Recuperar datos de una consulta y manejarlos

- El acceso a bases de datos desde Java se realiza mediante el estándar **JDBC**, que permite un acceso a las BD **independientemente del SGBD**.

1. INTRODUCCIÓN

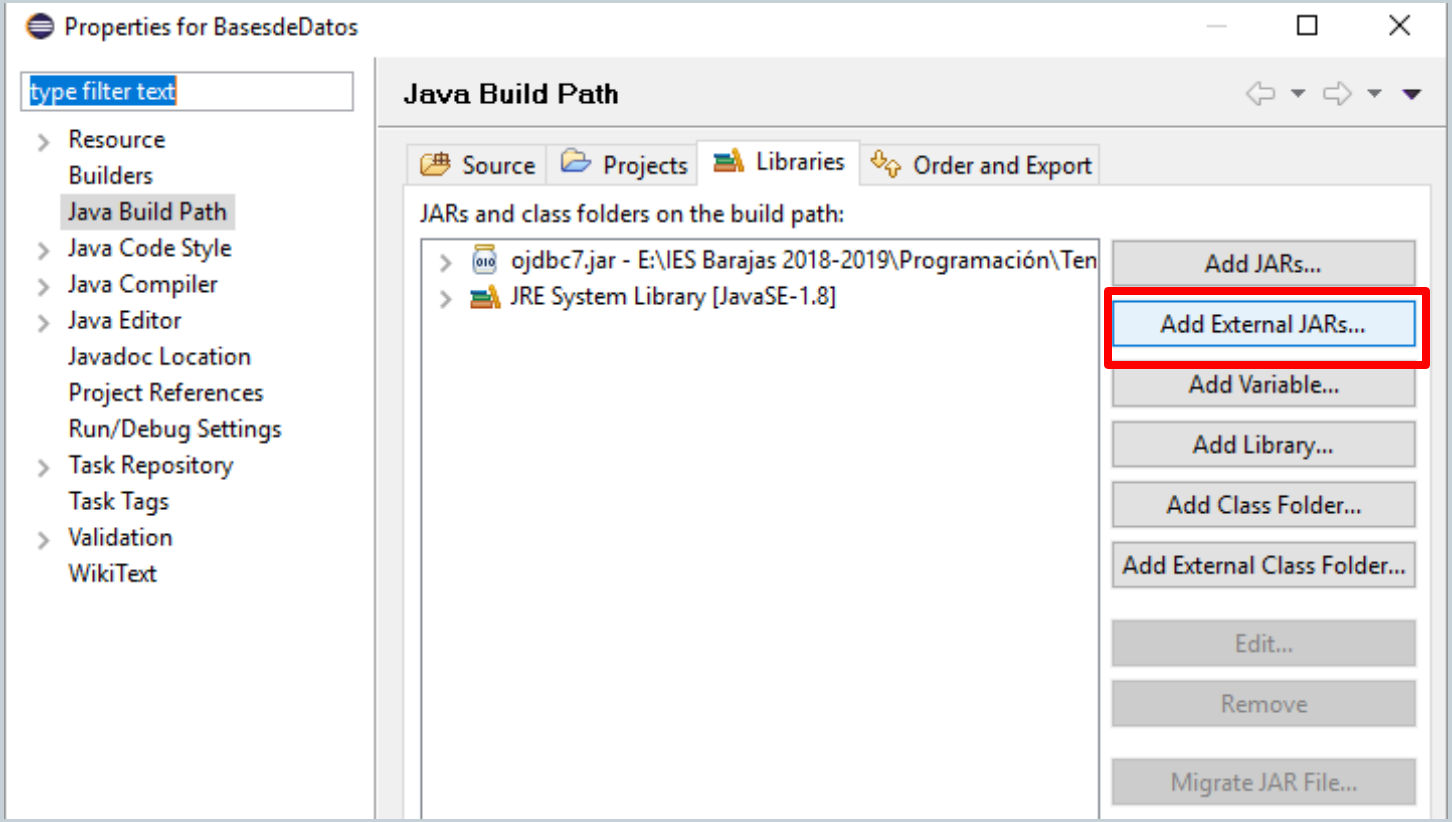
- ***Java Database Connectivity (JDBC)***, es una API que permite la ejecución de operaciones de BD desde el lenguaje de programación Java
  - Es independiente del sistema operativo
  - Es independiente del SGBD
  - Utiliza el dialecto SQL del modelo de base de datos que se utilice.
- Las aplicaciones escritas en Java no necesitan conocer las especificaciones de un SGBD en particular, basta con comprender el funcionamiento de JDBC.
- Cada SGBD que quiera utilizarse con JDBC debe contar con un adaptador o **controlador**.

