

Nota: La ejecución de los distintos tipos de sentencias a través del conector SQL se va a realizar empleando MySQL en XAMPP, pero los procedimientos que se describen son idénticos para el resto de BBDD empleando, se entiende, el correspondiente Driver e indicando la JDBC URL adecuada para hacer la conexión a la BD que usemos.

Para hacer los ejemplos debo tener iniciado el servidor Apache y la BD MySQL, que los inicio desde el panel de control de XAMPP. Si tengo iniciada otra instancia de MySQL, XAMPP no podrá inicial el servicio MySQL, por lo que debo antes parar la instancia que pudiera tener de MySQL Server.



1. Información sobre la base de datos y sus objetos (metadatos)

Cuando no se conoce la estructura de los objetos de la BD se recurre a metaobjetos. Se emplea la interfaz DatabaseMetaData => <https://docs.oracle.com/javase/8/docs/api/java/sql/DatabaseMetaData.html>

1.1 Ejemplo UD2_DDL_DatabaseMetaData. A partir del método getMetaData() de la interfaz Connection crea un objeto DatabaseMetaData que dispone de métodos para obtener información sobre la BD.

```
Class.forName("com.mysql.jdbc.Driver"); //Cargar el driver
//Establecemos la conexión con la BD
Connection conexion = DriverManager.getConnection("jdbc:mysql://localhost/ud2_xampp","alberto", "alberto");

DatabaseMetaData dbmd = conexion.getMetaData(); // Creamos objeto DatabaseMetaData
ResultSet resul = null;

// Usa varios métodos del objeto DatabaseMetaData para obtener inf
String nombre = dbmd.getDatabaseProductName();
String driver = dbmd.getDriverName();
String url = dbmd.getURL();
String usuario = dbmd.getUserName();

System.out.println("=====");
System.out.println("INFORMACIÓN SOBRE LA BASE DE DATOS:");
System.out.println("=====");
System.out.printf("Nombre : %s %n", nombre);
System.out.printf("Driver : %s %n", driver);
System.out.printf("URL : %s %n", url);
System.out.printf("Usuario: %s %n", usuario);
```

En concreto, el método [getTables](#) del objeto DatabaseMetaData devuelve un ResultSet con información sobre tablas y vistas de la BD

ResultSet	getTablePrivileges(String catalog, String schemaPattern, String tableNamePattern)
	Retrieves a description of the access rights for each table available in a catalog.

```
//Obtener información de las tablas y vistas que hay
resul = dbmd.getTables("ud2_xampp", null, null, null);

System.out.println("=====");
System.out.println("INFORMACIÓN SOBRE LAS TABLAS Y VISTAS:");
System.out.println("=====");
while (resul.next()) {
    String catalogo = resul.getString(1); //columna 1
    String esquema = resul.getString(2); //columna 2
    String tabla = resul.getString(3); //columna 3
    String tipo = resul.getString(4); //columna 4
    System.out.printf("%s - Catalogo: %s, Esquema: %s, Nombre: %s %n", tipo, catalogo, esquema, tabla);
}
```

```
=====
INFORMACIÓN SOBRE LA BASE DE DATOS:
=====
Nombre : MySQL
Driver : MySQL Connector Java
URL : jdbc:mysql://localhost/ud2_xampp
Usuario: alberto@localhost
```

```
=====
INFORMACIÓN SOBRE LAS TABLAS Y VISTAS:
=====
TABLE - Catalogo: ud2_xampp, Esquema: null, Nombre: departamentos
TABLE - Catalogo: ud2_xampp, Esquema: null, Nombre: empleados
```

1.2 Ejemplo UD2_DDL_GetColumns. Usa el método [getColumnns](#) del objeto DatabaseMetaData que devuelve un ResultSet con información sobre cada una de las columnas de una tabla

ResultSet	getColumnns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern) Retrieves a description of table columns available in the specified catalog.
-----------	--

```
DatabaseMetaData dbmd = conexion.getMetaData(); //Creamos objeto DatabaseMetaData
System.out.println("COLUMNAS TABLA DEPARTAMENTOS:");
System.out.println("=====");
ResultSet columnas = dbmd.getColumnns(null, null, "departamentos", null);
while (columnas.next()) {
    String nombCol = columnas.getString("COLUMN_NAME"); //getString(4)
    String tipoCol = columnas.getString("TYPE_NAME"); //getString(6)
    String tamCol = columnas.getString("COLUMN_SIZE"); //getString(7)
    String nula = columnas.getString("IS_NULLABLE"); //getString(18)

    System.out.printf("Columna: %s, Tipo: %s, Tamaño: %s, ¿Puede ser Nula:? %s %n", nombCol, tipoCol, tamCol, nula);
}
```

```
COLUMNAS TABLA DEPARTAMENTOS:
=====
Columna: dept_no, Tipo: TINYINT, Tamaño: 3, ¿Puede ser Nula:? NO
Columna: dnombre, Tipo: VARCHAR, Tamaño: 15, ¿Puede ser Nula:? YES
Columna: loc, Tipo: VARCHAR, Tamaño: 15, ¿Puede ser Nula:? YES
|
```

1.3 Ejemplo UD2_DDL_GetPrimaryKeys. Usa el método [getPrimaryKeys](#) del objeto DatabaseMetaData que devuelve un ResultSet con información sobre cada una de las claves primarias de una tabla.

ResultSet	getPrimaryKeys(String catalog, String schema, String table) Retrieves a description of the given table's primary key columns.
-----------	--

```
DatabaseMetaData dbmd = conexion.getMetaData(); //Creamos objeto DatabaseMetaData
System.out.println("CLAVE PRIMARIA TABLA DEPARTAMENTOS:");
System.out.println("=====");
ResultSet pk = dbmd.getPrimaryKeys(null, null, "DEPARTAMENTOS");
String pkDep="", separador="";
while (pk.next()) {
    pkDep = pkDep + separador + pk.getString("COLUMN_NAME");//getString(4)
    separador="+";
}
System.out.println("Clave Primaria: " + pkDep);

DatabaseMetaData dbmd = conexion.getMetaData(); //Creamos objeto DatabaseMetaData
System.out.println("CLAVE PRIMARIA TABLA DEPARTAMENTOS:");
System.out.println("=====");
ResultSet pk = dbmd.getPrimaryKeys(null, "EJEMPLO", "DEPARTAMENTOS");
String pkDep="", separador="";
while (pk.next()) {
    pkDep = pkDep + separador + pk.getString("COLUMN_NAME");//getString(4)
    separador="+";
}
System.out.println("Clave Primaria: " + pkDep);
```

```
CLAVE PRIMARIA TABLA DEPARTAMENTOS:
=====
Clave Primaria: dept_no
|
```

1.4 Ejemplo UD2_DDL_GetExportedKeys. Usa el método [getExportedKeys](#) del objeto DatabaseMetaData que devuelve un ResultSet con información sobre cada una de las claves foráneas o ajenas de una tabla.

ResultSet	getExportedKeys(String catalog, String schema, String table) Retrieves a description of the foreign key columns that reference the given table's primary key columns (the foreign keys exported by a table).
-----------	---

```
DatabaseMetaData dbmd = conexion.getMetaData();// Creamos
System.out.println("CLAVES ajenas que referencian a DEPARTAMENTOS:");
System.out.println("=====");
ResultSet fk = dbmd.getExportedKeys(null, null, "DEPARTAMENTOS");
while (fk.next()) {
    String fk_name = fk.getString("FKCOLUMN_NAME");
    String pk_name = fk.getString("PKCOLUMN_NAME");
    String pk_tablename = fk.getString("PKTABLE_NAME");
    String fk_tablename = fk.getString("FKTABLE_NAME");
    System.out.printf("Tabla PK: %s, Clave Primaria: %s %n", pk_tablename, pk_name);
    System.out.printf("Tabla FK: %s, Clave Ajena: %s %n", fk_tablename, fk_name);
}
```

```
CLAVES ajenas que referencian a DEPARTAMENTOS:
=====
Tabla PK: departamentos, Clave Primaria: dept_no
Tabla FK: empleados, Clave Ajena: dept_no
|
```

1.5 Ejemplo UD2_DDL_GetProcedures. Usa el método [getProcedures](#) del objeto DatabaseMetaData que devuelve un ResultSet con información sobre los procedimientos almacenados.

