ACCESO A BASES DE DATOS RELACIONALES DESDE JAVA

Introducción a Base de Datos Relacional

Una BDR o base de datos relacional es un sistema organizador de datos que emplea el modelo relacional, garantizando así, su integridad referencial. Se compone de tablas con la información de cada una de las entidades y las relaciones entre ellas.

Este tipo de base de datos son ampliamente utilizadas y existen gran número de alternativas tanto libres como propietarias. Para la redacción de este contenido se ha optado por una alternativa libre, concretamente Oracle Express Edition.

Instalar y configurar Oracle 21c

Instalar el motor de base de datos

01

Version Express para Windows 10 de 64 bits

Configurar la base de datos

02

Contraseña de administrador y directorios

Instalar SQL Developer

03

Configuración de las cuentas / usuarios de la BDR

Diseño de la Base de Datos Relacional

Entodad-Relacion

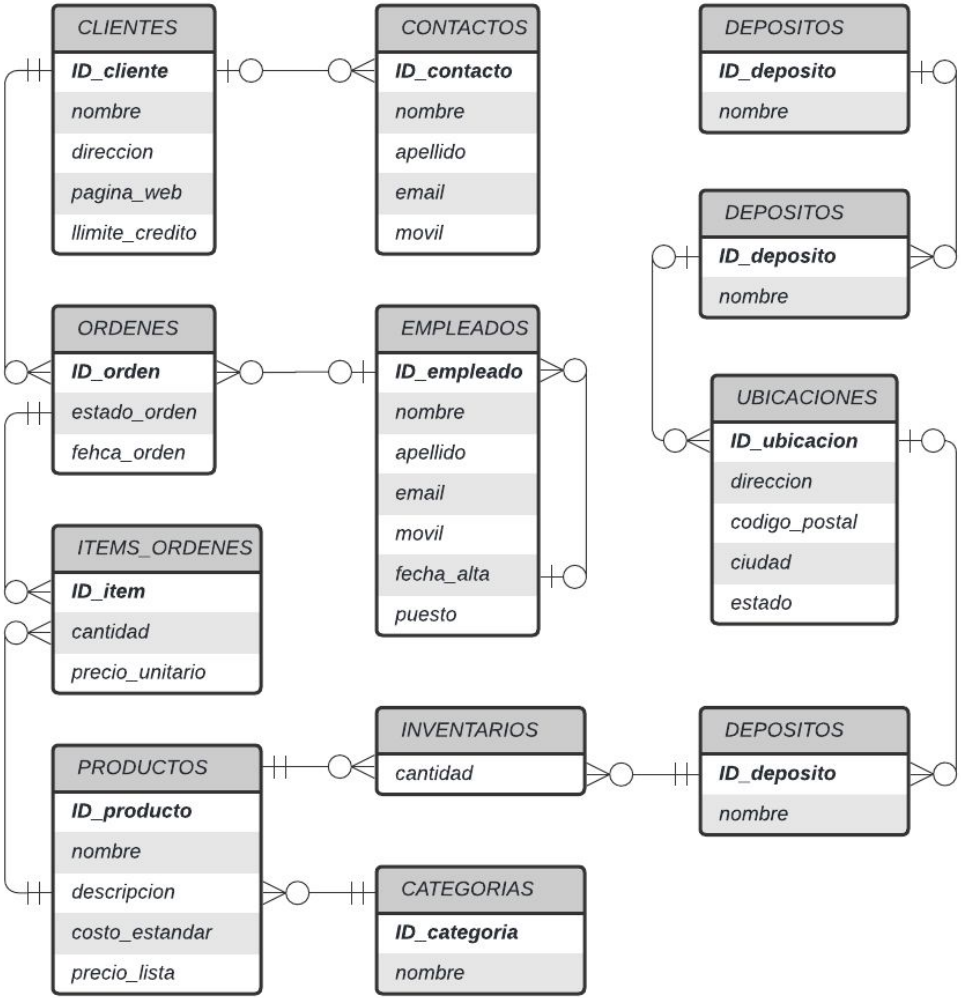
04

Modelo Físico

05

Scripts

06



BBDD PARA DEMOSTRACIÓN

Establecimiento de la Conexión

Para la conexión entre una base de datos y un programa Java, se necesita un driver JDBC en un formato de fichero .jar, el cual es específico de cada SGBD, por lo que se puede descargar desde la página oficial del proveedor de software.

Java **D**ata **B**ase **C**onnectivity es un driver que conecta un programa Java y un SGBD.

Permite establecer la conexión, manipular la BBDD mediante sentencias SQL.

Es proporcionado por el fabricante del SGBD.