2. Conexión desde JAVA

- Es necesario descargar el **driver JDBC específico** para conectar Java con el SGBD, *en nuestro caso Oracle*.
- Oracle provee de forma libre dicho driver: https://www.oracle.com/technetwork/database/features/jdbc/default-2280470.html
- El primer paso para conectarnos a una base de datos desde nuestro proyecto JAVA es incorporar el driver JDBC al classpath del mismo.
- El fichero jar es necesario para ejecutar la aplicación
  - Para utilizar nuestra app **el jar de JDBC debe estar disponible**

2. Conexión desde JAVA

- Botón derecho sobre el proyecto > Propiedades



3. Establecimiento de la conexión

- Para establecer la conexión con una BBDD, se debe realizar a través de un objeto **java.sql.Connection:**

```
import java.sql.Connection;
import java.sql.DriverManager;

public class EjemplolBBDD {
    private static String bd="XE"; //Nombre de la BBDD
    private static String login="alumno"; //Usuario de la BBDD
    private static String password="alumno"; //Contraseña de la BBDD
    //Ruta del servidor
    private static String url="jdbc:oracle:thin:@localhost:1521:"+bd;
    static Connection connection=null;

    public static void conectar(){
        try{
            //Driver para Oracle
            Class.forName("oracle.jdbc.driver.OracleDriver");
            connection=DriverManager.getConnection(url,login,password);
            if (connection!=null){
                System.out.println("Conexión realizada correctamente");
            }
        }
        catch(Exception e){
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        conectar();
    }
}
```

Puede ser / en lugar de :

4. Procesador de consultas

- Establecida la conexión, se pueden ejecutar consultas SQL a la base de datos conectada.
- Las consultas se realizan a través de un objeto **java.sql.Statement,** obtenido de un objeto **Connection**.
- El resultado de una consulta es un objeto **java.sql.ResultSet.**
- Un ResultSet contiene la tabla resultante de una consulta.

4. Procesador de consultas

```
    public static void ejecutarConsulta() throws SQLException{
        int empno;
        String apellido;
        String oficio;
        st=connection.createStatement();
        rs=st.executeQuery("select emp_no, apellido,oficio from emple");
        while (rs.next()){
            empno=rs.getInt("emp_no");
            apellido=rs.getString("apellido");
            oficio=rs.getString("oficio");
            System.out.println(empno+"*"+apellido+"*"+oficio);
        }
    }
    public static void main(String[] args) {
        conectar();
        try {
            ejecutarConsulta();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

4. Procesador de consultas

- Por último, debemos cerrar todo tras su uso:

```
public static void cerrar() throws SQLException{

    if (rs!=null)
        rs.close();
    if (st!=null)
        st.close();
    if (connection!=null)
        connection.close();
}
public static void main(String[] args) {
    conectar();
    try {
        ejecutarConsulta();
        cerrar();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

4.1 La clase STATEMENT

- El objeto **st** de la clase **Statement** lo hemos utilizado para enviar sentencias SQL a la base de datos.
- Existen tres tipos de objetos Statement:
  - **Statement:** envia órdenes SQL a la base de datos **sin parámetros**
  - **PreparedStatement:** hereda de Statement. Se utiliza para ejecutar comandos SQL con o sin parámetros de entrada ya **precompilados**.
  - **CallableStatement:** hereda de PreparedStatement. Se utiliza para llamar a procedimientos almacenados en la base de datos.
- Un objeto de la clase Statement se crea mediante el método de Connection **createStatement()**.

4.1 La clase STATEMENT

- Una vez realizada la conexión y creado el objeto **Statement** se pueden llamar a tres métodos diferentes para ejecutar sentencias SQL:
  - **executeQuery:** se utiliza para ejecutar sentencias SELECT y la llamada a este método devuelve un **ResultSet** que es un objeto que almacena los datos devueltos por la BD.
  - **executeUpdate:** se utiliza para ejecutar sentencias DDL (create table, drop table, etc.) y se puede utilizar para ejecutar sentencias INSERT, UPDATE y DELETE. Devuelve un entero que indica el número de filas afectadas por la sentencia (en sentencias DDL siempre es 0).
  - **execute:** utilizado en sentencias que devuelven más de un ResulSet. Se utiliza solamente en programación avanzada.

