2. Operaciones mediante JDBC

Date	Empty
Status	Not started
Type	Empty
Units	9. Base de datos

Establecimiento de conexiones

Con todas las librerías preparadas, simplemente queda la creación de la base de datos en el motor a utilizar con los usuarios y privilegios que se quiera. En este ejemplo se utilizarán las siguientes características

- Motor BD: MariaDB
- Host de connexion: 192.168.64.2
- Usuario BD: programacion
- Pass usuario: cesjpsegundo
- Servidor SQL con XAMPP (https://www.apachefriends.org/es/index.html)

Para poder establecer la conexión con la base de datos se necesita de un objeto Connection, el cual es inicializado mediante una url donde se indica el usuario, la contraseña, el host de conexión y la tabla donde va referida. Para poder realizar esta tarea hay dos posibles implementaciones:

- Crear una clase donde está este objeto con todos los parámetros pedidos. El objeto será inicializado y devuelto mediante un método

pedidos. El objeto será inicializado y devuelto mediante un método estático para que pueda ser utilizado de forma directa en las clases donde es necesario

- Crear una clase donde está este objeto con todos los parámetros pedidos. El objeto será inicializado directamente desde el constructor y se crearán tantos métodos como se necesiten dentro de la clase, de forma que el trabajo con base de datos se realiza desde la misma clase.

Para los ejemplo se utilizará la segunda metodología, ya que de esta forma se independiza la funcionalidad en capas, representando la clase de la base de datos su capa correspondiente

Para poder hacer esto, la clase GestorBD realiza la conexión mediante el método DriverManager.getConnection()

```
public class GestorDB { private Connection connection; public
GestorDB() { realizarConexion(); } private void realizarConex
ion(){ try { Class.forName("com.mysql.cj.jdbc.Driver"); Strin
g user="programacion"; String pass="cesjpsegundo"; String hos
t="192.168.64.2:3306"; String dbName= "inicial"; String url=
"jdbc:mariadb://"+host+"/"+dbName; connection = DriverManage
r.getConnection(url,user,pass); } catch (ClassNotFoundExcepti
on e) { e.printStackTrace(); } catch (SQLException e) { e.pri
ntStackTrace(); } } public void llamarConexion(){ System.out.
println(connection.toString()); }; }
```

Es importante que el método realizarConexion sea llamado desde el constructor para que el objeto connection sea inicializado directamente. Si esto no se hace se obtendría un error de NullPointer el todos los sitios donde sea utilizado el objeto

```
package videos.basedatos; import videos.basedatos.database.Ge
storDB; public class Entrada { public static void main(String
[] args) { GestorDB gestorDB = new GestorDB(); gestorDB.llama
rConexion(); } }
```

Una vez se tiene una conexión habilitada se pueden realizar las tareas CRUD de una base de datos. Para ello se utilizan los siguientes objetos:

- Statement: objeto sobre el cual podremos ejecutar una sentencia SQL mediante sintaxis propia. Este tipo de objeto no comprueba

internamente el tipo de objeto que le pasa a la quiere, pasando el proceso a ser responsabilidad del motor de base de datos. Este tipo de objeto se utiliza para las operaciones de escritura (inserción, borrado y actualización)

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

- PrepareStatement: objeto muy parecido al anterior, con la diferencia que se comprueban tipos antes de ser enviados al motor de base de datos para su ejecución. Este tipo de objeto se utiliza para las operaciones de escritura (inserción, borrado y actualización)
- ResultSet: objeto que me permite obtener todos los registros de una querrá de consulta. Son almacenados de forma directa, siendo recorridos con un formato de Iterator

Todos estos objetos parten de un objeto de tipo connection, el cual tiene una serie de métodos interesantes:

```
// cierra la base de datos. Este método es de uso obligatorio
al terminar de trabajar con la base de datos connection.close
(); // asegura la transacción. Es importante su ejecución des
pués de realizar cualquier operación para garantizar que se t
ermina el proceso connection.commit(); // evita la necesidad
de ejecutar el método anterior muchas veces, ya que lo hace d
e forma automática connection.setAutoCommit(true); // evalúa
su la base de datos ha sido inicializada y está disponible pa
ra su uso connection.isClosed(); // evalúa si la base de dato
s está disponible para operaciones solo de lectura connectio
n.isReadOnly();
```