Establecimiento de la Conexión

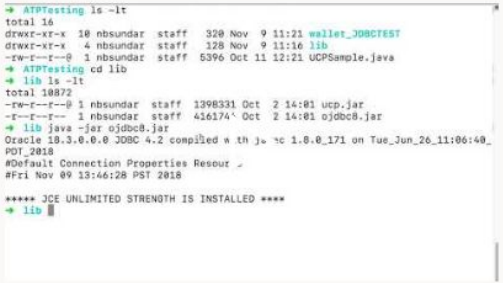
Descarga del driver JDBC Oracle

www.oracle.com/es/database/technologies/appdev/jdbc.html

ORACLE Productos Sectores Recursos Clientes Partners Desarrolladores Eventos

Mostrarme cómo

Puesta en marcha con JDBC



Conectarse a ATP con JDBC

Póngase en marcha

Descarga de JDBC | JDBC con DB Cloud

Refer to these 21c related documents for more information.

Oracle JDBC Developer's Guide
UCP Developer's Guide
Online JDBC Javadoc
Online UCP Javadoc
Online Reactive Streams Ingestion (RSI) Javadoc

Oracle Database 21c (21.5) JDBC Driver & UCP Downloads

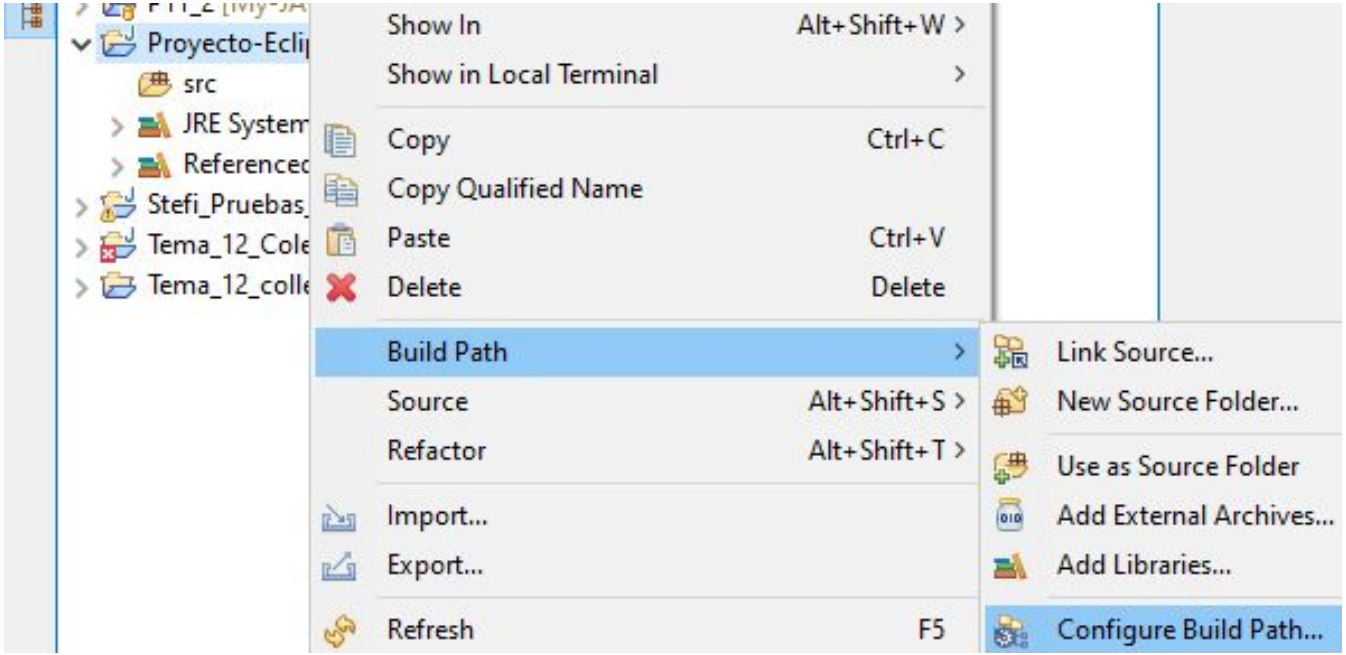
Supports Oracle Database versions - 21c, 19c, 18c, and 12.2

Name	Download	JDK Supported
Oracle JDBC driver	ojdbc11.jar	Implements JDBC 4.3 spec and certified with JDK11 and JDK17
Oracle JDBC driver	ojdbc8.jar	Implements JDBC 4.2 spec and certified with JDK8 and JDK11
Universal Connection Pool - ucp11.jar	ucp11.jar	Certified with JDK11 and JDK17
Universal Connection Pool	ucp.jar	Certified with JDK8
Zipped JDBC driver (ojdbc11.jar) and Companion Jars	ojdbc11-full.tar.gz	Certified with JDK11 and JDK17
Zipped JDBC driver (ojdbc8.jar) and Companion Jars	ojdbc8-full.tar.gz	Certified with JDK8 and JDK11