4.1 La clase STATEMENT

- Ejemplo:

```
public static void otrasOperaciones() throws SQLException{
    st=connection.createStatement();
    st.executeUpdate("drop table tablal");
    st.executeUpdate("create table tablal (cl int primary key)");
    st.executeUpdate("insert into tablal values(10)");
}

public static void main(String[] args) {
    conectar();
    try {
        ejecutarConsulta();
        otrasOperaciones();
        cerrar();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

4.1 La clase STATEMENT

- Como buena práctica se recomienda cerrar los objetos mediante este comando:

```
st.close();
```

- La llamada al método close() hace que se libere inmediatamente la basura y se eviten posibles problemas con la memoria.

- No obstante, los objetos Statement son cerrados automáticamente por el garbage collector de Java.

ACTIVIDAD

- Realiza los ejercicios del 1 al 4 de la Hoja de Ejercicios

5. Procesado de consultas preparadas

- Las sentencias preparadas de JDBC permiten consultas o actualizaciones más eficientes (precompiladas).
- Al compilar la sentencia SQL, se analiza cuál es la estrategia adecuada según las tablas, las columnas, los índices y las condiciones de búsqueda implicados.
- Esto consume tiempo de procesador, pero al realizar la compilación una sola vez, se logra mejorar el rendimiento en siguientes consultas iguales con valores diferentes.

5. Procesado de consultas preparadas

- Otra ventaja de las sentencias preparadas es que permiten la **parametrización:**
  - La sentencia SQL se escribe una vez, indicando las posiciones de los datos que van a cambiar
  - Cada vez que se utilice, los argumentos necesarios serán sustituidos en los lugares correspondientes.
  - Los parámetros se especifican con el carácter '?'.

- El objeto **java.sql.PreparedStatement** (sentencia preparada) se obtiene a partir de una instancia de java.sql.Connection.

5. Procesado de consultas preparadas

- Ejemplo:

```
public static void ejecutarConsultaPreparada() throws SQLException{
    PreparedStatement ps =
            connection.prepareStatement("insert into tablal values (?)");

    //Los parámetros empiezan en 1
    ps.setInt(1, 20);
    ps.executeUpdate();
    ps.setInt(1, 30);
    ps.executeUpdate();
    ps= connection.prepareStatement("select * from tablal");
    rs=ps.executeQuery();

    while (rs.next()){

        System.out.println("Valor: "+rs.getInt(1));
    }
    ps.close();

}
```

5. Procesado de consultas preparadas

- Otro ejemplo:

```
preparedStatement = connection
        .prepareStatement("insert into tabla1 values (default, ?, ?)");

// Los parámetros comienzan en 1
preparedStatement.setString(1, "Isabel");
preparedStatement.setString(2, "Morera");

preparedStatement.executeUpdate();

preparedStatement.setString(1, "Lucía");
preparedStatement.setString(2, "Hernández");

preparedStatement.executeUpdate();

preparedStatement = connection
        .prepareStatement("SELECT * from tabla1");
resultSet = preparedStatement.executeQuery();
```

ACTIVIDAD

- Realiza los ejercicios del 5 al 8 de la Hoja de Ejercicios

6. Procesado de consultas ejecutables

- Una consulta ejecutable es aquella en la que se llama a un procedimiento almacenado (procedure) en la BD.

- Dado el siguiente procedimiento, creado previamente en la BD a la que se ha hecho conexión, con un parámetro de salida (OUT) que devuelve el número de filas de la tabla empleados:

```
create or replace procedure cuantosEmpleados (cuantos OUT number)
as
begin
    select count(*) into cuantos from emple;
end;
```

6. Procesado de consultas ejecutables

- El programa Java que ejecute dicho procedimiento deberá recoger el valor de salida:

```
public static void cuantosEmpleadosProc() throws SQLException{
    int cuantos;
    CallableStatement cs = connection.prepareCall("{call cuantosEmpleados(?)}");
    cs.registerOutParameter(1, Types.INTEGER);
    cs.execute();
    cuantos=cs.getInt(1);
    System.out.println("El número de empleados es: " + cuantos);