ResultSet	getProcedures(String catalog, String schemaPattern, String procedureNamePattern)
	Retrieves a description of the stored procedures available in the given catalog.

```
Class.forName("com.mysql.jdbc.Driver"); // Cargar el driver
//Establecemos la conexion con la BD
Connection conexion = DriverManager.getConnection("jdbc:mysql://localhost/ud2_xampp","alberto", "alberto");

DatabaseMetaData dbmd = conexion.getMetaData();// Creamos objeto DatabaseMetaData
ResultSet proc = dbmd.getProcedures(null, null, null);
while (proc.next()) {
    String proc_name = proc.getString("PROCEDURE_NAME");
    String proc_type = proc.getString("PROCEDURE_TYPE");
    System.out.printf("Nombre Procedimiento: %s - Tipo: %s %n", proc_name, proc_type);
}
```

1.6 Ejemplo UD2_DDL_ResultSetMetaData. Crea un objeto [ResultSetMetaData](#) a partir del método getMetadata() del objeto ResultSet que se obtiene de una consulta sobre una Tabla. Después, empleando varios métodos del ResultSetMetaData obtiene información de las columnas que ha devuelto la consulta, es decir obtiene metadatos de los valores de los campos devueltos en la consulta.

```
Class.forName("com.mysql.jdbc.Driver"); // Cargar el driver
//Establecemos la conexion con la BD
Connection conexion = DriverManager.getConnection("jdbc:mysql://localhost/ud2_xampp","alberto", "alberto");

Statement sentencia = conexion.createStatement();
ResultSet rs = sentencia.executeQuery("SELECT * FROM departamentos");

// Crea el objeto ResultSetMetaData para después usar sus métodos y así obtener
// metadatos de las columnas devueltas por la consulta
ResultSetMetaData rsmd = rs.getMetaData();

int nColumnas = rsmd.getColumnCount();
String nula;
System.out.printf("Número de columnas recuperadas: %d%n", nColumnas);
for (int i = 1; i <= nColumnas; i++) {
    System.out.printf("Columna %d: %n ", i);
    System.out.printf(" Nombre: %s %n Tipo: %s %n ", rsmd.getColumnName(i), rsmd.getColumnTypeName(i));
    if (rsmd.isNullable(i) == 0)
        nula = "NO";
    else
        nula = "SI";

    System.out.printf(" Puede ser nula?: %s %n ", nula);
    System.out.printf(" Máximo ancho de la columna: %d %n",
        rsmd.getColumnDisplaySize(i));
} // for
```

```

Número de columnas recuperadas: 3
Columna 1:
  Nombre: dept_no
  Tipo: TINYINT
  Puede ser nula?: NO
  Máximo ancho de la columna: 2
Columna 2:
  Nombre: dnombre
  Tipo: VARCHAR
  Puede ser nula?: SI
  Máximo ancho de la columna: 15
Columna 3:
  Nombre: loc
  Tipo: VARCHAR
  Puede ser nula?: SI
  Máximo ancho de la columna: 15

```

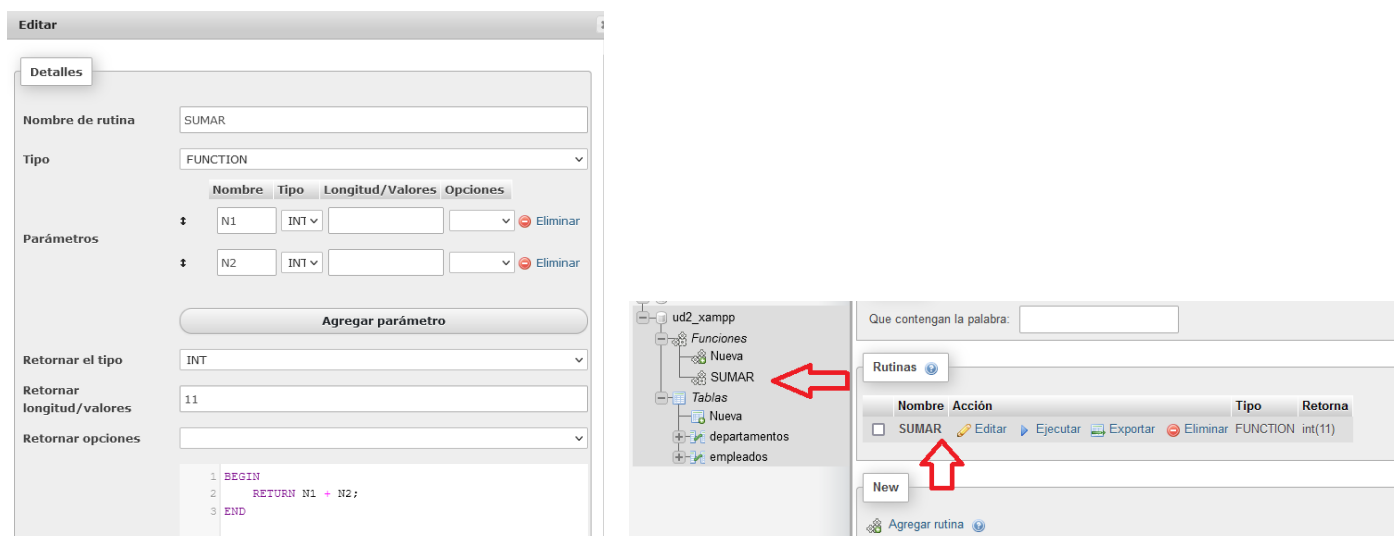
Actividad 2.8. Probar método getProcedures para obtener listado de procedimientos y funciones en la BD.

Creo una función en MySQL empleando PHPMyAdmin => <https://www.youtube.com/watch?v=9XeUVTlI8Y>

Selecciono la BD, pulso en Rutinas y luego Agregar rutina.



Indico, nombre, tipo FUNCTION, parámetros de entrada y tipo de dato que devuelve la función. Al pulsar Continuar se añade la función a la BD



Ahora añadido un procedimiento

The screenshot shows a database management interface. On the left, a tree view shows the database structure with 'ud2_xampp' selected, and 'Procedimientos' (Procedures) highlighted. The main area shows the details of the 'SUBIDA' procedure. The 'Definición' (Definition) tab displays the SQL code:

```
1 BEGIN
2   UPDATE EMPLEADOS SET SALARIO = SALARIO + 100
3   WHERE DEPT_NO = 30;
4   COMMIT;
5 END
```

The 'Ejecución' (Execution) tab shows the results of the procedure execution. A red arrow points to the 'Ejecutar' (Execute) button. The 'Rutinas' (Routines) tab shows a list of routines:

Nombre	Acción	Tipo	Retorna
SUMAR	<input type="checkbox"/> Editar <input type="checkbox"/> Ejecutar <input type="checkbox"/> Exportar <input type="checkbox"/> Eliminar	FUNCTION	int(11)
SUBIDA	<input type="checkbox"/> Editar <input type="checkbox"/> Ejecutar <input type="checkbox"/> Exportar <input type="checkbox"/> Eliminar	PROCEDURE	

A red arrow points to the 'Ejecutar' button for the 'SUBIDA' procedure. The 'New' button is also visible at the bottom.

Puedo ejecutar el procedimiento y comprobar que se produce la SUBIDA del salario de acuerdo con la definición del procedimiento.

The screenshot shows the 'Rutinas' (Routines) tab with a list of routines. The 'SUBIDA' procedure is highlighted, and the 'Ejecutar' (Execute) button is circled with a red box. The 'SUMAR' function is also visible below it.

Me conecto a la BD con JDBC y ejecuto el método getProcedures

```
/* Devuelve una lista de los procedimientos y funciones de la BD */
import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;

public class UD2_Actividad2_8 {

    public static void main(String[] args) {
        try {
            Class.forName("com.mysql.jdbc.Driver"); // Cargar el driver
            Connection conexion = DriverManager.getConnection("jdbc:mysql://localhost/ud2_xampp","alberto", "alberto");

            DatabaseMetaData dbmd = conexion.getMetaData(); // Creamos objeto DatabaseMetaData
            ResultSet proc = dbmd.getProcedures(null, null, null);
            while (proc.next()) {
                String proc_name = proc.getString("PROCEDURE_NAME");
                String proc_type = proc.getString("PROCEDURE_TYPE");
                System.out.printf("Nombre Procedimiento: %s - Tipo: %s %n", proc_name, proc_type);
            }
        } catch (ClassNotFoundException cn) {cn.printStackTrace();}
        catch (SQLException e) {e.printStackTrace();}
    }

} // fin de main
//fin de la clase
```

Y en la consola se obtiene la salida siguiente:

```
Nombre Procedimiento: SUBIDA - Tipo: 1
Nombre Procedimiento: SUMAR - Tipo: 2
```

Actividad 2.9. Visualizar información de las columnas de la tabla empleados.