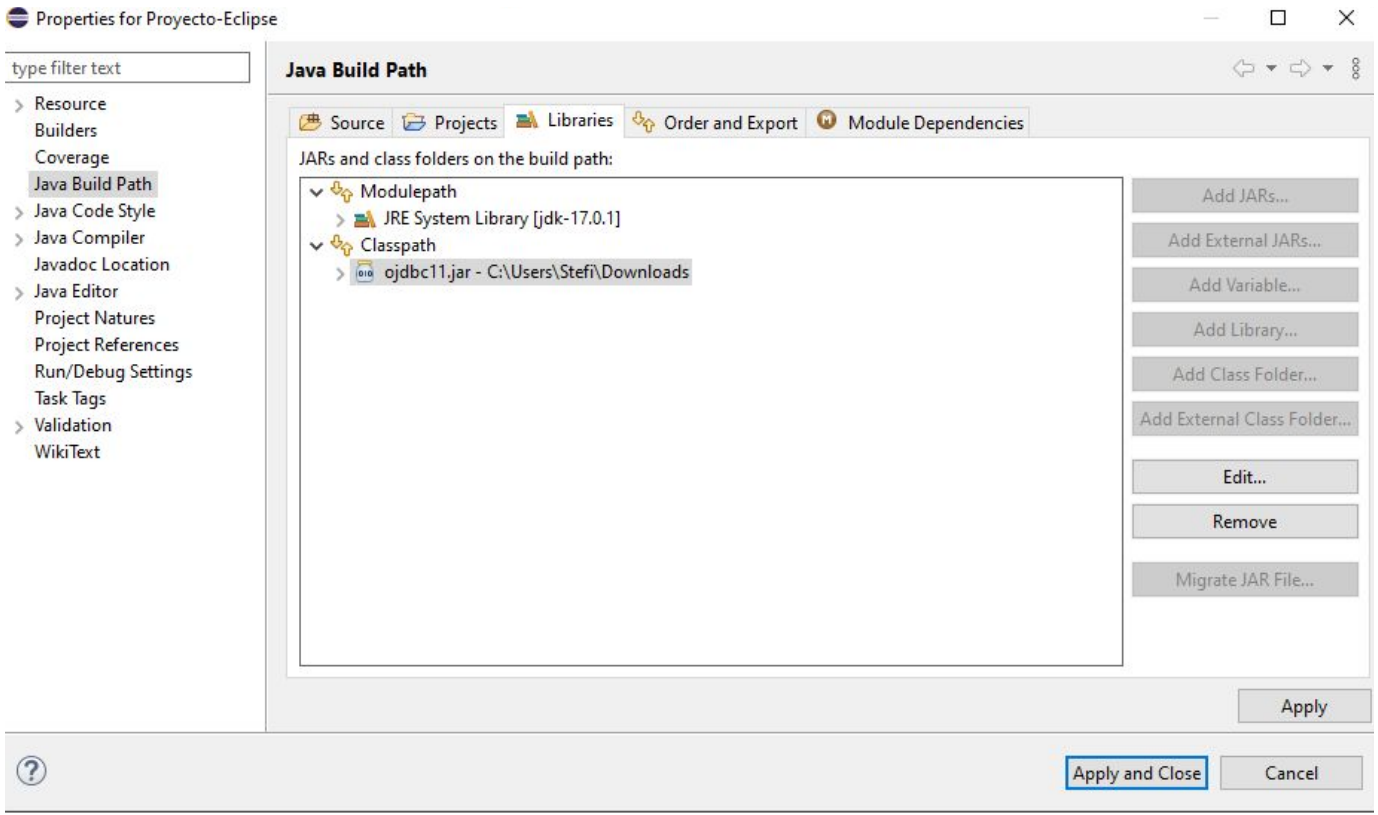
Establecimiento de la Conexión

Preparación Eclipse

1. Creación del proyecto e importar librería Oracle



Establecimiento de la Conexión



Conexión mediante código

1. Class.forName
2. Connection
3. Statement
4. ResultSet

1. Class.forName

Después de incluir el driver al proyecto, hay que registrarlo mediante código en una clase main y o un método estático Class.forName envuelto en una estructura try/catch para realizar la conexión. El primer paso es cargar el driver en el programa, esto se hace mediante el código que se ve a continuación:

```
try {
  Class.forName("oracle.jdbc.driver.OracleDriver");
} catch (java.lang.ClassNotFoundException e) {
  System.out.println("ClassNotFound: "+
  e.getMessage);
}
```

