
	RAMA:	Informática	CICLO:	Desenvolvemento de Aplicacions Multiplataforma				
	MÓDULO:	Acceso a datos					CURSO:	2º
	PROTOCOLO:	Apuntes clases	AVAL:	2	DATA:	2022/2023		
	UNIDAD		COMPETENCIA:					

Tema 4: Manejo de Conectores

Índice

1. Introducción.....	1
2. JDBC	1
2.1. Acceso a un SGSB mediante conectores JDBC	3
3. Realización de Consultas	5
3.1. executeQuery	6
3.2. executeUpdate.....	7
4. Sentencias preparadas	8
4.1. MariaDB/MySQL	8
4.2. Java	9
5. Ejecución de procedimientos almacenados	10
6. Obtención de información de la BD y de un ResultSet.	11
6.1. DatabaseMetaData:	11
6.2. ResultSetMetadata.....	12
7. Gestión de transacciones	12
8. El desfase Objeto-Relacional	13
9. Bases de datos embebidas: SQLite.	14
9.1. Conexión a SQLite	16
9.2. Diferencias entre SQLite y MariaDB/MySQL.....	16
9.3. Enlaces de interés	17
9.4. Editores gráficos para SQLite.....	18
10. Anexo 1: Drivers JDBC	18
11. Anexo 2: Equivalencia entre tipos de datos	19
12. Bibliografía	20

	RAMA:	Informática	CICLO:	Desenvolvemento de Aplicacions Multiplataforma				
	MÓDULO:	Acceso a datos					CURSO:	2º
	PROTOCOLO:	Apuntes clases	AVAL:	2	DATA:	2022/2023		
	UNIDAD		COMPETENCIA:					

1. Introducción

Para la conexión a una base de datos, desde un lenguaje de programación, se utilizan conectores. Un conector, también conocido como driver, es una colección de clases que se encargan de encapsular tanto el proceso de conexión como el proceso de consultas ofreciendo una API común, de alto nivel, que es independiente al sistema gestor de bases de datos que queramos utilizar.

Estos conectores pueden ser, generalmente, de dos tipos:

- Específicos: son protocolos propios del SGBD, con una API propia, que solo pueden ser utilizados por este SGBD.
- Genéricos: son protocolos que incluyen una capa software intermedia, el conector, que oculta los detalles concretos de cada base de datos ofreciendo una API común que se encarga de traducir las peticiones genéricas en código nativo del SGBD.

Cuando un programador necesita conectarse a una base de datos utiliza para ello esta API común, proporcionada por el conector, sin tener en cuenta la implementación concreta de la base de datos usada (da igual que sea Oracle, MySQL, MariaDB, SQLite, ...), por lo que una misma aplicación, cambiando de conector, puede usar distintos SGBD.

Ejemplos de conectores son:

- ODBC (Open DataBase Connectivity): es un estándar creado por Microsoft y soportado por distintos lenguajes de programación
- JDBC (Java DataBase Connectivity): creado por Sun (más tarde comprado por Oracle) como una versión del driver ODBC específicamente para Java.
- JDBC-ODBC: es un driver puente en Java que permite utilizar la API JDBC en SGBD que aún no poseen un conector JDBC pero si poseen un conector ODBC.

2. JDBC¹

Para conectarnos desde Java a una base de datos utilizaremos un conector JDBC.

En JDBC existen cuatro tipos de conectores². La denominación de estos controladores está asociada a un número de 1 a 4 y viene determinada por el grado de independencia respecto de la plataforma, prestaciones, etc.

- Tipo 1. Driver puente JDBC-ODBC: se traducen las llamadas de JDBC a invocaciones ODBC a través de librerías ODBC del sistema operativo.
- Tipo 2. API nativo/parte Java: controlador escrito parcialmente en Java y en código nativo. La aplicación Java hace una llamada a la base de datos a través del driver JDBC y este traduce la petición a invocaciones a la API del fabricante de la base de datos.

