

Acceso a Datos

Manejo de Conectores

JDBC

JasperReports

Paloma Sánchez Gómez

Introducción

- Acceso a datos → Proceso de recuperación o manipulación de datos extraídos de un origen **local** o remoto.
- Los orígenes de datos pueden ser de muy diversa índole:
 - Una base de datos relacional
 - Una hoja de cálculo
 - Un fichero de texto
 - Un servicio web remoto
 - etc.
- En esta unidad vamos a centrarnos en los orígenes de datos relacionales.
- En esta unidad vamos a aprender a realizar aplicaciones Java para acceder a bases de datos relacionales partiendo del concepto y uso de los **conectores**.

El desfase objeto-relacional

- A pesar de la existencia de bases de datos orientadas a objetos (OODB) el mercado está copado por las bases de datos relacionales pues ofrecen un rendimiento y flexibilidad superiores en diversos ámbitos.
- Sin embargo, el paradigma de programación dominante a día de hoy es la Programación Orientada a Objetos (POO).
- Las bases de datos relacionales no están diseñadas para almacenar objetos ya que existe un desfase entre las estructuras del modelo relacional y los modelos de datos utilizados en POO.
- Esto quiere decir que para aunar ambos conceptos se requiere de un esfuerzo extra a la hora de programar para adaptar unas estructuras a otras.
- A esto se le denomina el **desfase objeto-relacional** y se refiere a los problemas a los problemas que ocurren debido a las diferencias entre un modelo y otro.

El desfase objeto-relacional

- A pesar de la existencia de bases de datos orientadas a objetos (OODB) el mercado está copado por las bases de datos relacionales pues ofrecen un rendimiento y flexibilidad superiores en diversos ámbitos.
- Sin embargo, el paradigma de programación dominante a día de hoy es la Programación Orientada a Objetos (POO).
- Las bases de datos relacionales no están diseñadas para almacenar objetos ya que existe un desfase entre las estructuras del modelo relacional y los modelos de datos utilizados en POO.
- Esto quiere decir que para aunar ambos conceptos se requiere de un esfuerzo extra a la hora de programar para adaptar unas estructuras a otras.
- A esto se le denomina el **desfase objeto-relacional** y se refiere a los problemas a los problemas que ocurren debido a las diferencias entre un modelo y otro.

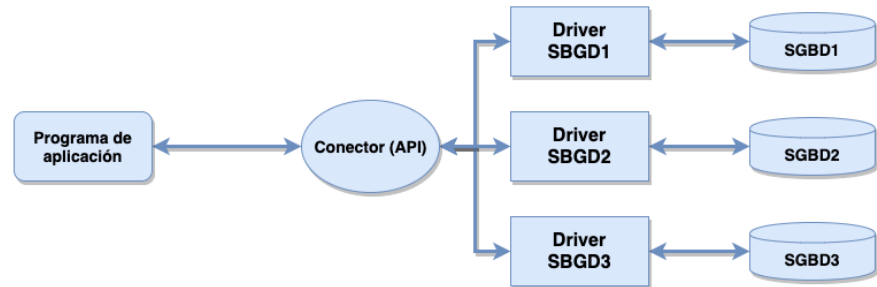
Conectores

- Los sistemas gestores de bases de datos (SGBD) de distintos tipos tienen sus propios lenguajes especializados para operar con los datos que almacenan.
- Por su parte, los programas de aplicación se escriben en lenguajes de programación de propósito general, como Java.
- Para que estos programas puedan operar con los SGBD se necesitan mecanismos que permitan a los programas de aplicación comunicarse con las bases de datos en estos lenguajes.
- Estos se implementan en una API y se denominan **conectores**.



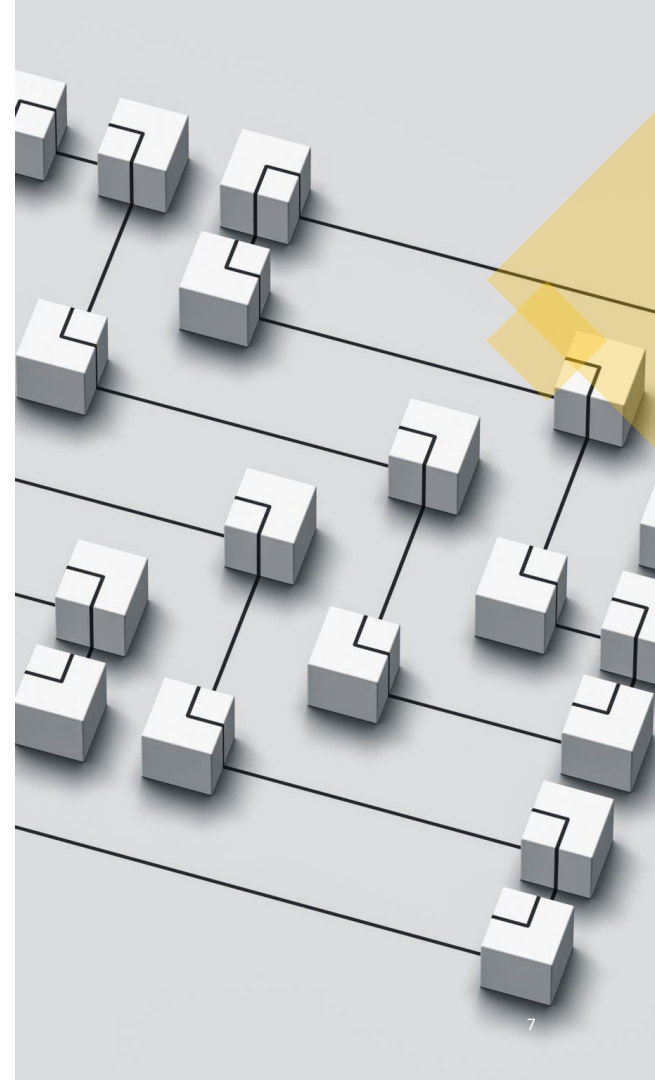
Conectores para RDBMS

- Para trabajar con sistemas gestores de bases de datos relacionales (RDBMS) se emplea el lenguaje SQL.
- Cada RDBMS utiliza su propia versión de SQL, con sus propias peculiaridades, es por ello que debemos usar sus propias estructuras de bajo nivel.
- El uso de drivers permite desarrollar una arquitectura genérica en la que el conector tiene una interfaz común tanto para las aplicaciones como para las bases de datos, y los drivers se encargan de las particularidades de las diferentes bases de datos.