Empleo los métodos del objeto `ResultSetMetaData` que se crea a partir del método `getMetadata()` del objeto `ResultSet` obtenido de una consulta sobre la Tabla de la que se quiere obtener la información.

```
public class UD2_Actividad2_9 {
    public static void main(String[] args) {
        try {
            Class.forName("com.mysql.jdbc.Driver"); // Cargar el driver
            Connection conexion = DriverManager.getConnection("jdbc:mysql://localhost/ud2_xampp", "alberto", "alberto");
            Statement sentencia = conexion.createStatement();
            ResultSet rs = sentencia.executeQuery("SELECT * FROM empleados"); // Consulta sobre la Tabla empleados
            ResultSetMetaData rsmd = rs.getMetaData();
            int nColumnas = rsmd.getColumnCount();
            String nula;
            System.out.printf("Número de columnas recuperadas: %d\n", nColumnas);
            // Recorre las columnas y muestra la información
            for (int i = 1; i <= nColumnas; i++) {
                System.out.printf("Columna %d: %n ", i);
                System.out.printf(" Nombre: %s %n Tipo: %s %n ", rsmd.getColumn(i), rsmd.getColumnType(i));
                if (rsmd.isNullable(i) == 0)
                    nula = "NO";
                else
                    nula = "SI";

                System.out.printf(" Puede ser nula?: %s %n ", nula);
                System.out.printf(" Máximo ancho de la columna: %d %n", rsmd.getColumnDisplaySize(i));
            } // for

            sentencia.close();
            rs.close();
            conexion.close();

        } catch (ClassNotFoundException cn) {cn.printStackTrace();}
        } catch (SQLException e) {e.printStackTrace();}
    }

    // fin de main
} // fin de la clase
```

```
Número de columnas recuperadas: 8
Columna 1:
Nombre: emp_no
Tipo: SMALLINT
Puede ser nula?: NO
Máximo ancho de la columna: 4
Columna 2:
Nombre: apellido
Tipo: VARCHAR
Puede ser nula?: SI
Máximo ancho de la columna: 10
Columna 3:
Nombre: oficio
Tipo: VARCHAR
Puede ser nula?: SI
Máximo ancho de la columna: 10
Columna 4:
Nombre: dir
Tipo: SMALLINT
Puede ser nula?: SI
Máximo ancho de la columna: 6
Columna 5:
Nombre: fecha_alt
Tipo: DATE
Puede ser nula?: SI
Máximo ancho de la columna: 10
Columna 6:
Nombre: salario
Tipo: FLOAT
Puede ser nula?: SI
Máximo ancho de la columna: 6
Columna 7:
Nombre: comision
Tipo: FLOAT
Puede ser nula?: SI
Máximo ancho de la columna: 6
Columna 8:
Nombre: dept_no
Tipo: TINYINT
Puede ser nula?: NO
Máximo ancho de la columna: 2
```

2. Sentencias de manipulación de datos (DML) y de definición de datos (DDL)

Una vez que se crea el objeto `Statement` (sentencia) entonces se crea un espacio para poder crear consultas y el método a emplear depende del tipo de operación de manipulación de datos a realizar:

- **ResultSet executeQuery(String)** para sentencias SELECT. El [objeto ResultSet](https://docs.oracle.com/javase/tutorial/jdbc/basics/retrieving.html) permite acceder a los valores de las columnas por el nombre de la columna o por el número de posición de la columna en la tabla. También permite obtener el número de columnas y su tipo. En la documentación de Oracle podemos encontrar ejemplos de su uso => <https://docs.oracle.com/javase/tutorial/jdbc/basics/retrieving.html>
- **int executeUpdate(String)** para sentencias de manipulación (DML) INSERT, UPDATE y DELETE, y de definición de datos (DDL) CREATE, DROP y ALTER. El método devuelve un entero con el número de filas afectadas para sentencias DML y para las sentencias DDL devuelve el valor 0.
- **boolean execute(String)** para cualquier sentencia SQL.
Este método devuelve TRUE si se devuelve un `ResultSet` y las filas devueltas se recuperan con el método `getResultSet()`.
Este método devuelve FALSE si no hay resultado o si recuenta actualizaciones y se puede usar `getUpdateCount()` para recuperar el valor devuelto

2.1 Ejemplo UD2_DML_RecorrerRegistros. Extrae los campos de una tabla de la base de datos con el método `ResultSet executeQuery(String)` y después aplica varios métodos sobre el `ResultSet` para desplazarse al último o al primer registro o fila, saber la posición que ocupa una fila y mostrar el contenido de los campos de una fila.


```

12 // Preparamos la consulta
13 Statement sentencia = conexion.createStatement();
14 String sql = "SELECT * FROM departamentos";
15 ResultSet resul = sentencia.executeQuery(sql);
16
17 resul.last(); //Nos situamos en el último registro
18 System.out.println ("NÚMERO DE FILAS: " + resul.getRow());
19
20 resul.beforeFirst(); //Nos situamos antes del primer registro
21
22 //Recorremos el resultado para visualizar cada fila
23 while (resul.next())
24     System.out.printf("Fila %d: %d, %s, %s %n",
25         resul.getRow(),
26         resul.getInt(1), resul.getString(2), resul.getString(3) );
27

```

Console

```

<terminated> UD2_DML_RecorrerRegistros [Java Application] C:\Users\JoséAlberto\p2\pool\plugins\org.eclipse.justj.open
NÚMERO DE FILAS: 5
Fila 1: 10, CONTABILIDAD, SEVILLA
Fila 2: 20, INVESTIGACIÓN, MADRID
Fila 3: 30, VENTAS, BARCELONA
Fila 4: 40, PRODUCCIÓN, BILBAO
Fila 5: 50, MARKETING, ALBACETE

```

2.2 Ejemplo UD2_DML_InsertarDpto. Añade un nuevo registro insertando los valores de cada uno de los campos.
Método => int executeUpdate(String)

```

/* asignar valores al registro a insertar */
String dep = "50"; // num. departamento
String dnombre = "MARKETING"; // nombre
String loc = "ALBACETE"; // localidad

//construir orden INSERT
String sql = String.format("INSERT INTO departamentos VALUES (%s, '%s', '%s')", dep,dnombre,loc);

System.out.println(sql);
Statement sentencia = conexion.createStatement();
int filas=0;
try {
    filas = sentencia.executeUpdate(sql.toString());
    System.out.println("Filas afectadas: " + filas);
} catch (SQLException e) {
    //e.printStackTrace();
    System.out.printf("HA OCURRIDO UNA EXCEPCIÓN:%n");
    System.out.printf("Mensaje : %s %n", e.getMessage());
    System.out.printf("SQL estado: %s %n", e.getSQLState());
    System.out.printf("Cód error : %s %n", e.getErrorCode());
}

sentencia.close(); // Cerrar Statement
conexion.close(); // Cerrar conexión

```

2.3 Ejemplo UD2_DML_InsertarEmple. Añade un nuevo registro insertando los valores de cada uno de los campos.
Método => int executeUpdate(String). Se puede comprobar la excepción que aparece cuando tratamos de insertar un registro con un valor de clave primaria existente.

```

//construir orden INSERT
String sql = "INSERT INTO empleados (emp_no, apellido, oficio,salario, dept_no) "
    + " VALUES (1001, 'nuevo', 'EMPLEADO', 2000, 10)";

System.out.println(sql);

//https://docs.oracle.com/javase/7/docs/api/java/util/Formatter.html

System.out.println(sql);
Statement sentencia = conexion.createStatement();
int filas=0;
try {
    filas = sentencia.executeUpdate(sql.toString());
    System.out.println("Filas afectadas: " + filas);
} catch (SQLException e) {
    //e.printStackTrace();
    System.out.printf("HA OCURRIDO UNA EXCEPCIÓN:%n");
    System.out.printf("Mensaje : %s %n", e.getMessage());
    System.out.printf("SQL estado: %s %n", e.getSQLState());
    System.out.printf("Cód error : %s %n", e.getErrorCode());
}

sentencia.close(); // Cerrar Statement
conexion.close(); // Cerrar conexión