Operaciones

Inserciones

Para poder realizar operaciones de escritura se utilizan objetos de tipo Statement o PrepareStateme. Como se ha comentado antes la diferencia entre ambos es la capacidad de evaluar los tipos de datos

Teniendo en cuenta que formato de una inserción :

```
INSERT INTO table_name (column1, column2, column3, ...) VALUE
S (value1, value2, value3, ...);
```

1. Lo primero que se debe tener es un objeto de tipo Statement, el cual se obtiene a partir de la conexión. En este caso se va a comprobar antes que la conexión no está cerrada y por lo tanto está habilitada para su uso

```
public void realizarInsercion(){ try { if (!connection.isClos
ed()){ Statement statement = connection.createStatement(); }
} catch (SQLException e) { e.printStackTrace(); } }
```

1. Con el objeto statemen creado se ejecuta el método execute, el cual obtiene como parámetro un string con la querré que se quiere ejecutar. Este método devuelve un valor booleano con el resultado de la operación

```
public void realizarInsercion(){ try { if (!connection.isClos
ed()){ Statement statement = connection.createStatement(); st
atement.execute("INSERT INTO usuario (nombre,apellido,telefon
o) VALUES ('Usuario','Apellido',123)"); } } catch (SQLExcepti
on e) { e.printStackTrace(); } }
```

Esta forma de trabajar es totalmente válida pero hay que tener en cuenta que puede ser muy compleja a la hora de escribir una query, ya que cuando los campos que se quieren insertar provienen de variables o campos capturados (en un formulario por ejemplo) hay que empezar a realizar concatenaciones teniendo en cuenta que se respetan las comillas simples ('') cuando se quiere insertar un String:

```
String usuario = "ejemplo"; String apellido = "ejemplo"; stat
ement.execute("INSERT INTO usuario (nombre,apellido,telefono)
VALUES ('"+usuario+"','"+apellido+"',"+123+")");
```

La forma más correcta de utilizarlo es mediante el método String.format() el cual garantiza que los elementos se asocian con el formato correcto a la querrá que se quiere construir

```
                    '       '       '
```

```
String usuario = "ejemplo"; String apellido = "ejemplo"; Stri
ng query = "INSERT INTO usuario (nombre,apellido,telefono) "
+ "VALUES ('%s','%s',%d)"; statement.execute(String.format(qu
ery,usuario,apellido,123));
```

Incluso se podría tener una interfaz con los nombres de todas las tablas y columnas para utilizarlas de la misma forma

1. Por último se debe cerrar tanto el statement como la conexión asociada

```
public void realizarInsercion(){ try { if (!connection.isClos
ed()){ Statement statement = connection.createStatement(); St
ring usuario = "ejemplo"; String apellido = "ejemplo"; String
query = "INSERT INTO usuario (nombre,apellido,telefono) " + "
VALUES ('%s','%s',%d)"; statement.execute(String.format(quer
y,usuario,apellido,123)); statement.close(); statement.getCon
nection().close(); } } catch (SQLException e) { e.printStackT
race(); } }
```

En el caso de querer trabajar con un PrepareStatement el proceso es muy parecido, con la única diferencia que se irán identificando los datos asociados a la querrá con sus tipos

1. Crear el prepareStatement partiendo de una query. La diferencia está en que los datos se identifican con ?

```
public void realizarInsercionPrepare(){ try { if (!connectio
n.isClosed()){ String query = "INSERT INTO usuario (nombre,ap
ellido,telefono) " + "VALUES (?,?,=)"; PreparedStatement pSta
tement = connection.prepareStatement(query); } } catch (SQLEx
ception e) { e.printStackTrace(); } }
```