¹ https://es.wikipedia.org/wiki/Java_Database_Connectivity

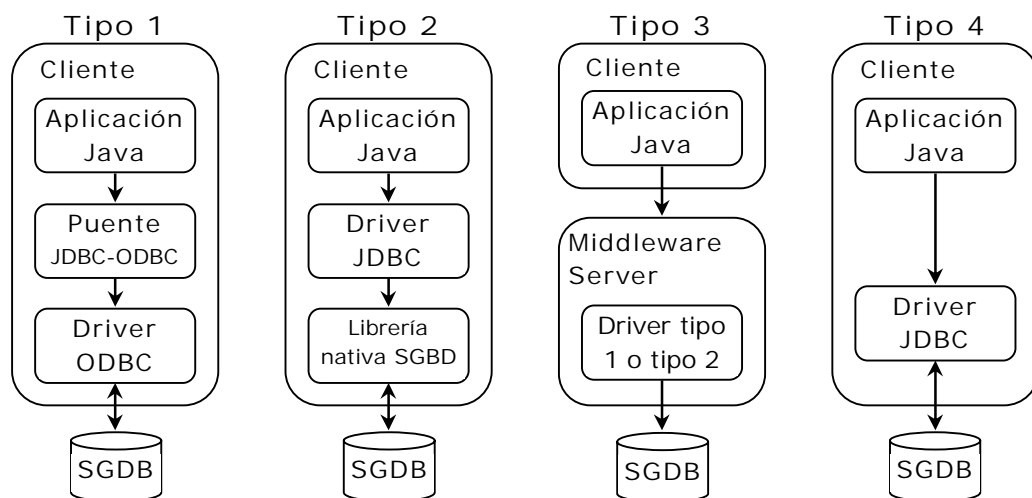
² <https://www.ibm.com/docs/es/i/7.3?topic=jdbc-types-drivers>

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	Desenvolvemento de Aplicacions Multiplataforma				
	MÓDULO:	Acceso a datos					CURSO:	2º
	PROTOCOLO:	Apuntes clases		AVAL:	2	DATA:	2022/2023	
	UNIDAD		COMPETENCIA:					

- Tipo 3. Network: controlador Java puro que utiliza un protocolo TCP/IP de red para comunicarse con un servidor de base de datos. Las solicitudes de la base de datos JDBC están traducidas en un protocolo de red independiente de la base de datos y dirigidas a un middleware server. El middleware server recibe las solicitudes y las envía a la base de datos utilizando para ello un driver JDBC del tipo 1 o del tipo 2
- Tipo 4. Thin: controlador de Java puro con protocolo nativo. Traduce llamadas JDBC en el protocolo de la red usado por SGBD directamente. Esto permite llamadas directas desde la máquina cliente al servidor SGBD. Se necesitara un driver distinto para cada base de datos y estos suelen ser suministrados por los fabricantes. Este tipo de driver es el que se trabajará en este tema.

Donde:

- Los tipos 1 y 2 se usan normalmente cuando no existen drivers JDBC disponibles para el SGBD y hay que usar un driver ODBC. Exigen la instalación de software en el puesto cliente.
- Los tipos 3 y 4 son la mejor forma para acceder a bases de datos JDBC siendo el tipo 4 es más eficiente.



COLEXIO VIVAS <small>S.L.</small>	RAMA:	Informática	CICLO:	Desenvolvemento de Aplicacions Multiplataforma				
	MÓDULO:	Acceso a datos					CURSO:	2º
	PROTOCOLO:	Apuntes clases	AVAL:	2	DATA:	2022/2023		
	UNIDAD		COMPETENCIA:					

2.1. Acceso a un SGSB mediante conectores JDBC

Los pasos que se deben realizar para conectarse a un SGBD utilizando conectores JDBC son:

1. Cargar el driver JDBC: mediante la instrucción `Class.forName`³. En SGBD como pueden ser MySQL o MariaDB al añadir el JAR con el driver este paso se realiza de forma automática.

Los siguientes ejemplos se muestra su utilización para la conexión a los SGBD MariaDB, MySQL y SQLite respectivamente (entre comillas, se especifica la clase del driver a cargar).

```
Class.forName("org.mariadb.jdbc.Driver");
Class.forName("com.mysql.jdbc.Driver");
Class.forName("org.sqlite.JDBC");
```

Se puede consultar una lista de clases disponibles en el apéndice 1.

Como cada SGBD utiliza un driver propio se necesita, como paso previo, importar el fichero JAR con el driver al proyecto o introducirlo en la CLASSPATH. Estos se pueden descargar desde:

- MariaDB: <https://mariadb.com/kb/es/acerca-de-mariadb-connectorj/>
- MySQL: <https://dev.mysql.com/downloads/connector/>

2. Una vez el driver está cargado se necesita crear una conexión contra la base de datos. Para ello se utiliza un objeto `Connection` del paquete `java.sql`. Cada SGBD tiene una URL de conexión distinta.

Ejemplos de creación de un objeto de tipo `Connection` son:

```
Connection conexion = DriverManager.getConnection
    ("jdbc:mariadb://equipo/baseDeDatos", "usuario", "contraseña");

Connection conexion = DriverManager.getConnection
    ("jdbc:mysql://equipo/ baseDeDatos ", "usuario", "contraseña");

Connection conexion = DriverManager.getConnection
    ("jdbc:sqlite:/rutaBaseDeDatos/baseDeDatos.db");
```

Se puede consultar una lista de URL de conexiones en el apéndice 1.