2. Connection

Luego de registrar el driver, hay que crear el objeto Conexión, este representa la conexión, para lo cual debemos construir una cadena de conexión que se nos pedirá como parámetro. En esta cadena hay que incluir el subprotocolo para el motor de base de datos, el host (dirección IP o nombre DNS en caso de ser online o el nombre plano en caso de ser local), el puerto de escucha y la base de datos. Sintaxis:

```
API:BaseDatos:driver:@servidor/IP:puerto:nombreServidor
```

En mi caso se escribiría de la siguiente forma:

```
jdbc:oracle:thin:@localhost:1521:xe
```

2. Connection

Esta cadena va a ser enviada como argumento al método DriverManager.getConnction junto con las credenciales del usuario con el que se realiza la conexión. Quedando de la siguiente forma:

```
Connection miConexion = DriverManager.getConnection (
“jdbc:oracle:thin:@localhost:1521:xe”,”system”,”stefi”);
```

3. Statement

A partir dl objeto Connection, se puede construir el objeto Statement que será el que canalice las consultas / querys hacia la base de datos. Ejemplo:

```
Statement miStatement = miConexion.createStatement();
```

4. ResultSet

Los resultados de las consultas que realiza el objeto Statement mediante el método executeQuery son almacenados en un ResultSet. Ejemplo:

```
ResultSet miResultSet = mi.Statement.executeQuery (
SELECT * FROM PRODUCTOS);
```

4. ResultSet

Para manejar los resultados de la consulta, ResultSet tiene getters a los que hay que especificarle el tipo de dato esperado correspondiente con el tipo SQL devuelto de la consulta. Ejemplos en Oracle:

- **getInt - INTEGER**
- **getLong - LONG**
- **getFloat - FLOAT**
- **getString - VARCHAR2**
- **getString - CHAR**
- **getDate - DATE**
- **getTimestamp - TIMESTAMP**

4. ResultSet

Además, la recuperación de los datos, debe estar envuelta en un bucle hasta que los registros devueltos sean la cantidad deseada. Para esto, existen los siguientes métodos:

- **next () :** Pasa al registro siguiente. Como condición devuelve true si hay un registro siguiente y false si no.
- **previous () :** Vuelve al registro anterior. Como condición devuelve true si hay un registro anterior y false si no.
- **first () :** Se posiciona en el primer registro.
- **last () :** Se posiciona en el último registro.

```java
public static void main(String[] args) {
    try {

        // 1.CREAR CONEXION
        Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection miConexion = DriverManager.getConnection(
                "jdbc:oracle:thin:@localhost:1521:xe", "system", "stefi");

        //2.CREAR STATEMENT
        Statement miStatement = miConexion.createStatement();

        //3.EJECUTAR SQL
        ResultSet miResultSet = miStatement.executeQuery("SELECT * FROM PRODUCTOS");

        //4.RECORRER EL RESULTSET
        while (miResultSet.next()) {
            System.out.println(miResultSet.getString("ID_PRODUCTO") + " " +
                               miResultSet.getString("PRODUCTO_NOMBRE") + " " +
                               miResultSet.getFloat("COSTO_ESTANDAR"));
        }

        //5.CERRAR CONEXION
        miConexion.close();

    } catch (java.lang.ClassNotFoundException e) {
        System.out.println("ClassNotFound: " + e.getMessage());
    } catch (SQLException e) {
        System.out.println("Error consulta SQL: ");
        e.printStackTrace();
    }
}
```

Cualquier tipo de dato puede ser recuperado usando getString() pero no se podrán utilizar los métodos o cálculos correspondientes a ese tipo de datos. Además, en vez de hacer referencia al nombre del campo de la tabla, se puede referenciar a ellos mediante el número de la columna en la que toma lugar en la tabla.

```
//4.RECORRER EL RESULTSET
while (miResultSet.next()) {
    System.out.println(miResultSet.getString(1) + " " +
                       miResultSet.getString(2) + " " +
                       miResultSet.getFloat(4));
}
```

Recuperación de Datos

Para realizar consultas se emplea el método executeQuery del objeto Statement.

Para recuperar la metainformación se puede optar por:

1. Utilizando consultas SQL como SHOW TABLES, SELECT USER, etc.
2. Database.MetaData

Para obtener los metadatos se puede utilizar métodos getMetaData en el objeto conexión:

Recuperación de Datos

Devuelve	Método	Descripción
String	getURL ()	Devuelve la URL de conexión.
String	getDriverName ()	Devuelve el nombre del driver empleado.
String	getDriverVersion ()	Devuelve la versión del driver empleado.
String	getDatabaseProductVersion ()	Devuelve la versión de SGBD empleado.
ResultSet	getCatalogs ()	Devuelve las BD del SGBD.
ResultSet	getTables ()	Devuelve las tablas de un BD.
ResultSet	getColumns ()	Devuelve las columnas de una tabla de la BD.