1. Sustituir cada una de las ? Por los datos con los que se quiere trabajar, utilizando el método setXXX(posición,dato) donde la posición es el indice de la ? que se va a sustituir y el dato por lo que se va a sustituir y llamar al método exectute()

```
public void realizarInsercionPrepare(){ try { if (!connectio
```

```
public void realizarInsercionPrepare(){ try { if (!connectio
n.isClosed()){ String query = "INSERT INTO usuario (nombre,ap
ellido,telefono) " + "VALUES (?,?,=)"; PreparedStatement pSta
tement = connection.prepareStatement(query); String usuario =
"ejemplo"; String apellido = "ejemplo"; pStatement.setString
(1,usuario); pStatement.setString(2,apellido); pStatement.set
Int(3,123); pStatement.execute(); } } catch (SQLException e)
{ e.printStackTrace(); } }
```

Con esto lo que se consigue es que los tipos de datos que se llevan a base de datos sean los correctos

1. Por último se cierra tanto el PrepareStatement como la conexión asociada

```
pStatement.close(); pStatement.getConnection().close();
```

Borrados

Teniendo en cuenta la sintaxis de un borrado

```
DELETE FROM table_name WHERE condition;
```

La forma de proceder es la misma que en el caso de las inserciones con las siguientes diferencias:

1. Cuando se ejecuta la querrá, existe la posibilidad de llamar al método executeQuery, el cual ademas de hacer la operación devuelve el número de filas que se han visto afectadas por la sentencia

```
public void realizarBorrado(){ try { if (!connection.isClosed
()){ Statement statement = connection.createStatement(); Stri
ng usuario = "ejemplo"; String query = "DELETE FROM usuario"
+ " WHERE nombre = '%s'"; int filasAfectadas = statement.exec
uteUpdate(String.format(query,usuario)); System.out.println("
Las filas que se han borrado han sido "+filasAfectadas); stat
ement.close(); statement.getConnection().close(); } } catch
(SQLException e) { e.printStackTrace(); } }
```

En el caso de hacerlo con un prepareStatement sería de la misma

En el caso de hacerlo con un prepareStatement sería de la misma forma

```
public void realizarBorradoPrepare(){ try { if (!connection.i
sClosed()){ String usuario = "ejemplo"; String query = "DELET
E FROM usuario" + " WHERE nombre = ?"; PreparedStatement pSta
tement = connection.prepareStatement(query); pStatement.setSt
ring(1,"ejemplo"); int filasAfectadas = pStatement.executeUpd
ate(); System.out.println("Las filas que se han borrado han s
ido "+filasAfectadas); pStatement.close(); pStatement.getConn
ection().close(); } } catch (SQLException e) { e.printStackTr
ace(); } }
```

Actualizaciones

Teniendo en cuenta la sintaxis de una actualización

```
UPDATE table_name SET column1 = value1, column2 = value2, ...
WHERE condition;
```

La forma de proceder es la misma que en el caso de los borrados:

```
public void realizarActualizacion() { try { if (!connection.i
sClosed()){ Statement statement = connection.createStatement
(); String usuario = "ejemplo"; String usuarioNuevo = "ejempl
oNuevo"; String query = "UPDATE usuario SET nombre = '%s'" +
" WHERE nombre = '%s'"; int filasAfectadas = statement.execut
eUpdate(String.format(query,usuarioNuevo,usuario)); System.ou
t.println("Las filas que se han actualizado han sido "+filasA
fectadas); statement.close(); statement.getConnection().close
(); } } catch (SQLException e) { e.printStackTrace(); } }
```

En el caso de querer hacerse con un prepareStatement

```
public void realizarActualizacionPrepare() { try { if (!conne
ction.isClosed()){ String usuario = "ejemplo"; String usuario
Nuevo = "ejemploNuevo"; String query = "UPDATE usuario SET no
mbre = ?" + " WHERE nombre = "; PreparedStatement pStatement
= connection.prepareStatement(query); pStatement.setString(1,
usuarioNuevo); pStatement.setString(2,usuario); int filasAfec
tadas = pStatement.executeUpdate(String.format(query,usuarioN
```

```
tadas = pStatement.executeUpdate(String.format(query,usuarioN
uevo,usuario)); System.out.println("Las filas que se han actu
alizado han sido "+filasAfectadas); pStatement.close(); pStat
ement.getConnection().close(); } } catch (SQLException e) {
e.printStackTrace(); } }
```

Consultas

Teniendo en cuenta la sintaxis de una consulta:

```
SELECT CustomerName, City FROM Customers [WHERE condition];
```

En este caso la forma de trabajar cambia ya que no es necesario utilizar objetos de tipo statement o preparestatement. Cuando se realiza una consulta no se modifican datos, sino que simplemente se leen los registros y se obtienen. Para ello se utiliza un objeto de tipo ResultSet.