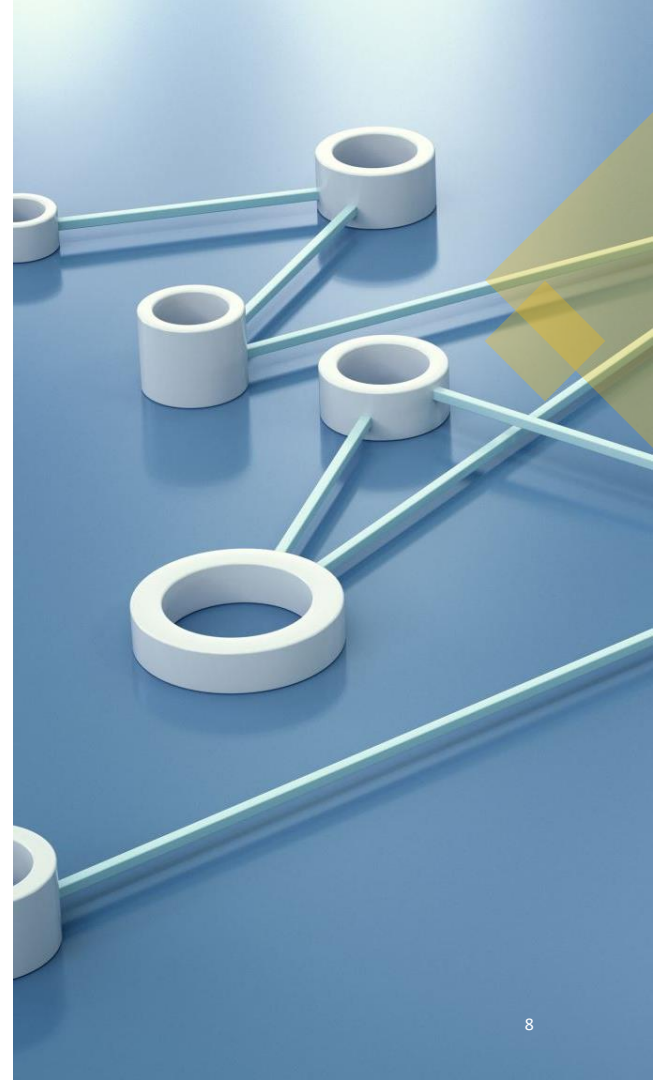
Conectores para RDBMS

- De esta manera, el conector no es solo una API, sino una arquitectura, porque especifica unas interfaces que los distintos drivers tienen que implementar para acceder a las bases de datos.
- Existen diferentes arquitecturas en este sentido:
 - ODBC
 - OLE-DB
 - ADO
 - JDBC
- Las predominantes a día de hoy son ODBC y JDBC ya que la casi totalidad de los SGBD las implementan y permiten trabajar con ellas.
- Los beneficios que proporcionan los conectores basados en drivers se consiguen a cambio de una mayor complejidad de programación.



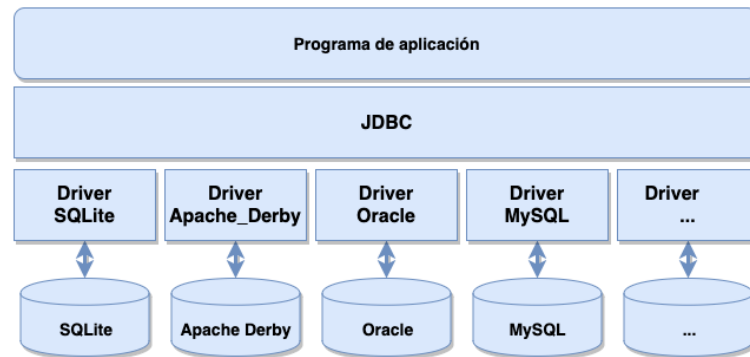
Conectores para RDBMS

- ODBC (Open Database Connectivity)
 - Define una API que pueden usar las aplicaciones para abrir una conexión con la base de datos, enviar consultas, actualizaciones, etc.
 - Las aplicaciones pueden usar esta API para conectarse a cualquier servidor de bases de datos compatible.
 - Está escrito en lenguaje C.
- JDBC (Java Database Connectivity)
 - Es la API por antonomasia de conexión de bases de datos con aplicaciones Java y es en la que nos centraremos en la presente unidad.



Acceso a DB mediante JDBC

- JDBC proporciona una API para acceder a fuentes de datos orientados a bases de datos relacionales SQL.
- Provee una arquitectura que permite a los fabricantes puedan crear Drivers que permitan a las aplicaciones Java el acceso a datos.
- JDBC dispone de una interface distinta para cada SGBD, es lo que llamamos Driver.
- Esto permite que las llamadas a métodos Java se correspondan con la API de la base de datos.

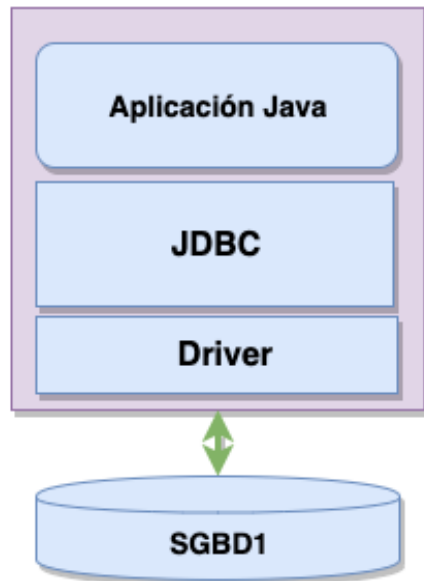




Acceso a DB mediante JDBC

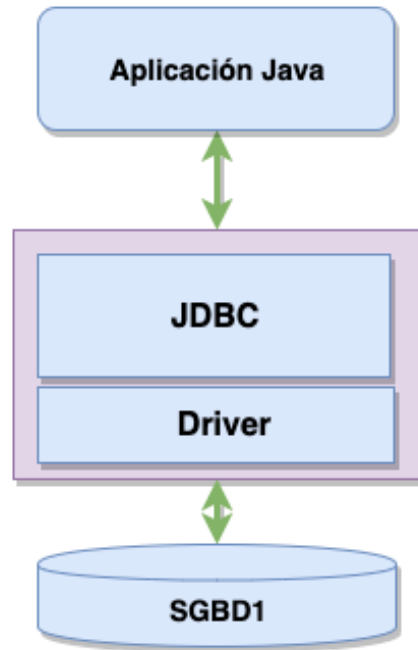
- JDBC consta de un conjunto de interfaces y clases que nos permiten escribir aplicaciones en Java para gestionar las siguientes tareas con una base de datos relacional:
 - Conectarse a una base de datos.
 - Enviar consultas e instrucciones de actualización a una base de datos.
 - Recuperar y procesar los resultados recibidos de la base de datos en respuesta a dichas consultas.