Recuperación de Datos

```
DatabaseMetaData miMetadata = miConexion.getMetaData();
mi.ResultSet = miMetadata.getCatalogs();
System.out.println("BD del sistema: ");
while(miResultSet.next()){
System.out.println("- "+miResutSet.getString("TABLE_CAT"));
}
```

Recuperación de Datos

Execute

Con el método execute se puede ejecutar cualquier operación y devuelve un valor booleano, true si devuelve un objeto ResultSet utilizable con un getResultSet y false en lo contrario.

```
If (miStatement.execute("SELECT * FROM PRODUCTOS")){
miResultSet = miStatement.getResultSet();
}
```

Actualización de la BDR

También es posible modificar una base de datos de la misma forma que realizando una operacion INSERT, UPDATE o DELETE

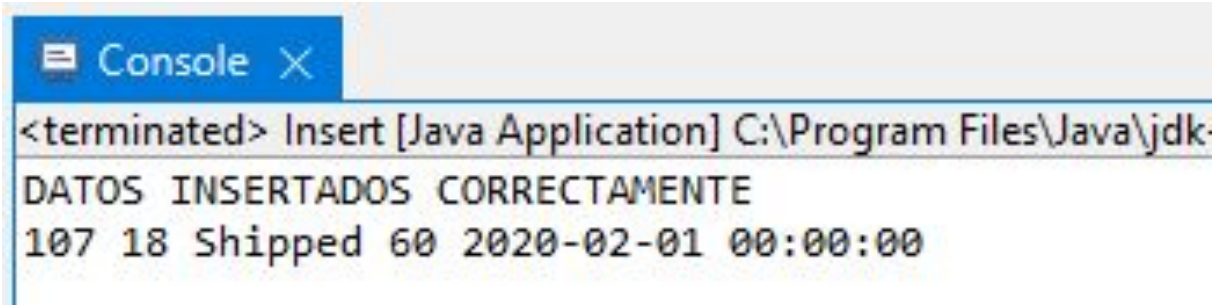
executeUpdate

Para efectuar cambios en la BDR, se usa el método executeUpdate al que se le manda las query SQL como parámetro y devuelve la cantidad de filas afectadas.

Por defecto, tras cada transacción ejecutada, se hace commit automaticamente. Para evitarlo, se puede ejecutar el método **setAutoCommit(false)** y si se desea deshacer un commit se puede hacer uso del método **rollback().**

INSERT

```java
//3.EJECUTAR SQL
miStatement.executeUpdate(
        "Insert into ordenes (id_orden,id_clientes,estado_orden,id_vendedor,fecha_orden) "
        + "values (107,18,'Shipped',60,to_date('01-FEB-20','DD-MON-RR'))");

System.out.println("DATOS INSERTADOS CORRECTAMENTE");

//4.RECORRER EL RESULTSET

ResultSet miResultSet = miStatement.executeQuery("SELECT * FROM ORDENES WHERE ID_ORDEN=107");

while (miResultSet.next()) {
        System.out.print(miResultSet.getString(1)+' '+miResultSet.getString(2)+' '+
                miResultSet.getString(3)+' '+miResultSet.getString(4)+' '+
                miResultSet.getString(5));
}
```