1. Crear un objeto de tipo ResultSet, para lo cual antes es necesario crear un objeto de tipo statement o preparestatement

```
public void realizarConsultaTodos(){ try { if (!connection.is
Closed()){ String query = "SELECT * FROM usuario"; Statement
statement = connection.createStatement(); ResultSet resultSet
= statement.executeQuery(String.format(query)); } } catch (SQ
LException e) { e.printStackTrace(); } }
```

1. El objeto de tipo ResulSet obtiene una lista con todos los registros. Para poder objetenerlos se cuenta con una serie de métodos que facilitan la tarea:

```
ResultSet resultSet = statement.executeQuery(String.format(qu
ery)); // indica el la fila del registro actual resultSet.get
Row(); // evalúa si hay siguiente elemento y lo sitúa resultS
et.next(); // evalúa si hay anterior elemento y lo sitúa resu
ltSet.previous(); // evalúa si el puntero está situado en la
primera posición de los registros resultantes resultSet.isFir
st(); // evalúa si el puntero está situado en la última posic
ión de los registros resultantes resultSet.isLast(); // obtie
ne el tipo de dato indicado (del registro que le toca) de la
columna en la posición 1 resultSet.getString(1); // obtiene e
l tipo de dato indicado (del registro que le toca) de la colu
```

```
mna cuyo nombre es el indicado resultSet.getString("nombre_co
lumna"); // sitúa el puntero en la primera posición de los re
gistros resultantes: resultSet.first(); // sitúa el puntero e
n la primera posición de los registros resultantes: resultSe
t.last(); // cierra el objeto resultSet.close();
```

Con la posibilidad de utilizar estos métodos, simplemente hay que recorrerlos todos he ir sacando la información de cada registro. Para ello basta con utilizar un bucle while que evalúe si hay siguiente registro. En caso positivo ir obteniendo los datos del registro que se está evaluando

```
public void realizarConsultaTodos() { try { if (!connection.i
sClosed()) { String query = "SELECT * FROM usuario"; Statemen
t statement = connection.createStatement(); ResultSet resultS
et = statement.executeQuery(String.format(query)); System.ou
t.println("ejecutado"); if (resultSet != null) { while (resul
tSet.next()) { String nombre = resultSet.getString("nombre");
String apellido = resultSet.getString("apellido"); int telefo
no = resultSet.getInt("telefono"); System.out.printf("el usua
rio es: %d %s, %s, %d %n", resultSet.getRow(), nombre, apelli
do, telefono); } } } } catch (SQLException e) { e.printStackT
race(); } }
```

1. Por último, es necesario realizar los cerrados de todos los objetos que han tenido intervención en el proceso

```
resultSet.close(); preparedStatement.close(); preparedStateme
nt.getConnection().close();
```

En el caso de querer hacerlo mediante un preparestatement sería de la siguiente forma

```
public void realizarConsultaPrepare() { try { if (!connectio
n.isClosed()) { String query = "SELECT * FROM usuario"; Prepa
redStatement preparedStatement = connection.prepareStatement
(query); ResultSet resultSet = preparedStatement.executeQuery
(); System.out.println("ejecutado"); if (resultSet != null) {
while (resultSet.next()) { String nombre = resultSet.getStrin
g("nombre"); String apellido = resultSet.getString("apellid
```

```
o"); int telefono = resultSet.getInt("telefono"); System.out.
printf("el usuario es: %d %s, %s, %d %n", resultSet.getRow(),
nombre, apellido, telefono); } } } } catch (SQLException e) {
e.printStackTrace(); } }
```