Acceso a DB mediante JDBC



- Modelo de acceso
 - La API JDBC es compatible con dos modelos de acceso:
 - **Modelo de dos capas** → La aplicación Java habla directamente con la base de datos, esto requiere un driver JDBC residiendo en el mismo espacio que la aplicación.
 - Desde el programa Java se envían sentencias SQL al SGBD para que las procese y este envía los resultados de vuelta al programa.
 - El SGBD puede encontrarse en la misma o en otra máquina y las peticiones se realizan a través de la red.
 - El driver es el encargado de gestionar las conexiones de forma transparente al programador.

Acceso a DB mediante JDBC



- Modelo de acceso
 - **Modelo de tres capas** → Los comandos se envían a una capa intermedia que se encarga de enviar los comandos SQL a la base de datos y de recoger los resultados.



Acceso a DB mediante JDBC

- Tipos de drivers
 - **JDBC-ODBC Bridge** → Permite el acceso a SGBD JDBC mediante el protocolo ODBC.
 - **Native** → Controlador escrito parcialmente en Java y en código nativo de bases de datos. Traduce las llamadas de la API Java en llamadas propias del SGBD.
 - **Network** → Controlador de Java puro que utiliza un protocolo de red para comunicarse con el servidor de Base de datos.
 - **Thin** → Controlador de Java puro con protocolo nativo. Traduce las llamadas de la API a llamadas propias del protocolo de red utilizado por el SGBD.

Acceso a DB mediante JDBC

- Funcionamiento JDBC
 - JDBC define clases e interfaces en el paquete java.sql. La siguiente tabla muestra las más relevantes.

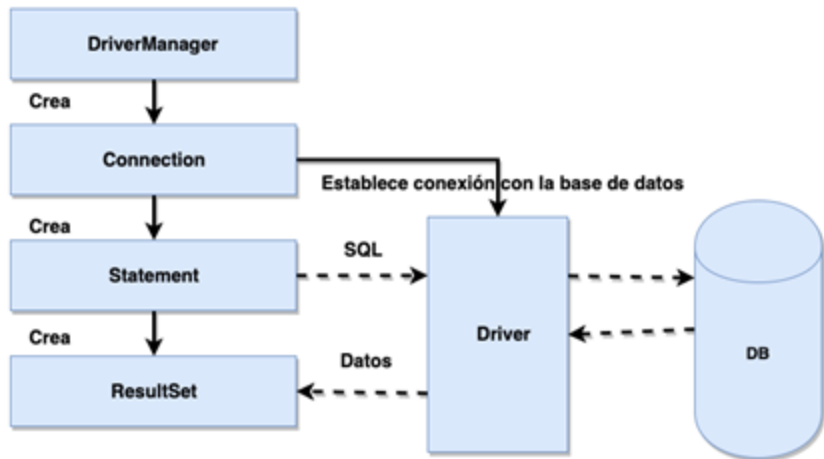
Clase o interface	Descripción
Driver	Permite conectarse a una base de datos
DriverManager	Permite gestionar los Driver instalados en el sistema
DriverPropertyInfo	Proporciona información del driver
Connection	Representa una conexión la base de datos
DatabaseMetadata	Proporciona información de la base de datos
Statement	Permite ejecutar sentencias SQL sin parámetros
PreparedStatement	Permite ejecutar sentencias SQL con parámetros
CallableStatement	Permite ejecutar sentencias SQL con parámetros de entrada y salida
ResultSet	Contiene el resultado de las consultas
ResultSetMetadata	Permite obtener información de un ResultSet

5. Acceso a DB mediante JDBC

Funcionamiento JDBC

- El funcionamiento de un programa con JDBC sigue los siguientes pasos:

- Importar las clases necesarias
- Cargar el Driver JDBC
- Identificar el origen de datos
- Crear un objeto Connection
- Crear un objeto Statement
- Ejecutar la consulta con el objeto Statement
- Recuperar el resultado en un ResultSet
- Liberar el objeto ResultSet
- Liberar el objeto Statement
- Liberar el objeto Connection



Acceso a DB mediante JDBC

- Funcionamiento JDBC
- Veamos un ejemplo de conexión y consulta a base de datos MySQL usando el siguiente esquema de bases de datos.
- DBName → Empresa
- TableName → Empleados
 - Campos:
 - NIF Varchar(9) PK
 - Nombre Varchar(100)
 - Apellidos varchar(100)
 - Salario Float (6,2)

5. Acceso

Funcionamiento JDBC

```
public static void main(String[] args) {  
    //cargar el Driver  
    try {  
        Class.forName("com.mysql.jdbc.Driver");  
  
        //Establecemos la conexión con la BBDD  
        Connection conexion=  
            DriverManager.getConnection("jdbc:mysql://localhost/empresa", "root", "");  
        //Preparamos la consulta  
        Statement sentencia= conexion.createStatement();  
        String sql= "Select * from empleados";  
        ResultSet resultado= sentencia.executeQuery(sql);  
  
        //Recorremos el resultSet obteniendo su contenido.  
        while(resultado.next()) {  
            String nif= resultado.getString("nif");  
            String nombre= resultado.getString("nombre");  
            String apellidos= resultado.getString("Apellidos");  
            Double salario= resultado.getDouble("salario");  
            System.out.println(nif+ " "+ nombre + " "+apellidos+ " "+salario);  
        }  
        //Liberamos los recursos  
        resultado.close();  
        sentencia.close();  
        conexion.close();  
    } catch (ClassNotFoundException e) {  
        // TODO Auto-generated catch block  
        e.printStackTrace();  
    } catch (SQLException e) {  
        // TODO Auto-generated catch block  
        e.printStackTrace();  
    }  
}
```

5. Acceso a DB mediante JDBC

Funcionamiento JDBC

En el código anterior hemos seguido los siguientes pasos

1. **Cargar el driver** → En primer lugar se carga el driver con el método forName de la clase Class, se le pasa un objeto String con el nombre de la clase del driver como argumento.