UPDATE

```
        //2.CREAR STATEMENT
        Statement miStatement = miConexion.createStatement();

        miConexion.setAutoCommit(false);
        miConexion.rollback();

        //4.RECORRER EL RESULTSET

        ResultSet miResultSet = miStatement.executeQuery("SELECT * FROM PRODUCTOS WHERE producto_nombre LIKE '%Samsung%'");

        while (miResultSet.next()) {
                System.out.println(miResultSet.getString(1)+' '+miResultSet.getString(2)+' '+
                        miResultSet.getString(3)+' '+miResultSet.getString(4));
        }

//3.EJECUTAR SQL
miStatement.executeUpdate("UPDATE PRODUCTOS SET COSTO_ESTANDAR=COSTO_ESTANDAR*0.50 WHERE producto_nombre LIKE '%Samsung%'");

System.out.println("DATOS MODIFICADOS CORRECTAMENTE");


        //4.RECORRER EL RESULTSET

        ResultSet miResultSet2 = miStatement.executeQuery("SELECT * FROM PRODUCTOS WHERE producto_nombre LIKE '%Samsung%'");

        while (miResultSet2.next()) {
                System.out.println(miResultSet2.getString(1)+' '+miResultSet2.getString(2)+' '+
                        miResultSet2.getString(3)+' '+miResultSet2.getString(4));
        }

//5.CERRAR CONEXION
miConexion.close();
```

```
25 Samsung MZ-75E250B/AM Series:850 EVO-Series,Type:SSD,Capaciudad:250GB,Cache:N/A 11
26 Samsung MZ-75E500B/AM Series:850 EVO-Series,Type:SSD,Capaciudad:500GB,Cache:N/A 19.73
135 Samsung MZ-V6E250 Series:960 EVO,Type:SSD,Capaciudad:250GB,Cache:512MB 11.63
285 Samsung MZ-75E1T0B/AM Series:850 EVO-Series,Type:SSD,Capaciudad:1TB,Cache:N/A 32.53
286 Samsung MZ-V6E500 Series:960 EVO,Type:SSD,Capaciudad:500GB,Cache:512MB 26.21
49 Samsung MZ-75E4T0B Series:850 EVO,Type:SSD,Capaciudad:4TB,Cache:4GB 144.21
131 Samsung MZ-V6P512BW Series:960 PRO,Type:SSD,Capaciudad:512GB,Cache:512MB 28
208 Samsung MZ-V6P2T0BW Series:960 Pro,Type:SSD,Capaciudad:2TB,Cache:2GB 105.02
231 Samsung MZ-V6E1T0 Series:960 EVO,Type:SSD,Capaciudad:1TB,Cache:1000MB 44.76
253 Samsung MZ-V6P1T0BW Series:960 Pro,Type:SSD,Capaciudad:1TB,Cache:1GB 58.32
254 Samsung MZ-7KE256BW Series:850 Pro Series,Type:SSD,Capaciudad:256GB,Cache:N/A 12.15
264 Samsung MZ-75E120B/AM Series:850 EVO-Series,Type:SSD,Capaciudad:120GB,Cache:N/A 9.31
DATOS MODIFICADOS CORRECTAMENTE
25 Samsung MZ-75E250B/AM Series:850 EVO-Series,Type:SSD,Capaciudad:250GB,Cache:N/A 5.5
26 Samsung MZ-75E500B/AM Series:850 EVO-Series,Type:SSD,Capaciudad:500GB,Cache:N/A 9.87
135 Samsung MZ-V6E250 Series:960 EVO,Type:SSD,Capaciudad:250GB,Cache:512MB 5.82
285 Samsung MZ-75E1T0B/AM Series:850 EVO-Series,Type:SSD,Capaciudad:1TB,Cache:N/A 16.27
286 Samsung MZ-V6E500 Series:960 EVO,Type:SSD,Capaciudad:500GB,Cache:512MB 13.11
49 Samsung MZ-75E4T0B Series:850 EVO,Type:SSD,Capaciudad:4TB,Cache:4GB 72.11
131 Samsung MZ-V6P512BW Series:960 PRO,Type:SSD,Capaciudad:512GB,Cache:512MB 14
208 Samsung MZ-V6P2T0BW Series:960 Pro,Type:SSD,Capaciudad:2TB,Cache:2GB 52.51
231 Samsung MZ-V6E1T0 Series:960 EVO,Type:SSD,Capaciudad:1TB,Cache:1000MB 22.38
253 Samsung MZ-V6P1T0BW Series:960 Pro,Type:SSD,Capaciudad:1TB,Cache:1GB 29.16
254 Samsung MZ-7KE256BW Series:850 Pro Series,Type:SSD,Capaciudad:256GB,Cache:N/A 6.08
264 Samsung MZ-75E120B/AM Series:850 EVO-Series,Type:SSD,Capaciudad:120GB,Cache:N/A 4.66
```