```

Excepción que aparece al intentar duplicar PK

```
INSERT INTO empleados (emp_no, apellido, oficio,salario, dept_no) VALUES (1001, 'nuevo', 'EMPLEADO', 2000, 10)
INSERT INTO empleados (emp_no, apellido, oficio,salario, dept_no) VALUES (1001, 'nuevo', 'EMPLEADO', 2000, 10)
HA OCURRIDO UNA EXCEPCIÓN:
Mensaje : Duplicate entry '1001' for key 'PRIMARY'
SQL estado: 23000
Cód error : 1062
```

2.4 Ejemplo UD2_DML_ModificarSalario. Modifica los valores de un campo en aquellos registros que cumplen una determinada condición. Método => int executeUpdate(String)

```
String dep = "10", subida = "200";
//String dep = args[0];
//String subida = args[1];

try {
    Class.forName("com.mysql.jdbc.Driver");// Cargar el driver
    // Establecemos la conexión con la BD
    Connection conexion = DriverManager.getConnection("jdbc:mysql://localhost/ud2_xampp","alberto", "alberto");

    String sql = String.format("UPDATE empleados SET salario = salario + %s WHERE dept_no = %s", subida, dep);

    System.out.println(sql);

    Statement sentencia = conexion.createStatement();
    int filas = sentencia.executeUpdate(sql);
    System.out.printf("Empleados modificados: %d %n", filas);

    sentencia.close(); // Cerrar Statement
    conexion.close(); // Cerrar conexión

} catch (ClassNotFoundException cn) {
    cn.printStackTrace();
} catch (SQLException e) {
    if (e.getErrorCode() == 1062)
        System.out.println("CLAVE PRIMARIA DUPLICADA");
    else
        if (e.getErrorCode() == 1452)
            System.out.println("CLAVE AJENA " + dep + " NO EXISTE");
        else {
            System.out.println("HA OCURRIDO UNA EXCEPCIÓN:");
            System.out.println("Mensaje: " + e.getMessage());
            System.out.println("SQL estado: " + e.getSQLState());
            System.out.println("Cód error: " + e.getErrorCode());
        }
}
```

2.5 Ejemplo UD2_DML_SQLconExecute. Realiza una consulta SELECT, pero con el método execute. Esto permite actuar de acuerdo al valor booleano que devuelve la sentencia execute.

```
String sql="SELECT * FROM departamentos";
Statement sentencia = conexion.createStatement();
boolean valor = sentencia.execute(sql); // ejecuta la sentencia SQL y devuelve valor booleano

if(valor){ // actua en función del valor devuelto por el método execute
    ResultSet rs = sentencia.getResultSet();
    while (rs.next())
        System.out.printf("%d, %s, %s %n",
            rs.getInt(1), rs.getString(2), rs.getString(3));
    rs.close();
} else {
    int f = sentencia.getUpdateCount();
    System.out.printf("Filas afectadas:%d %n", f);
}
```


2.6 Ejemplo UD2_DDL_CrearVista. Crea una vista a partir de los datos de 2 tablas. Método => int executeUpdate(String)

```

/* Crea una Vista en BD MySQL en Xampp a partir de los datos de 2 tablas */
import java.sql.Connection;

public class UD2_DDL_CrearVista {

    public static void main(String[] args) throws ClassNotFoundException, SQLException {
        Class.forName("com.mysql.jdbc.Driver");// Cargar el driver
        // Establecemos la conexión con la BD
        Connection conexion = DriverManager.getConnection("jdbc:mysql://localhost/ud2_xampp","alberto", "alberto");

        StringBuilder sql = new StringBuilder();
        sql.append("CREATE OR REPLACE VIEW totales ");
        sql.append("(dep, dnombre, nemp, media) AS ");
        sql.append("SELECT d.dept_no, dnombre, COUNT(emp_no), AVG(salario) ");
        sql.append("FROM departamentos d LEFT JOIN empleados e ");
        sql.append("ON e.dept_no = d.dept_no ");
        sql.append("GROUP BY d.dept_no, dnombre ");
        System.out.println(sql);

        Statement sentencia = conexion.createStatement();
        int filas = sentencia.executeUpdate(sql.toString());
        System.out.printf("Resultado de la ejecución: %d %n", filas);

        sentencia.close(); // Cerrar Statement
        conexion.close(); // Cerrar conexión

    } // fin de main
} // fin de la clase

```

2.7 Ejemplo UD2_DML_FormularioBD. Una vez que ya sabemos conectarnos a una BD y crear sentencias DDL y DML, podemos emplear este tipo de sentencias para actualizar objetos de un interfaz gráfico, como por ejemplo un JTable.

```

// Se obtiene cada una de las etiquetas para cada columna
String[] etiquetas = new String[nColumnas];

for (int i = 1; i <= nColumnas; i++) {
    rsmd.getColumnNames(i);
    // System.out.println("Añado la columna " + rsmd.getColumnNames(i).toUpperCase());
    etiquetas[i - 1] = rsmd.getColumnNames(i).toUpperCase();
}
//System.out.println("Filas: " + filas + ", columnas: " + nColumnas);

// Recorremos el resul para cargar las filas de la consulta al array bidimensional de objetos
int numeroFila = 0;
Object[][] datos = new Object[filas][nColumnas];
resul = sentencia.executeQuery(consulta);
while (resul.next()) {
    //Bucle para cada fila, añadir las columnas
    for (int i = 0; i < nColumnas; i++) {
        datos[numeroFila][i] = resul.getObject(i + 1);
        // System.out.println("Añado la columna " + i + ", datos " + resul.getString(i + 1));
    }
    numeroFila++;
} // while

// Asignamos los datos al modelo
DefaultTableModel modelo = new DefaultTableModel(datos, etiquetas);
modelo.setColumnIdentifiers(etiquetas);
modelo.setDataVector(datos, etiquetas);

// Asignamos el modelo a la tabla
tblEmpleados.setModel(modelo);
Color fg = Color.LIGHT_GRAY;
tblEmpleados.setBackground(fg);
tblEmpleados.setForeground(Color.BLUE);

```

3. Ejecución de scripts. Ejemplo UD2_EjecutarScriptMySQL.

Esto es aplicable en las BBDD, como es el caso de MySQL, que permiten la ejecución de varias sentencias DDL y DML en la misma cadena de texto. Para habilitar esta propiedad, en el caso de MySQL hay que fijarse que en la cadena de conexión se define la propiedad `allowMultiQueries = True`

Podemos encontrar documentación de esta y otras propiedades configurables del conector MySQL en el siguiente enlace => <https://dev.mysql.com/doc/connector-j/8.0/en/connector-j-reference-configuration-properties.html>

```
try {
    //Connection connmysql = (Connection) DriverManager.getConnection("jdbc:mysql://localhost/ejemplo?allowMultiQueries=true", "ejemplo", "ejemplo");
    Connection connmysql = (Connection) DriverManager.getConnection("jdbc:mysql://localhost/ud2_xampp?allowMultiQueries=true", "alberto", "alberto");
    Statement sent = connmysql.createStatement();
    int res = sent.executeUpdate(consulta);
    System.out.println("Script creado con éxito, res = " + res);
    connmysql.close();
    sent.close();
} catch (SQLException e) {
    System.out.println("ERROR AL EJECUTAR EL SCRIPT: " + e.getMessage());
    e.printStackTrace();
}
```

Además, el script lo tenemos en un fichero, se lee su contenido y se incluye en una cadena de texto empleando un [StringBuilder](#), que es una clase sustituta de StringBuffer, que en este caso empleamos para concatenar Strings.

```
File scriptFile = new File("./script/scriptmysql.sql");
System.out.println("-----");
System.out.println("\n\nFichero de consulta : " + scriptFile.getName());
System.out.println("Convirtiendo el fichero a cadena...");

BufferedReader entrada = null;
try {
    entrada = new BufferedReader(new FileReader(scriptFile));
} catch (FileNotFoundException e) {
    System.out.println("ERROR NO ENCUENTRA el fichero: " + e.getMessage());
    e.printStackTrace();
}
String linea = null;
StringBuilder stringBuilder = new StringBuilder();
String salto = System.getProperty("line.separator");
try {
    while ((linea = entrada.readLine()) != null) {
        stringBuilder.append(linea);
        stringBuilder.append(salto);
    }
} catch (IOException e) {
    System.out.println("ERROR de E/S, al operar con el fichero: " + e.getMessage());
    e.printStackTrace();
}
String consulta = stringBuilder.toString();

System.out.println(consulta);
```