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

1. **Establecer la conexión** → A continuación se establece la conexión con la BBDD usando la clase DriverManager y el método getConnection pasando como parámetros *URL*, *USER* y *PASS* de la BBDD.

```
Connection conexion=  
    DriverManager.getConnection("jdbc:mysql://localhost/empresa", "root", "");
```

5. Acceso a DB mediante JDBC

Funcionamiento JDBC

- 3. Ejecutar sentencias SQL** → A continuación realizamos la consulta a través de la interface Statement.
- i. Para obtener un objeto Statement se llama al método createStatement() de un objeto Connection válido.
 - ii. El objeto Statement tiene el método executeQuery() que ejecuta una consulta en la BBDD
 1. Este método recibe como parámetro una String que debe contener una sentencia SQL.

```
Statement sentencia= conexion.createStatement();  
String sql= "Select * from empleados";  
ResultSet resultado= sentencia.executeQuery(sql);
```

- i. El resultado se recoge en un objeto ResultSet, este objeto es una suerte de array o listado que incluye todos los datos devueltos por el SGBD

Acceso a DB mediante JDBC

- Funcionamiento JDBC
 - **Recorrer el resultado** → Una vez hemos obtenido los resultados de la consulta y los hemos almacenado en nuestro ResultSet tan solo nos queda recorrerlos y tratarlos como estemos oportuno.
 - ResultSet tiene implementado un puntero que apunta al primer elemento de la lista.
 - Mediante el método next() el puntero avanza al siguiente elemento.
 - El método next(), además de avanzar al siguiente registro, devuelve true en caso de existir más registros y false en caso de haber llegado al final.
 - Mediante los métodos getString() y getDouble() vamos obteniendo los valores de las diferentes columnas. Estos métodos reciben el nombre de la columna como parámetro.

```
while(resultado.next()) {  
    String nif= resultado.getString("nif");  
    String nombre= resultado.getString("nombre");  
    String apellidos= resultado.getString("Apellidos");  
    Double salario= resultado.getDouble("salario");  
    System.out.println(nif+ " "+ nombre + " "+apellidos+" "+salario);  
}
```

5. Acceso a DB mediante JDBC

Funcionamiento JDBC

5. **Liberar recursos** → Por último debemos liberar todos los recursos utilizados para garantizar la correcta ejecución del programa.

```
conexion.close();  
sentencia.close();  
resultado.close();
```

Ejecución de sentencias DML

- En SQL, las sentencias DML son aquellas que nos permiten trabajar con los datos de una BBDD.
 - **SELECT** - para obtener datos de una base de datos.
 - **INSERT** - para insertar datos a una tabla.
 - **UPDATE** - para modificar datos existentes dentro de una tabla.
 - **DELETE** - elimina todos los registros de la tabla; no borra los espacios asignados a los registros.
- Para ejecutar cualquiera de estas sentencias debemos usar objetos:
 - **Statement** → Permite ejecutar sentencias SQL sin parámetros
 - **PreparedStatement** → Permite ejecutar sentencias SQL con parámetros

Ejecución de sentencias DML

- La interface Statement
 - La interface **Statement** permite crear objetos Statement, al tratarse de una interface no podemos instanciar directamente este tipo de objetos sino que debemos valernos del método `createStatement()` de la clase `Connection`.
 - Los principales métodos de esta interface son:

Método	Descripción
<code>ResultSet executeQuery(String query)</code>	Se utiliza para ejecutar sentencias SQL que recuperan datos. Devuelve un objeto <code>ResultSet</code> con los datos recuperados.
<code>int executeUpdate(String Query)</code>	Se utiliza para ejecutar sentencias de manipulación como Insert, Update y Delete. Devuelve un entero indicando el número de registros que se han visto afectados.
<code>boolean execute(String Query)</code>	Se puede utilizar para ejecutar cualquier consulta SQL. En caso de que la sentencia devuelva un <code>ResultSet</code> , el método <code>execute()</code> devuelve true y debemos recuperar el objeto <code>ResultSet</code> a través del método <code>getResultSet()</code> . En caso contrario (Cualquier otro tipo de sentencia) nos devolverá false y debemos usar el método <code>getUpdateCount()</code> para recuperar el valor devuelto.

6. Ejecución de sentencias DML

La interface PreparedStatement

- Es muy habitual el uso de variables dentro de una sentencia SQL:
 - Valores a insertar, actualizar o borrar
 - Filtros en consultas de selección.
 - Etc.
- Para este tipo de consultas con parte variable debemos hacer uso de las llamadas Sentencias Preparadas (Prepared Statements)
- La interface PreparedStatement nos va a permitir crear sentencias SQL placeholders (marcadores de posición) que representan los datos que serán agregados posteriormente (Variables). Estos placeholders se representan mediante signos de interrogación (?)

```
String insert="insert into empleados values (?,?,,?)";
```


Ejecución de sentencias DML

- La interface PreparedStatement
 - Cada Placeholder tiene un índice, siendo 1 el primer elemento que se encuentre en la cadena.
 - Antes de ejecutar una PreparedStatement es necesario asignar valores a cada uno de los placeholders.
 - La ventaja que nos ofrece esto frente a la sentencias tradicionales es el hecho de poder preparar consultas (precompilar) una sola vez y tener la posibilidad de ejecutarlas tantas veces como sea necesario con diferentes valores de entrada. Ofrecen una gran flexibilidad.
 - Los principales métodos de PreparedStatement son:

MÉTODO	DESCRIPCIÓN
ResultSet executeQuery()	Similar a su homónima en la interface Statement
int executeUpdate()	Similar a su homónima en la interface Statement
boolean execute()	Similar a su homónima en la interface Statement

Ejecución de sentencias DML

- La interface PreparedStatement
 - Además, contamos con una serie de métodos par asignar valores a cada uno de los placeholders.