DELETE

```
//3.EJECUTAR SQL
miStatement.executeUpdate("DELETE PRODUCTOS WHERE producto_nombre LIKE '%Samsung%'");

System.out.println("DATOS MODIFICADOS CORRECTAMENTE");
```

```
25 Samsung MZ-75E250B/AM Series:850 EVO-Series,Type:SSD,Capaciudad:250GB,Cache:N/A 5.5
26 Samsung MZ-75E500B/AM Series:850 EVO-Series,Type:SSD,Capaciudad:500GB,Cache:N/A 9.87
135 Samsung MZ-V6E250 Series:960 EVO,Type:SSD,Capaciudad:250GB,Cache:512MB 5.82
285 Samsung MZ-75E1T0B/AM Series:850 EVO-Series,Type:SSD,Capaciudad:1TB,Cache:N/A 16.27
286 Samsung MZ-V6E500 Series:960 EVO,Type:SSD,Capaciudad:500GB,Cache:512MB 13.11
49 Samsung MZ-75E4T0B Series:850 EVO,Type:SSD,Capaciudad:4TB,Cache:4GB 72.11
131 Samsung MZ-V6P512BW Series:960 PRO,Type:SSD,Capaciudad:512GB,Cache:512MB 14
208 Samsung MZ-V6P2T0BW Series:960 Pro,Type:SSD,Capaciudad:2TB,Cache:2GB 52.51
231 Samsung MZ-V6E1T0 Series:960 EVO,Type:SSD,Capaciudad:1TB,Cache:1000MB 22.38
253 Samsung MZ-V6P1T0BW Series:960 Pro,Type:SSD,Capaciudad:1TB,Cache:1GB 29.16
254 Samsung MZ-7KE256BW Series:850 Pro Series,Type:SSD,Capaciudad:256GB,Cache:N/A 6.08
264 Samsung MZ-75E120B/AM Series:850 EVO-Series,Type:SSD,Capaciudad:120GB,Cache:N/A 4.66
DATOS MODIFICADOS CORRECTAMENTE
```

Actualización de la BDR

Para ejecutar una cantidad elevada de cambios, es recomendado hacer un batch para no afectar negativamente el rendimiento del sistema. La realización de un batch implica los pasos siguientes:

Paso 2

Utilizar string para guardar query SQL y añadirlas al batch mediante **addBatch()** al objeto statement creado.

Paso 1

Asignar al objeto conexión, **setAutoCommit(false)**.

Paso 3

Ejecutar todos los cambios mediante el método **executeBatch()** al objeto statement. Este devolverá un string con los números de registros afectados por cada sentencia SQL añadida.

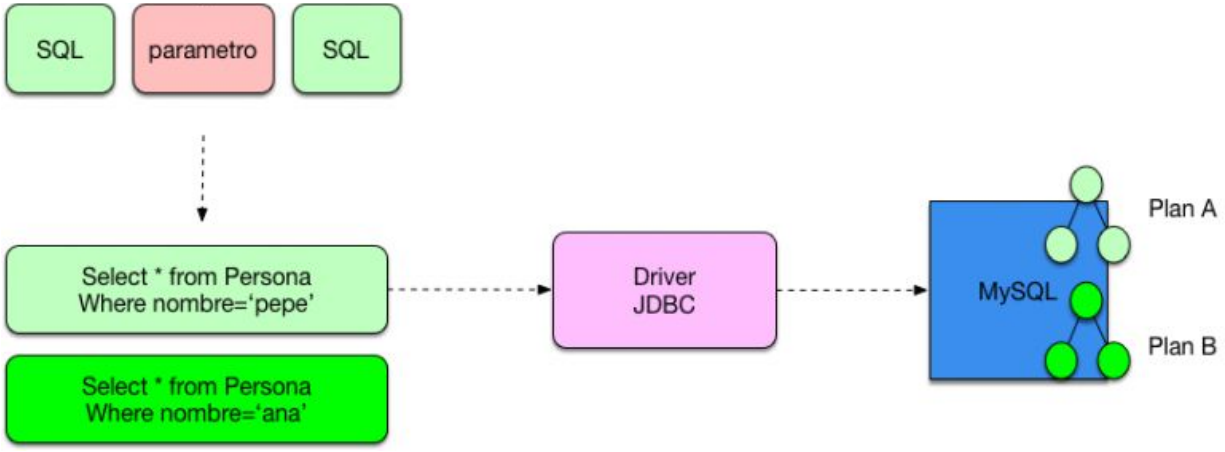
Paso 4

Finalmente, confirmar todos los cambio mediante el método **commit()** al objeto Connection.