Puedo comprobar que, tras la ejecución del script se han creado varios objetos nuevos (alumnos, ...) en la BD

Tabla	Acción
<input type="checkbox"/> alumnos	★ Examinar Estructura
<input type="checkbox"/> asignaturas	★ Examinar Estructura
<input type="checkbox"/> departamentos	★ Examinar Estructura
<input type="checkbox"/> empleados	★ Examinar Estructura
<input type="checkbox"/> notas	★ Examinar Estructura
<input type="checkbox"/> totales	★ Examinar Estructura

6 tablas Número de filas

↑ ☐ Seleccionar todo Para los elementos q

4. Sentencias preparadas. Ejemplo UD2_DML_InsertarDptoPreparedStatement

Usamos la interfaz PreparedStatement para construir una sentencia SQL que incluye marcadores de posición (representado por ?). Antes de ejecutar la sentencia PreparedStatement se asigna valor a cada marcador de posición con métodos del tipo `setXXX`, siendo XXX el tipo de datos.

```
// construir orden INSERT con sentencia preparada
String sql = "INSERT INTO departamentos VALUES "
    + "( ?, ?, ? )";

System.out.println(sql);
PreparedStatement sentencia = conexion.prepareStatement(sql);
sentencia.setInt(1, Integer.parseInt(dep));
sentencia.setString(2, dnombre);
sentencia.setString(3, loc);

int filas;
try {
    filas = sentencia.executeUpdate(); // Ejecuta el INSERT INTO
    System.out.println("Filas afectadas: " + filas);
} catch (SQLException e) {
    System.out.println("HA OCURRIDO UNA EXCEPCIÓN:");
    System.out.println("Mensaje: " + e.getMessage());
    System.out.println("SQL estado: " + e.getSQLState());
    System.out.println("Cód error: " + e.getErrorCode());
}
```

4.1 Actividad 2.11 Se trata de realizar consultas sobre la BD empleando PreparedStatement

Visualizar valores (APELLIDO, SALARIO, OFICIO) de los empleados de un departamento cuyo valor se pasa al programa como argumento.

```
// construir orden SELECT con sentencia preparada
String sql = "SELECT apellido, salario, oficio FROM empleados WHERE dept_no = "
    + "?";

System.out.println(sql);
System.out.println(dep);
PreparedStatement sentencia = conexion.prepareStatement(sql);
sentencia.setString(1, dep);

try {
    ResultSet resul = sentencia.executeQuery(); // Ejecuta el SELECT con la sentencia preparada
    while (resul.next())
        System.out.printf("Fila %d: %s, %f, %s %n", resul.getRow(), resul.getString(1), resul.getFloat(2), resul.getString(3));
} catch (SQLException e) {
    System.out.println("HA OCURRIDO UNA EXCEPCIÓN:");
    System.out.println("Mensaje: " + e.getMessage());
    System.out.println("SQL estado: " + e.getSQLState());
    System.out.println("Cód error: " + e.getErrorCode());
}
```

5. Ejecución de procedimientos. Ejemplo UD2_ProcSubida

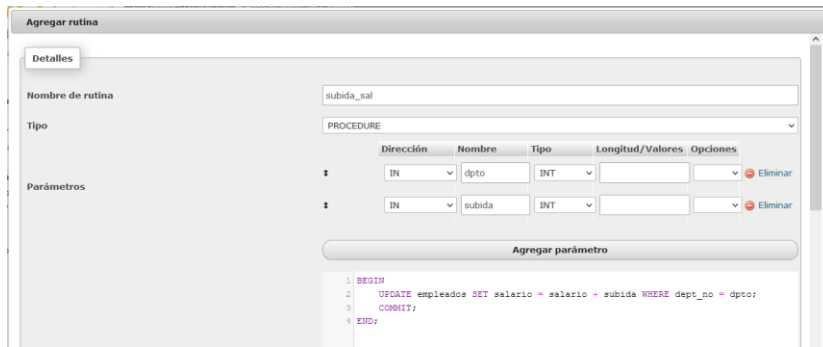
Para llamar a los procedimientos almacenados se emplea la interfaz CallableStatement

La llamada al procedimiento se declara en el método prepareCall(String) del objeto Connection.

Cuando el procedimiento o función incluye parámetros de entrada o de salida se indican en forma de marcadores de posición. Así, la llamada al procedimiento o función se puede realizar de 4 formas, dependiendo de los parámetros que tengamos:

- {call nombre_procedimiento} procedimiento sin parámetros
- {? = call_nombre_función} función sin parámetros (devuelve valor)
- {call nombre_procedimiento(?,?,...)} procedimiento con parámetros
- {? = call_nombre_funcion(?,?,...)} función con parámetros (devuelve valor)

Añado el procedimiento a la BD



✓ Se creó la rutina 'subida_sal'.

```
CREATE PROCEDURE `subida_sal` (IN `depto` INT, IN `subida` INT) NOT DETERMINISTIC CONTAINS SQL SQL SECURITY DEFINER BEGIN UPDATE
empleados SET salario = salario + subida WHERE dept_no = depto; COMMIT; END;
```

[Editar en línea] [Editar] [Crear código PHP]

```
// recuperar parametros de main
String dep = args[0]; // "10"; // departamento
String subida = args[1]; // "1000"; // subida
```

```
// construir orden DE LLAMADA
String sql = "{ call subida_sal (?, ?) }";
```

```
// Preparamos la llamada
```

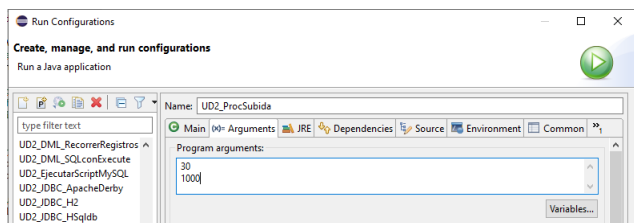
```
CallableStatement llamada = conexion.prepareCall(sql);
```

```
// Damos valor a los argumentos
```

```
llamada.setInt(1, Integer.parseInt(dep)); // primer argumento - departamento
llamada.setFloat(2, Float.parseFloat(subida)); // segundo argumento - subida
```

```
llamada.executeUpdate(); // ejecutar el procedimiento
System.out.println("Subida realizada...");
llamada.close();
conexion.close();
```

Compruebo que funciona el ejemplo. En el ejemplo, el departamento y la subida se deben pasar al programa como argumentos.



Antes de la subida

	emp_no	apellido	oficio	dir	fecha_alt	salario	comision	dept_no
<input type="checkbox"/>	1001	nuevo	EMPLEADO	NULL	NULL	3000.00	NULL	10
<input type="checkbox"/>	7369	SÁNCHEZ	EMPLEADO	7902	1990-12-17	1040.00	NULL	20
<input type="checkbox"/>	7499	ARROYO	VENDEDOR	7698	1990-02-20	2800.00	390.00	30
<input type="checkbox"/>	7521	SALA	VENDEDOR	7698	1991-02-22	2925.00	650.00	30

Después de la subida

	emp_no	apellido	oficio	dir	fecha_alt	salario	comision	dept_no
<input type="checkbox"/>	1001	nuevo	EMPLEADO	NULL	NULL	3000.00	NULL	10
<input type="checkbox"/>	7369	SÁNCHEZ	EMPLEADO	7902	1990-12-17	1040.00	NULL	20
<input type="checkbox"/>	7499	ARROYO	VENDEDOR	7698	1990-02-20	3800.00	390.00	30
<input type="checkbox"/>	7521	SALA	VENDEDOR	7698	1991-02-22	3925.00	650.00	30

6. Informes con JasperReports. Ejemplos UD2_JR_Ejemplo y UD2_JR_InformeEmpleados

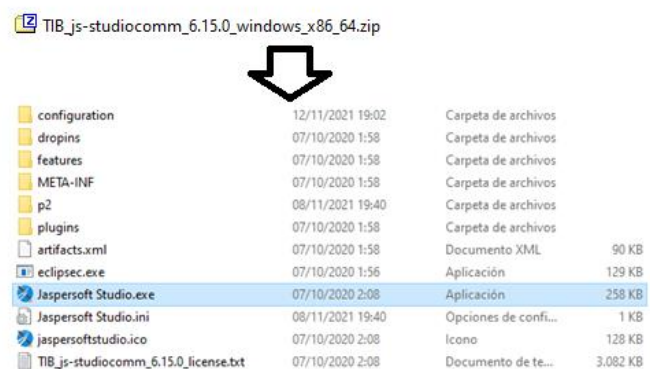
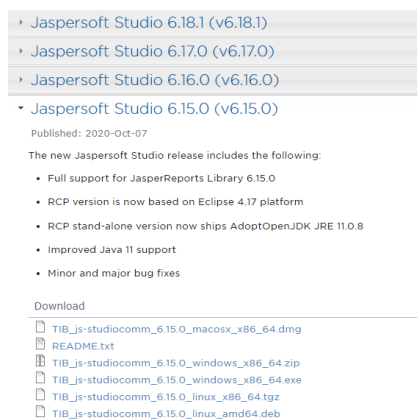
JasperReports permite generar informes en distintos formatos. Se crea una plantilla (fichero jrxml) que se puede compilar desde el proyecto java para obtener el objeto JasperReport.