Método	Tipo SQL
void setString(int index, String valor)	VARCHAR
void setBoolean(int índice, boolean valor)	BIT
void setByte(int índice, byte valor)	TINYINT
void setShort(int índice, short valor)	SMALLINT
void setInt(int indice, int valor)	INTEGER
void setLong(int indice, int valor)	BIGINT
void setFloat(int índice, float valor)	FLOAT
void setDouble(int índice, double valor)	DOUBLE
void setBytes(int índice, byte[] valor)	VARBINARY
void setDate(int índice, Date valor)	DATE
void setTime(int índice, Time valor)	TIME

Ejecución de sentencias DML

- La interface PreparedStatement

- Para asignar valores NULL a los placeholders debemos usar
 - `setNull(int indice, int tipoSQL)`
 - Donde tipoSQL es cualquiera de los tipos definidos en `java.sql.Types`.

```
String insert="insert into empleados values (?,?,,?)";
PreparedStatement sentenciaPreparada=conexion.prepareStatement(insert);
sentenciaPreparada.setString(1, "0000A");
sentenciaPreparada.setString(2, "Paco");
sentenciaPreparada.setString(3, "Perez");
sentenciaPreparada.setFloat(4, 12.1f);
int valorConsulta= sentenciaPreparada.executeUpdate();
System.out.println(valorConsulta);
```

Ejecución de sentencias DML

- La clase ResultSet
 - Mediante un objeto de la clase ResultSet podemos recoger los valores devueltos por una consulta de selección.
 - A través de un objeto ResultSet podemos acceder al valor de cualquier columna de la tupla actual a través de su posición o su nombre.
 - También podemos obtener información general sobre la consulta (Como el número de columnas, su tipo, etc) mediante el método **getMetadata()**.
 - Para acceder a cualquiera de las columnas de la tupla actual haremos uso de los métodos getXXX().

6. Ejecución de sentencias DML

La clase ResultSet

Método	Tipo Java
getString(int numeroCol)	String
getBoolean(int numeroCol)	boolean
getByte(int numeroCol)	byte
getShort(int numeroCol)	short
getInt(int numeroCol)	int
getLong(int numeroCol)	long
getFloat(int numeroCol)	float
getDouble(int numeroCol)	double
getBytes(int numeroCol)	byte[]
getDate(int numeroCol)	Date
getTime(int numeroCol)	Time
getTimestamp(int numeroCol)	Timestamp

Método	Tipo Java
getString(String nombreCol)	String
getBoolean(String nombreCol)	boolean
getByte(String nombreCol)	byte
getShort(String nombreCol)	short
getInt(String nombreCol)	int
getLong(String nombreCol)	long
getFloat(String nombreCol)	float
getDouble(String nombreCol)	double
getBytes(String nombreCol)	byte[]
getDate(String nombreCol)	Date
getTime(String nombreCol)	Time
getTimestamp(String nombreCol)	Timestamp

Ejecución de sentencias DML

- La clase ResultSet
 - Para recorrer un ResultSet haremos uso del método next() como ya vimos en ejemplos anteriores.

```
while(resultado.next()) {  
    String nif= resultado.getString("nif");  
    String nombre= resultado.getString("nombre");  
    String apellidos= resultado.getString("Apellidos");  
    Double salario= resultado.getDouble("salario");  
    System.out.println(nif+" "+ nombre  +" "+apellidos+" "+salario);  
}
```

Ejecución de sentencias DDL

- Aunque el grueso de las operaciones que cualquier operaciones que se realizan desde una aplicación cliente contra una BBDD sea operaciones de manipulación de datos, pueden darse casos en los que nos veamos obligados a realizar operaciones de definición.
 - Acabamos de instalar una aplicación en un nuevo equipo y necesitamos crear, de forma automatizada, una BBDD local para su funcionamiento (El caso más habitual).
 - Tras una actualización de software la estructura de la BBDD ha cambiado y debemos actualizarla desde nuestra aplicación Java.
 - Desconocemos la estructura de la BBDD y queremos realizar consultas dinámicas sobre la misma.
- En este apartado estudiaremos los mecanismos y procedimientos necesarios para solventar cualquiera de estas situaciones.

Ejecución de sentencias DDL

- Definición y modificación de estructuras

- Las sentencias DDL tradicionales no dejan de ser sentencias SQL y, por tanto, las podremos ejecutar de la forma que ya hemos estudiado a través de la interfaz

Statement/PreparedStatement.

```
Class.forName("com.mysql.jdbc.Driver");
Connection conexion=
    DriverManager.getConnection("jdbc:mysql://localhost", "root", "");
Statement sentencia= conexion.createStatement();
String sql= "Sentencia DDL";
int resultado= sentencia.executeUpdate(sql);
System.out.println(resultado);
conexion.close();
sentencia.close();
```


Ejecución de sentencias DDL

- **Definición y modificación de estructuras**

- Podemos hacer uso del modificador IF NOT EXISTS dentro de nuestro código SQL para asegurarnos de que la base de datos se crea antes de iniciar los procesos CRUD de nuestra aplicación.

```
Class.forName("com.mysql.jdbc.Driver");
Connection conexion=
    DriverManager.getConnection("jdbc:mysql://localhost/animales", "root", "");
Statement sentencia= conexion.createStatement();
String sql= "Create table if not exists mascotas("
    + "codigo varchar(3),"
    + "nombre varchar(20),"
    + "Constraint mascotas_PK PRIMARY KEY(codigo)"
    + ");";
int resultado= sentencia.executeUpdate(sql);
System.out.println(resultado);
conexion.close();
sentencia.close();
```

Ejecución de sentencias DDL

- **Definición y modificación de estructuras**

- A través de este mecanismo podríamos crear bases de datos y sus correspondientes estructuras (Tablas, restricciones, vistas, índices...).
- Es una práctica muy habitual cuando vamos a usar una BBDD local, en la actualidad es muy empleado en aplicaciones para dispositivos móviles, aplicaciones locales de gestión, gestores de contenido web, etc.
- Incluso, a través de este diseño podríamos crear un cliente de bases de datos similar a MySQLWorkbench o el propio HeidiSQL.

Ejecución de sentencias DDL

- **Definición y modificación de estructuras**

- En ocasiones puede darse el caso de que no conozcamos la estructura de una BBDD, por suerte esta información se encuentra almacenada en los denominados metaobjetos de la base de datos.
- La interface DatabaseMetaData proporciona información sobre la base de datos a través de múltiples métodos de los cuales es posible obtener una gran cantidad de información.