}
```

- Llamada a un procedimiento SIN parámetros:

  cs = connection.prepareCall(“{call nomProc}”);

6. Procesado de consultas ejecutables

- Llamada a un procedimiento con un parámetro OUT:

```
cs = connection.prepareCall("{call  nomProc(?)}");

 //Hay que registrar el parámetro OUT con su tipo
//Se puede registrar o indicando el número de parámetro o mediante un nombre de parámetro
cs.registerOutParameter(1,Types.VARCHAR);  ó
cs.registerOutParameter("valor",Types.VARCHAR);

 //Ejecutar el procedimiento  y recuperar el parámetro
cs.execute();
cs.getString(1); ó  cs.getString("valor");
```

6. Procesado de consultas ejecutables

- Llamada a un procedimiento con un parámetro IN:

```
cs = connection.prepareCall("{call  nomProc(?)}");

//Hay que actualizar el valor del parámetro IN
cs.setString(1,"HOLA");

 //Ejecutar el procedimiento
cs.execute();
```

6. Procesado de consultas ejecutables

- Llamada a un procedimiento con un parámetro IN/OUT:

```
cs = connection.prepareCall("{call  nomProc(?)}");

//Hay que registrar el parámetro IN/OUT con su tipo
cs.registerOutParameter(1,Types.VARCHAR);

//Hay que actualizar el valor del parámetro IN
cs.setString(1,"HOLA");

//Ejecutar el procedimiento  y recuperar el parámetro
cs.execute();
cs.getString(1);
```

6. Procesado de consultas ejecutables

- Otro ejemplo: https://www.mkyong.com/jdbc/jdbc-callablestatement-stored-procedure-out-parameter-example/

7. Procesado de transacciones

- En ocasiones se necesita que las operaciones se ejecuten en bloque (todas o ninguna) para evitar estados inconsistentes de la BD.
- En los ejemplos anteriores los cambios se producen cuando la sentencia (borrar, actualizar, insertar) se ejecuta.
  - No hace falta ejecutar una orden para actualizar cambios en la BD.
- Esto es así porque está habilitado el modo **auto-commit** en la conexión con la BD.

7. Procesado de transacciones

- Para deshabilitar el modo auto-commit en la base de datos hay que ejecutar la siguiente sentencia:

```
conn.setAutoCommit(false);
```

- Hecho esto, es posible ejecutar "virtualmente" una serie de consultas, y hacerlas efectivas al final con la llamada a **Connection.commit();**

```
try{
   //Assume a valid connection object conn
   conn.setAutoCommit(false);
   Statement stmt = conn.createStatement();

   String SQL = "INSERT INTO Employees  " +
                "VALUES (106, 20, 'Rita', 'Tez')";
   stmt.executeUpdate(SQL);
   //Submit a malformed SQL statement that breaks
   String SQL = "INSERTED IN Employees  " +
                "VALUES (107, 22, 'Sita', 'Singh')";
   stmt.executeUpdate(SQL);
   // If there is no error.
   conn.commit();
}catch(SQLException se){
   // If there is any error.
   conn.rollback();
}
```

7. Procesado de transacciones

- Si ha ocurrido una excepción, se pueden deshacer los cambios con **Connection.rollback();**
- Es posible procesar consultas por lotes (batch).
  - Para ello, se añaden consultas secuencialmente a un mismo objeto Statement mediante el método **addBatch().**
  - Cuando todas las consultas deseadas se han añadido, se ejecutan con **commit().**

7. Procesado de transacciones

```
// Create statement object
Statement stmt = conn.createStatement();

// Set auto-commit to false
conn.setAutoCommit(false);

// Create SQL statement
String SQL = "INSERT INTO Employees (id, first, last, age) " +
             "VALUES(200,'Zia', 'Ali', 30)";
// Add above SQL statement in the batch.
stmt.addBatch(SQL);

// Create one more SQL statement
String SQL = "INSERT INTO Employees (id, first, last, age) " +
             "VALUES(201,'Raj', 'Kumar', 35)";
// Add above SQL statement in the batch.
stmt.addBatch(SQL);

// Create one more SQL statement
String SQL = "UPDATE Employees SET age = 35 " +
             "WHERE id = 100";
// Add above SQL statement in the batch.
stmt.addBatch(SQL);

// Create an int[] to hold returned values
int[] count = stmt.executeBatch();

//Explicitly commit statements to apply changes
conn.commit();
```