Para crear la plantilla se utiliza la herramienta JasperSoft Studio, que es el sustituto de iReports para diseñar plantilla jrxml. Como no puede ser de otro modo, al igual que las aplicaciones que hemos ido creando en esta unidad, estas herramientas también necesitan conectarse a la base de datos con la que trabajan.

Descarga JasperSoft Studio => [Jaspersoft® Studio | Jaspersoft Community](#)

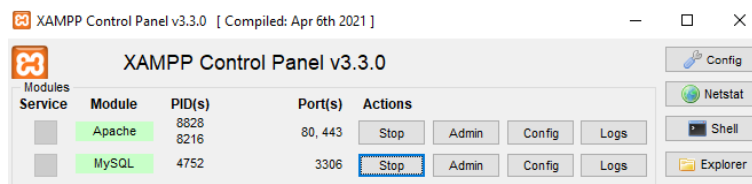
Elijo la versión 6.15.0 para Windows en zip

Descomprimo el archivo y ejecuto JaspertSoft Studio.exe

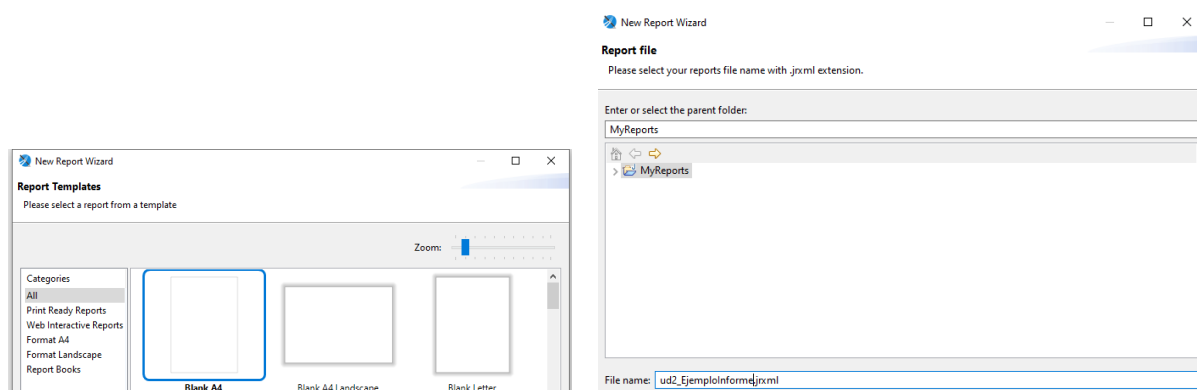


Tengo información para crear un primer informe en => [Designing a Report with Jaspersoft Studio | Jaspersoft Community](#)

Sigo el tutorial. Inicio los servicios en Xampp para usar su BD para seguir el tutorial.

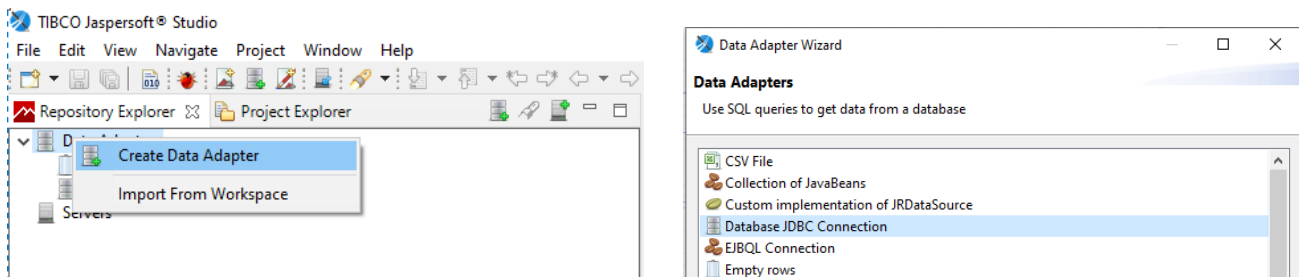


Creo un nuevo informe de nombre ud2_EjemploInforme, partiendo de una plantilla en blanco

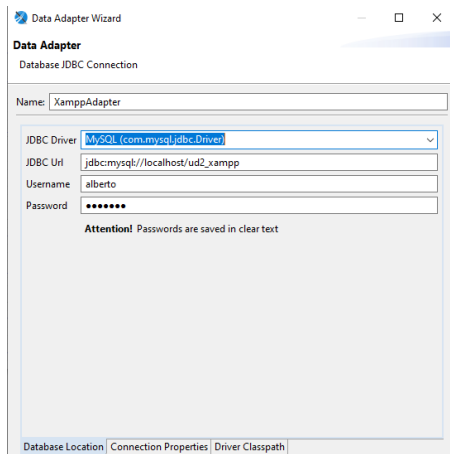


Como data source voy a usar la BD MySQL de Xampp, tengo que crear un nuevo Data Adapter y también tengo que añadir el Driver Classpath.

En el Repository Explorer selecciono Data Adapters y crear nuevo Data Adapter y se inicia el asistente de creación de Data Adapters. Indico que va a ser conexión JDBC.

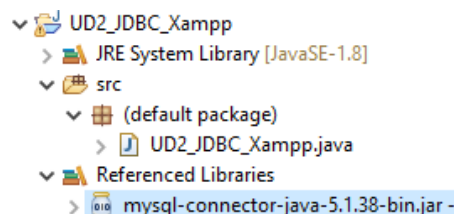
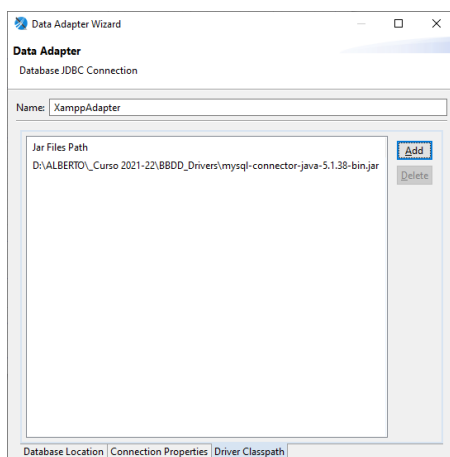


En la pestaña database location indico los mismos datos que usé para conectarme a la base de datos desde el proyecto java. En la pestaña Driver classpath le añado el fichero jar de MySQL que contiene el driver.