Método	Descripción
ResultSet getTables()	Proporciona información sobre diferentes tipos de objeto de la base de datos
ResultSet getColumns()	Devuelve información sobre las columnas de una tabla
ResultSet getPrimaryKeys()	Devuelve las claves primarias.
ResultSet getExportedKeys()	Devuelve las claves ajenas que apuntan a una tabla determinada.
ResultSet getImportedKeys()	Devuelve las claves ajenas de una tabla.

- **Definición y modificación de estructuras**

- ResultSet getTables(String catálogo, String esquema, String patronDeTabla, String tipos)
 - Catálogo → Catálogo de la BBDD al que estamos apuntando. Con null indicamos que estamos apuntando al actual.
 - Esquema → Esquema del cual queremos obtener información. Con null indicamos que queremos información del esquema actual.
 - Patrón → Permite refinar la búsqueda indicando mediante un patrón los nombres o parte de los nombres que estamos buscando. Similar al funcionamiento de la cláusula like.
 - Tipos → Debemos indicar que tipos de objeto queremos obtener, estos tipos pueden ser: TABLE, VIEW, SYSTEM TABLE, GLOBAL TEMPORARY, LOCAL TEMPORARY, ALIAS y SYNONYM.

Ejecución de sentencias DDL

Ejecución de sentencias DDL

- Definición y modificación de estructuras

- ResultSet getTables(String catálogo, String esquema, String patronDeTabla, String tipos)

```
DatabaseMetaData dbmd= conexion.getMetaData();
String tipos[]= {"TABLE", "VIEW"};
ResultSet tablas=dbmd.getTables(null, null, null, tipos);
while(tablas.next()){
    System.out.println(
        tablas.getString("TABLE_CAT")+
        tablas.getString("TABLE_SCHEM")+
        tablas.getString("TABLE_NAME")+
        tablas.getString("TABLE_TYPE"));
}
```

Acceso a datos puente JDBC-ODBC

- **Definición y modificación de estructuras**

- Existen algunos productos (los menos) que no cuentan con acceso JDBC y ten solo son accesibles mediante el protocolo ODBC.
- Para acceder a los datos almacenados en este tipo de productos usamos un puente denominado **ODBC-JDBC Bridge**.
- El ODBC-JDBC Bridge es un controlador JDBC que traduce las operaciones JDBC que ya conocemos a operaciones compatibles con ODBC.
- Este puente está implementado en Java dentro del paquete `sun.jdbc.odbc` por lo que no es necesario añadir ninguna librería externa para trabajar con él.

8. Acceso a datos puente JDBC-ODBC

Definición y modificación de estructuras

- Para crear una conexión ODBC en Java simplemente debemos importar el Driver y crear la conexión de la forma habitual.




















```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");  
Connection conexion= DriverManager.getConnection("jdbc:odbc:Mysql-odbc");
```

- En el ejemplo, cuando creamos la conexión, estamos apuntando a un origen de datos odbc llamado "Mysql-odbc" que previamente habremos creado en nuestro servidor de bases de datos.

Informes con JasperReports

- Una de las principales funciones de cualquier sistema informático es la manipulación de datos y, en gran medida, esa manipulación está orientada a la obtención de información con un formato y un significado determinado.
- Los informes (reports) son uno de los formatos más extendidos a la hora de presentar dicha información.
- Un informe es básicamente un documento imprimible que contiene información.
 - Facturas
 - Resúmenes de ventas.
 - Reservas de hotel
 - Billetes de avión
 - Etc.

Informes con JasperReports

- ▶  jasperreports-6.10.0.jar
- ▶  jfreechart-1.0.19.jar
- ▶  javax.inject-1.jar
- ▶  jcommon-1.0.23.jar
- ▶  castor-core-1.4.1.jar
- ▶  castor-xml-1.4.1.jar
- ▶  commons-beanutils-1.9.3.jar
- ▶  commons-collections-3.2.2.jar
- ▶  commons-collections4-4.2.jar
- ▶  commons-digester-2.1.jar
- ▶  commons-lang3-3.4.jar
- ▶  commons-logging-1.1.1.jar
- ▶  ecj-4.4.2.jar
- ▶  itext-2.1.7.js7.jar
- ▶  jackson-annotations-2.9.9.jar
- ▶  jackson-core-2.9.9.jar
- ▶  jackson-databind-2.9.9.jar
- ▶  bcprov-jdk15on-1.62.jar
- ▶  mysql-connector-java-5.1.48.jar

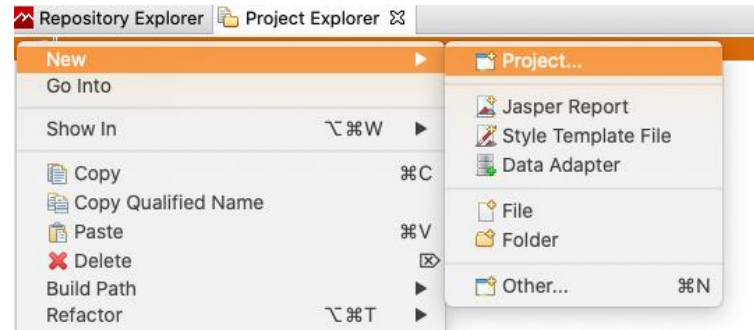
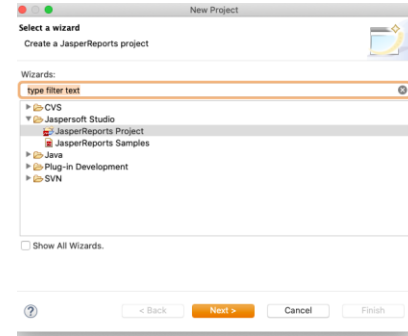
- Una de las herramientas más extendidas para la generación de informes en Java es JasperReports.
- JasperReports es un conjunto de librerías de código abierto y licencia GPL.
- Es capaz de generar informes en distintos formatos: PDF, HTML, XLS, RTF, ODT, CSV, TXT y HTML.
- Para poder usar estas librerías lo primero que debemos hacer es importar en nuestro proyecto Java todas las dependencias de JasperReports y el conector JDBC para poder acceder a la BBDD.
- https://jar-download.com/?search_box=jasperreports