Para vaciar el objeto Statement se hace uso del **clearBatch()**

CONSULTAS PREPARADAS - prepareStatement

Las consultas preparadas son sentencias SQL precompiladas y reutilizables, por lo que tienen un mejor rendimiento. El método sobrecargado prepareStatement() produce un objeto prepeareStatement a partir de un Stirng pasado como parámetro.
Este objeto tiene una gran cantidad de métodos que tiene como argumento un entero y otro que varía, pudiendo ser de diferentes tipos.



```java
//2.PREPARAR CONSULTA
PreparedStatement miStatement = miConexion.prepareStatement("SELECT * FROM EMPLEADOS WHERE ID_MANAGER=? AND PUESTO=?");



//3.ESTABLECER PARÁMETROS DE CONSULTA

miStatement.setInt(1, 4);
miStatement.setString(2, "Programmer");


//4.RECORRER EL RESULTSET

ResultSet miResultSet = miStatement.executeQuery();

while (miResultSet.next()) {
        System.out.println(miResultSet.getString(1)+' '+miResultSet.getString(2)+' '+
                miResultSet.getString(3)+' '+miResultSet.getString(4)+' '+
                miResultSet.getString(5)+' '+miResultSet.getString(6)+' '+
                miResultSet.getString(7)+' '+miResultSet.getString(8));
}
```

```
5 Nathan Cox nathan.cox@example.com 590.423.4568 2016-05-21 00:00:00 4 Programmer
8 Bobby Torres bobby.torres@example.com 590.423.5567 2016-02-07 00:00:00 4 Programmer
7 Charles Ward charles.ward@example.com 590.423.4560 2016-02-05 00:00:00 4 Programmer
6 Gabriel Howard gabriel.howard@example.com 590.423.4569 2016-06-25 00:00:00 4 Programmer
```

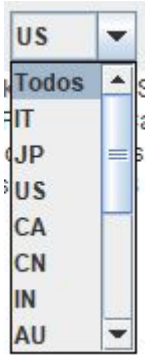
```java
//REUTILIZACION DE LA CONSULTA PREPARADA SQL

System.out.println("Reutilizacion");

miStatement.setInt(1, 49);
miStatement.setString(2, "Sales Representative");

miResultSet = miStatement.executeQuery();

while (miResultSet.next()) {
        System.out.println(miResultSet.getString(1)+' '+miResultSet.getString(2)+' '+
                miResultSet.getString(3)+' '+miResultSet.getString(4)+' '+
                miResultSet.getString(5)+' '+miResultSet.getString(6)+' '+
                miResultSet.getString(7)+' '+miResultSet.getString(8));
}
```

```
5 Nathan Cox nathan.cox@example.com 590.423.4568 2016-05-21 00:00:00 4 Programmer
8 Bobby Torres bobby.torres@example.com 590.423.5567 2016-02-07 00:00:00 4 Programmer
7 Charles Ward charles.ward@example.com 590.423.4560 2016-02-05 00:00:00 4 Programmer
6 Gabriel Howard gabriel.howard@example.com 590.423.4569 2016-06-25 00:00:00 4 Programmer
Reutilizacion
69 Evelyn Tucker evelyn.tucker@example.com 011.44.1343.929268 2016-03-11 00:00:00 49 Sales Representative
70 Eva Porter eva.porter@example.com 011.44.1343.829268 2016-03-23 00:00:00 49 Sales Representative
71 Millie Hunter millie.hunter@example.com 011.44.1343.729268 2016-01-24 00:00:00 49 Sales Representative
72 Sofia Hicks sofia.hicks@example.com 011.44.1343.629268 2016-02-23 00:00:00 49 Sales Representative
73 Lucy Crawford lucy.crawford@example.com 011.44.1343.529268 2016-03-24 00:00:00 49 Sales Representative
74 Elsie Henry elsie.henry@example.com 011.44.1343.329268 2016-04-21 00:00:00 49 Sales Representative
```

Práctica Nº1 Aplicación Consulta

Contenido:

- JComboBox, JTextArea, JButton
- ActionListener, actionPerformed
- Class.forName, Connection, Statement, ResultSet
- prepareStatement



Es una aplicación en la que se puede consultar las ubicaciones de la compañía buscando por estado y país.

Se podría haber usado para mejores ejemplos como buscar un producto por categoría y país, pero eso no se contempló en el diseño de mi BBDD

MODELO VISTA CONTROLADOR

Es una arquitectura la cual modela la lógica del programa, la interfaz del usuario y las comunicaciones y o eventos. Por lo cual, tomando en cuenta la modularidad, esto hace más fácil la programación y la corrección de error.

Estructura Modelo Vista Controlador

Modelo: Todo lo relacionado con los datos y su encapsulamiento.

Vista: Todo lo relacionado con la interfaz gráfica.

Controlador: Es la interacción / conexión que hay entre el modelo , la vista y los eventos que se producen.

Funcionamiento y Flujo de Modelo Vista Controlador

Estas tres partes en su totalidad conforman lo que es la aplicación, como la anterior, pero estructurada de otra forma.

Para utilizar una aplicación, el usuario la utiliza mediante eventos. Por ejemplo, cuando pulsamos un botón. Luego este controlador hace una consulta o manipulación en el modelo. El modelo recibe esta consulta o manipulación, la procesa y da una respuesta a la vista que así mismo se la enseña al usuario. Demostración:

Práctica N.º 2 (N.º 1 Con Vista Controlador). – MarcoAplicacion2

Paquetes:

- controlador
- vista
- marco
- principal