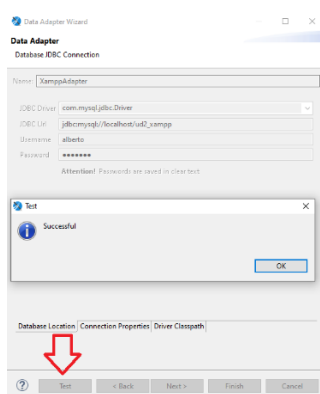


```
// Paso 2. Carga el driver JDBC
Class.forName("com.mysql.jdbc.Driver");

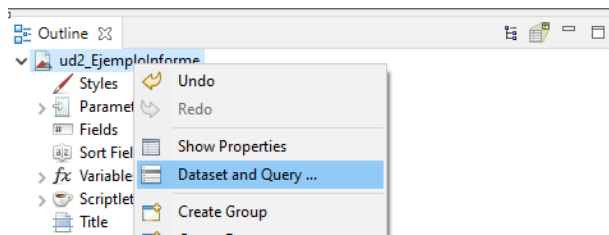
// Paso 3. Identifico el origen de datos
String url = "jdbc:mysql://localhost/ud2_xampp";
String usuario = "alberto";
String passwd = "alberto";
```



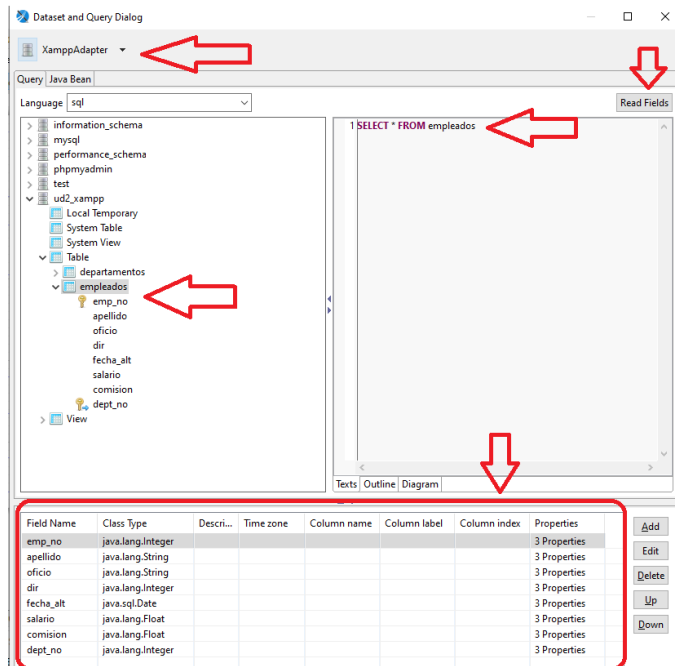
Pruedo la conexión (pulso Test) y funciona



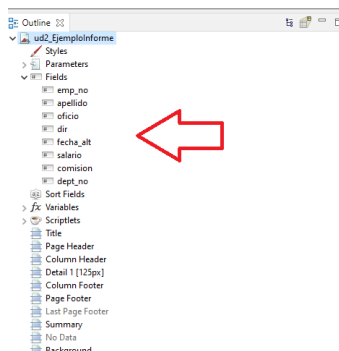
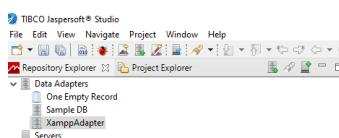
Ahora selecciono el informe y en el menú que aparece al pulsar el botón derecho selecciono 'DataSet and Query' para definir la consulta que va a extraer los datos de la BD hacia el informe.



Selecciono el Data Adapter, la base de datos, escribo la consulta y al pulsar “Read Fields” me aparecen los campos en la parte inferior.

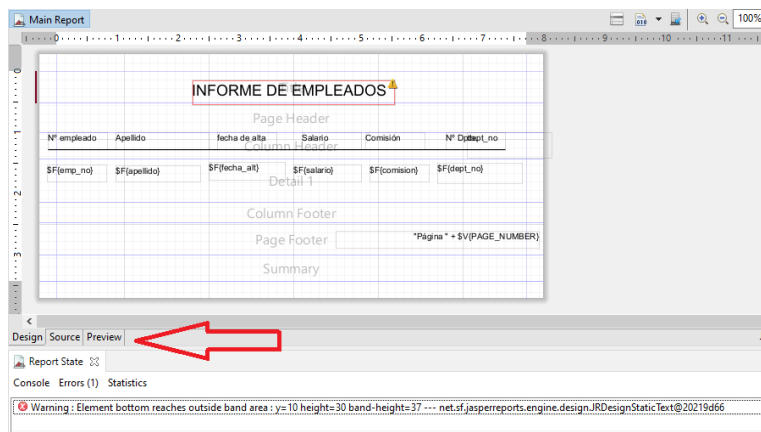


Al definir la consulta, ya tengo los campos de la tabla a mi disposición para usarlos en el Informe en la posición que quiera.



Diseño el informe arrastrando a la sección del informe adecuada (Header, Page Header, ...) los objetos (etiquetas, líneas, ...) desde la Paleta y lo mismo con los campos con la información de la BD

Trato de hacer una vista previa, pero me aparece una indicación de error que debo resolver. Me indica que es la etiqueta del Header.



Subo la etiqueta para que quede completamente dentro del Header y entonces ya compila y aparece la vista previa.

N° empleado	Apellido	fecha de alta	Salario	Comisión	N° Dpto.
7369	SÁNCHEZ	17/12/90 0:00	1040.0	null	20
7499	ARROYO	20/2/90 0:00	1800.0	390.0	30
7521	SALA	22/2/91 0:00	1925.0	650.0	30
7566	JIMÉNEZ	2/4/91 0:00	2900.0	null	20

Design Source Preview
 Report State
 Console Errors (0) Statistics
 Compilation Time 0.578 sec
 Filling Time 1.099 sec
 Report Execution Time 2.008 sec
 Export Time 0 sec
 Total Pages 2 pages
 Processed Records Count 14 records
 Fill Size 0 bytes
 Data Queried At Fri Nov 12 20:33:57 CET 2021
 Used Data From Snapshot No
 Data Snapshot File

En las propiedades del informe puedo ver la ubicación del fichero jrxml

El archivo con la plantilla lo llevo al proyecto y desde allí lo compilo y se ejecuta. No hay que olvidar incluir en el proyecto las librerías de JasperReports

```
import java.util.Map;

import com.mysql.jdbc.exceptions.jdbc4.CommunicationsException;
import com.mysql.jdbc.Connection;

import net.sf.jasperreports.engine.JRException;
import net.sf.jasperreports.engine.JasperCompileManager;
import net.sf.jasperreports.engine.JasperExportManager;
import net.sf.jasperreports.engine.JasperFillManager;
import net.sf.jasperreports.engine.JasperPrint;
import net.sf.jasperreports.engine.JasperReport;
import net.sf.jasperreports.view.JasperViewer;

import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.HashMap;
```

```

/* Uso de plantilla de JasperReports para generar informes de distintos formatos */
import java.util.Map;

public class UD2_JR_InformeEmpleados {

    public static void main(String[] args) {
        String reportSource = "./plantilla/ud2_EjemploInforme.jrxml"; // Especifica el informe a usar
        String reportHTML = "./informes/Informe.html"; // Ruta relativa donde deja el documento HTML (dentro de la carpeta 'informes' del propio proyecto)
        String reportPDF = "./informes/Informe.pdf"; // Ruta relativa donde deja el documento PDF (dentro de la carpeta 'informes' del propio proyecto)
        String reportXML = "./informes/Informe.xml"; // Ruta relativa donde deja el documento XML (dentro de la carpeta 'informes' del propio proyecto)

        // Paso de parámetros al informe a través de HashMap
        Map<String, Object> params = new HashMap<String, Object>();
        /*
        params.put("titulo", "RESUMEN DATOS DE DEPARTAMENTOS.");
        params.put("autor", "ARM");
        params.put("fecha", (new java.util.Date()).toString());
        */

        try {
            JasperReport jasperReport = JasperCompileManager.compileReport(reportSource);
            Class.forName("com.mysql.jdbc.Driver");
            // Identifico el origen de datos
            String url = "jdbc:mysql://localhost/ud2_xampp";
            String usuario = "alberto";
            String passwd = "alberto";
            Connection conn = (Connection) DriverManager.getConnection(url, usuario, passwd);
            JasperPrint MiInforme = JasperFillManager.fillReport(jasperReport, params, conn);
            // Visualizar en pantalla
            JasperViewer.viewReport(MiInforme);
            // Convertir a HTML
            JasperExportManager.exportReportToHtmlFile(MiInforme, reportHTML);
            // Convertir a PDF
            JasperExportManager.exportReportToPdfFile(MiInforme, reportPDF);
            // Convertir a XML, false es para indicar que no hay imágenes
            // (isEmbeddingImages)
            JasperExportManager.exportReportToXmlFile(MiInforme, reportXML, false);
            System.out.println("ARCHIVOS CREADOS");
        } catch (CommunicationsException c) {
            System.out.println("Error de comunicación con la BD. No está arrancada.");
        } catch (ClassNotFoundException e) {
            System.out.println("Error driver. ");
        } catch (SQLException e) {
            System.out.println("Error al ejecutar sentencia SQL ");
        } catch (JRException ex) {
            System.out.println("Error Jasper.");
            ex.printStackTrace();
        }
    }
}

```

7. Pool de conexiones

Cuando trabajamos con java contra una base de datos, es normal que realicemos una conexión como hemos hecho hasta ahora creando un objeto connection a partir de la clase DriverManager. Esto puede ser válido para una aplicación sencilla, con pocos accesos de forma simultánea a la base de datos, pero en una aplicación más compleja es mejor recurrir a un pool de conexiones. Crear directamente la conexión con el DriverManager nos puede plantear algún problema como, por ejemplo:

- Abrir una conexión, realizar cualquier operación con la base de datos y cerrar la conexión puede ser lento. La apertura de las conexiones puede tardar más que la operación que queremos realizar. Es mejor, por tanto, abrir una o más conexiones y mantenerlas abiertas.
- Mantener una única conexión abierta compartida puede traernos problemas de concurrencia de hilos. Si varios hilos intentan hacer una operación con la conexión sin sincronizarse entre ellos, puede haber conflictos.