Informes con JasperReports

- Una vez tenemos el proyecto Java preparado vamos a crear nuestra plantilla, para ello debemos crear un fichero JasperReport (Archivo JRXML).
 - Se trata de un fichero con formato XML propio de Jasperreports, por tanto, es un fichero de texto plano que podríamos crear con cualquier editor de texto.
 - Sin embargo, Jasperreports nos proporciona una potente herramienta de creación de informes: Jasper Studio (<https://community.jaspersoft.com/project/jaspersoft-studio/releases>).
 - Esta aplicación nos permite crear un report mediante un cómodo editor WYSIWYG.
- https://download.cnet.com/Jaspersoft-Studio-64-bit/3001-2383_4-75748747.html

Informes con JasperReports

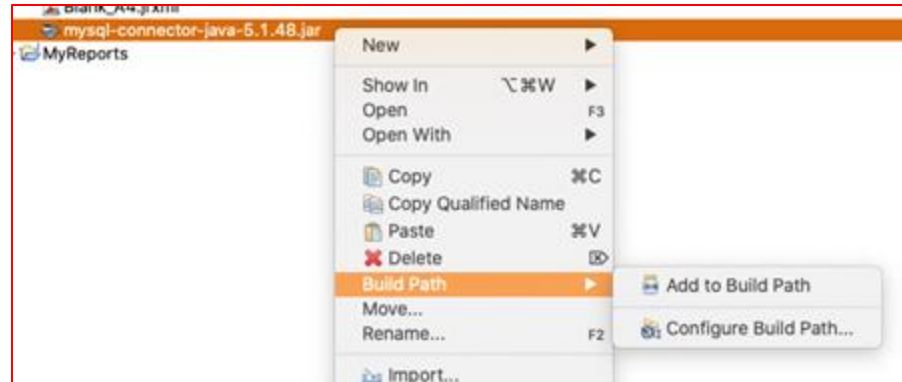
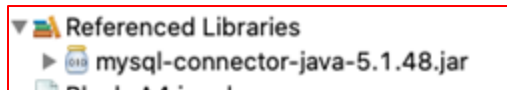
- Jasper Studio
 - Una vez instalada la aplicación lo primero que haremos será crear un nuevo Proyecto del mismo modo que lo hacemos en Eclipse



9. Informes con JasperReports

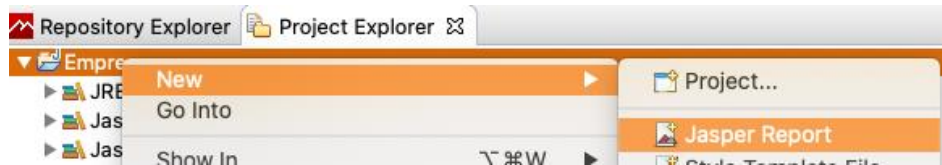
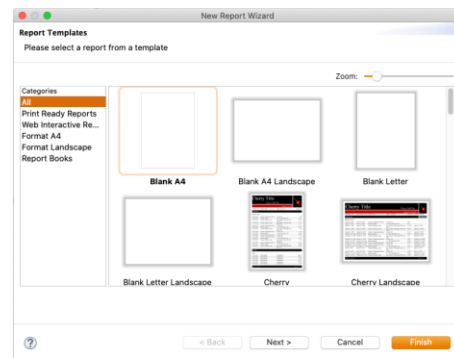
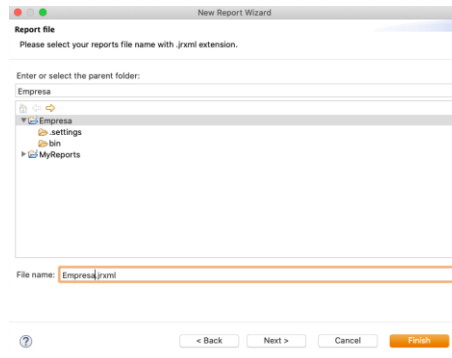
Jasper Studio

- Una vez creado el proyecto, vamos a importar el Driver JDBC de Mysql para poder hacer la conexión a la BBDD, esto se hace exactamente igual que en Eclipse, copiando el fichero .jar dentro del proyecto y añadiéndolo al Build Path.



Informes con JasperReports

- Jasper Studio
 - Ya tenemos el proyecto preparado para empezar a trabajar, vamos a crear nuestro primer informe, para ello hacemos clic derecho sobre el proyecto → new → Jasper Report.
 - Seleccionamos la plantilla A4 vacío, el proyecto al que corresponderá y le damos un nombre.



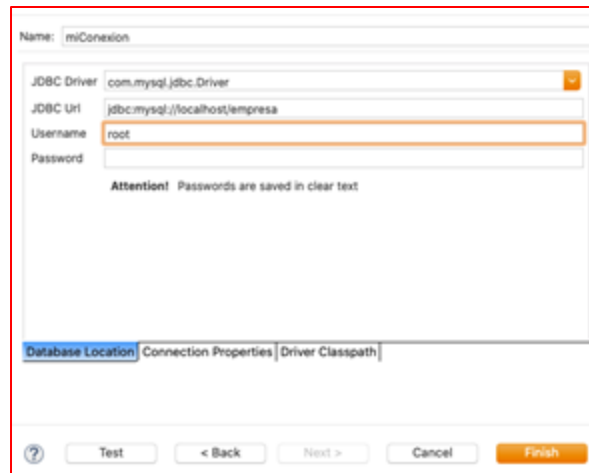
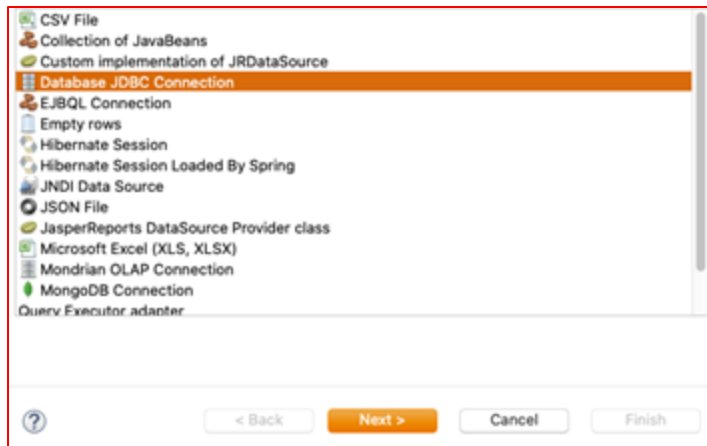
9. Informes con JasperReports

Jasper Studio

- El siguiente paso es indicar cuál será nuestro origen de datos, para ello crearemos un nuevo Data Adapter pulsando sobre el botón new.