Para evitar estos problemas, lo mejor es no abrir nosotros directamente la conexión con el DriverManager, sino delegar esta tarea en una clase que implemente la interface `javax.sql.DataSource` y, por supuesto, elegir una implementación adecuada para nuestros propósitos. En la API de java no hay ninguna implementación concreta de este `DataSource`, así que tendremos que buscarla en alguna librería externa.

Normalmente los .jar con los conectores a base de datos, como `ojdbc14.jar` para Oracle o `java-mysql-connector-5.0.5-bin.jar` para Mysql, suelen tener varias implementaciones disponibles. Es cuestión de revisar la API de ellos y elegir la adecuada. Sin embargo, también tenemos disponibles implementaciones capaces de manejar los `DataSource`, gestionando las conexiones por nosotros, con lo que se denomina un pool de conexiones.

Básicamente, un "pool" nos facilita conexiones según se las vamos pidiendo, pero las "reaprovechan" de una petición a otra. La idea es la siguiente: Le pedimos al pool una conexión. Este busca una que esté libre y nos la da, apuntando que la tenemos nosotros y que deja de estar libre. Nosotros realizamos nuestra operación (consulta, inserción, borrado, ...) y cerramos la conexión. El pool recibe esta petición de cierre y NO cierra la conexión, sino que la deja abierta y la vuelve a marcar libre para el siguiente que la pida. Con esta forma de trabajo, el pool mantiene varias conexiones abiertas que va sirviendo y marcando como usadas según se le piden. Cuando desde fuera se cierran esas conexiones, el pool no las cierra y las marca como libres, para poder reaprovecharlas. Un ejemplo de este tipo de librerías es Apache Commons, disponible en el siguiente enlace => <https://commons.apache.org/proper/commons-dbc/>

Esta API dispone de métodos que permiten configurar el comportamiento del BasicDataSource, como por ejemplo los métodos que permiten gestionar el número de conexiones:

- `setMaxActive()` : Número máximo de conexiones que se pueden abrir simultáneamente.
- `setMinIdle()` : Número mínimo de conexiones inactivas que queremos que haya. Si el número de conexiones baja de este número, se abrirán más.
- `setMaxIdle()` : Número máximo de conexiones inactivas que queremos que haya. Si hay más, se irán cerrando.
- `setInitialSize()` : Número de conexiones que se quiere que se abran en cuanto el pool comienza a trabajar (se llama por primera vez a `getConnection()`, `setLogwriter()`, `setLoginTimeout()`, `getLoginTimeout()` o `getLogWriter()`). Debe llamarse a `setInitialSize()` antes de llamar a cualquiera de estos métodos y después no puede cambiarse el valor.

Y los métodos que permiten verificar que la conexión funciona correctamente cuando se la pasa a alguien, cuando se la devuelven y mientras está inactiva, de forma que intentará la reconexión en caso de fallo:

- `setValidationQuery()` : SQL a usar con la validación. En MySQL suele ser `SELECT 1`, en Oracle `SELECT 1 FROM DUAL`.
- `setTestOnBorrow()` : Indica si se debe testear la conexión antes de pasársela a alguien.
- `setTestOnReturn()` : Indica si se debe testear la conexión cuando ese alguien la libera.
- `setTextWhileIdle()` : Indica si se debe testear la conexión mientras está inactiva. El tiempo entre tests se puede fijar con `setTimeBetweenEvictionRunsMillis()`

Para ver su funcionamiento vamos a usar el proyecto UD2_JDBC_MySQL_POOL_Conexiones. En este proyecto, se crea una clase `ConnectionPool.java` que es desde la que se crea la conexión con la base de datos, en este caso MySQL, y se definen los parámetros del pool (número máximo de conexiones simultáneas, número máximo de conexiones inactivas, ...).

```
public class ConnectionPool {  
    private final String DB="ud2_mysql";  
    private final String URL="jdbc:mysql://localhost:3306/"+DB+"?useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC";  
    private final String USER="root";  
    private final String PASS="Adirectory";  
  
    private static ConnectionPool dataSource;  
    private BasicDataSource basicDataSource=null;  
  
    private ConnectionPool(){ // Constructor de la clase. Inicializa el objeto BasicDataSource. Al ser privado no se le puede llamar directamente  
        basicDataSource = new BasicDataSource();  
        basicDataSource.setDriverClassName("com.mysql.cj.jdbc.Driver");  
        basicDataSource.setUsername(USER);  
        basicDataSource.setPassword(PASS);  
        basicDataSource.setUrl(URL);  
  
        basicDataSource.setMinIdle(5); // Mínimo conexiones inactivas  
        basicDataSource.setMaxIdle(20); // Máximo conexiones inactivas  
        basicDataSource.setMaxTotal(50); // Máximo de conexiones activas + inactivas se permiten en el pool.  
        basicDataSource.setMaxWaitMillis(-1); // Tiempo en espera que se espera. -1 significa infinito, hasta que se libere una conexión  
    }  
}
```

En la clase `ConnectionPool.java` también se incluyen los métodos a los que llamaremos para establecer y cerrar el pool de conexiones.

```

public static ConnectionPool getInstance() { // Este método garantiza que sólo habrá una instancia de esta clase
    if (dataSource == null) {
        dataSource = new ConnectionPool();
        return dataSource;
    } else {
        return dataSource;
    }
}

public Connection getConnection() throws SQLException{
    return this.basicDataSource.getConnection();
}

public void closeConnection(Connection connection) throws SQLException {
    connection.close();
}

```

Al disponer de la clase que gestiona las conexiones (connectionPool), para conectarnos desde nuestra aplicación emplearemos los mismos pasos que hemos realizado hasta el momento, con la diferencia de que el objeto connection, en lugar de obtenerlo a partir de DriverManager, lo obtendremos a partir de la clase ConnectionPool

```

// Conexión a servidor MySQL, pero usando un POOL DE CONEXIONES.

import java.sql.Connection;

public class UD2_JDBC_MySQL_POOL_Conexiones {

    public static void main(String[] args) {
        // TODO code application logic here

        try {
            // LOS PASOS 2 y 3 los hace la clase ConnectionPool con un objeto BasicDataSource
            // Paso 4. Crea objeto Connection
            Connection conexion = ConnectionPool.getInstance().getConnection();

            if(conexion !=null){ // Muestra un mensaje indicando si conecta o no
                System.out.println("conectado ");
                // ConnectionPool.getInstance().closeConnection(conexion);
            }else{
                System.out.println("No conectado");
            }

            // Paso 5. Crea objeto Statement
            Statement sentencia = conexion.createStatement();

            // Pasos 6 y 7. Ejecuta la consulta y recupera los datos en el ResultSet
            String sql = "SELECT * FROM departamentos";
            ResultSet res = sentencia.executeQuery(sql);

            // Hace el tratamiento de los datos. En este caso los imprime recorriendo el ResultSet
            while (res.next() ) {
                System.out.printf("%d, %s, %s %n", res.getInt(1), res.getString(2), res.getString(3));
            }




            res.close(); // Paso 8. Libera objeto ResultSet
            sentencia.close(); // Paso 9. Libera objeto Statement

        } catch (SQLException ex) {
            System.out.println(ex.getMessage());
        }
    }
}

```

Por último, no hay que olvidar que, para poder emplear las clases y métodos de la API Apache Commons, tenemos que incluir en el classpath de nuestro proyecto los archivos .jar que se muestran en la siguiente imagen:

▼ Drivers_Apache_Commons

-  commons-dbcp2-2.6.0.jar
-  commons-logging-1.2.jar
-  commons-pool2-2.6.2.jar