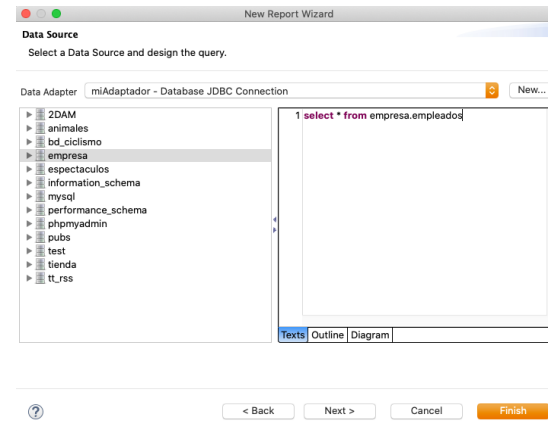
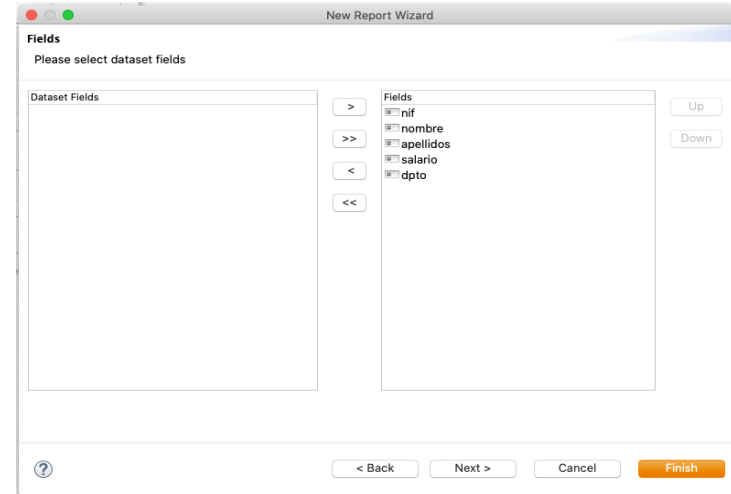


- Seleccionamos la opción JDBC Connection y la configuramos con los parámetros de nuestra conexión.



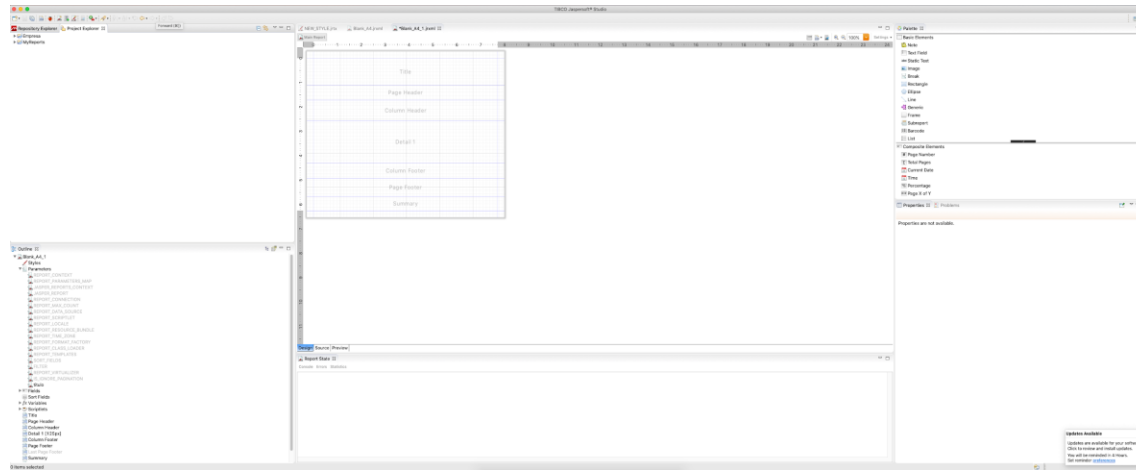
Informes con JasperReports

- Jasper Studio
 - Escribimos la consulta para obtener los datos que incluirá nuestro informe y seleccionamos aquellos campos que queremos que aparezcan.



Informes con JasperReports

- Jasper Studio
 - Una vez seguidos estos pasos nos encontramos con un sencillo editor para crear nuestro informe.



Informes con JasperReports

- Proyecto Java
 - Una vez hemos terminado de editar nuestro informe, copiamos el archivo jrxml dentro de nuestro proyecto Java.
 - Para poder trabajar con Jasper Reports en un proyecto Java importamos las siguientes clases:
 - El siguiente código es un ejemplo de las principales acciones que podemos realizar con nuestro informe.

```
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;
```

Informes con JasperReports

- Proyecto Java

```
//Ruta del fichero .jrxml
String reportResource= "./Blank_A4.jrxml";
//Rutas de los ficheros de salida
String reportHTML="./informe.html";
String reportPDF="./informe.PDF";
String reportXML="./informe.xml";
//Creamos un hashmap para los posibles parámetros de entrada (Si los hubiera)
Map<String, Object> params= new HashMap<String, Object>();
params.put("titulo", "");
params.put("autor", "");
params.put("fecha", LocalDate.now().toString());
//Compilamos el report
JasperReport miReport= JasperCompileManager.compileReport(reportResource);
//Creamos la conexión con la BBDD
Class.forName("com.mysql.jdbc.Driver");
Connection con= DriverManager.getConnection("jdbc:mysql://localhost/empresa", "root", "");
//Rellenamos el informe pasandole un objeto JasperReport, los parametros (o null) y la conexión
JasperPrint miInforme= JasperFillManager.fillReport(miReport, params, con);
JasperViewer.viewReport(miInforme, false);
//Convertir a HTML
JasperExportManager.exportReportToHtmlFile(miInforme, reportHTML);
//Convertir a PDF
JasperExportManager.exportReportToPdfFile(miInforme, reportPDF);
//Convertir a XML
JasperExportManager.exportReportToXmlFile(miInforme, reportXML, false);
```