3. Una vez creada la conexión se puede crear y ejecutar consultas contra la base de datos.
4. Una vez ejecutada la consulta se pueden obtener los datos devueltos por la consulta.
5. Cuando se termina de usar la conexión con la base de datos esta se ha de cerrar para liberar recursos.

³ <https://es.stackoverflow.com/questions/76160/por-qué-es-necesario-usar-class-fornamecom-mysql-jdbc-driver>

<div>COLEXIO</div> <div>VIVAS S.L.</div>	RAMA:	Informática	CICLO:	Desenvolvemento de Aplicacions Multiplataforma				
	MÓDULO:	Acceso a datos					CURSO:	2º
	PROTOCOLO:	Apuntes clases		AVAL:	2	DATA:	2022/2023	
	UNIDAD		COMPETENCIA:					

El siguiente método nos permite abrir una conexión con una base de datos:

Ejemplo 1: Creación de una conexión contra una base de datos

```

1 public class JDBC {
2     private Connection conexion;
3     public void abrirConexion(String bd, String servidor , String usuario,
4                               String password) {
5         try {
6             String url = String.format("jdbc:mariadb://%s:3306/%s", servidor, bd);
7             // Establecemos la conexión con la BD
8             this.conexion = DriverManager.getConnection(url, usuario, password);
9             if (this.conexion != null) {
10                 System.out.println ("Conectado a "+bd+" en "+servidor);
11             } else {
12                 System.out.println ("No conectado a "+bd+" en "+servidor);
13             }
14         } catch (SQLException e) {
15             System.out.println("SQLException: " + e.getLocalizedMessage());
16             System.out.println("SQLState: " + e.getSQLState());
17             System.out.println("Código error: " + e.getErrorCode());
18         }
19     }
20 }

```

Donde:

- Línea 5: Creamos la URL de conexión. Esta se compone de las siguientes partes:
 - jdbc:mariadb → Base de datos a la que nos vamos a conectar y con ello el driver a cargar.
 - servidor → Nombre o dirección IP del servidor a conectarse.
 - :3306 → Número del puerto en el que está corriendo la base de datos. Si la base de datos se usa el puerto por defecto no hace falta especificarlo.
 - bd → nombre de la base de datos a que queremos conectarnos.
- Línea 7: Utilizando DriverManager⁴, el JAR con el driver y la URL de conexión se crea un objeto de tipo Connection (perteneciente a java.sql) que nos permitirá acceder a la base de datos. Tiene los siguientes parámetros (estos dependerán del SGDB al que deseamos conectarnos):
 - Url de conexión.
 - Nombre del usuario de la bases de datos con el que nos vamos a conectar.
 - Contraseña del usuario.

⁴ <https://docs.oracle.com/javase/8/docs/api/java/sql/DriverManager.html>

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	Desenvolvemiento de Aplicacions Multiplataforma				
	MÓDULO:	Acceso a datos				CURSO:	2º	
	PROTOCOLO:	Apuntes clases		AVAL:	2	DATA:	2022/2023	
	UNIDAD		COMPETENCIA:					

- Línea 10: si se produce un error durante la conexión a la base de datos se produce una excepción (SQLException⁵) que ha de ser tratada. Se puede obtener información del error producido mediante:
 - getLocalizedMessage: mensaje genérico con el error producido.
 - getErrorCode: código de error interno del SGBD.
 - getSQLState: código de error estándar SQLSTATE. Viene del ANSI SQL y de ODBC .

Como se comentaba en el paso 5, una vez se termina de usar la conexión con la base de datos esta ha de ser cerrada, mediante el método close de connection, para liberar recursos.

Ejemplo 2: Se cierra la conexión de la base de datos

```

1 public void cerrarConexion () {
2     try {
3         this.conexion.close();
4     } catch (SQLException e) {
5         System.out.println("Error al cerrar la conexión: "+e.getLocalizedMessage());
6     }
7 }
```

3. Realización de Consultas

El primer paso para la realización de consultas en la base de datos es la creación de un objeto de tipo Statement⁶ el cual se encargará de enviar la consulta a la base de datos usando la conexión que hemos creado en el paso anterior. El ejemplo de definición es el siguiente:

```
Statement st=this.conexion.createStatement();
```


Statement dispone de los dos siguientes métodos para ejecutar consultas:

- executeQuery → Consultas que devuelven filas. Son aquellas sentencias que se utilizan para recuperar datos de la base de datos: SELECT o CALL. Estas consultas devolverán un objeto de tipo ResultSet con las filas que se han obtenido de la consulta.
- executeUpdate → Consultas que no devuelve filas. Son las sentencias que se engloban dentro del lenguaje de manipulación de datos o DML (INSERT, UPDATE y DELETE) y del lenguaje de definición de datos o DDL (CREATE, DROP y ALTER). Estas consultas devolverán un entero que indica el número de filas afectadas por la consulta.

El objeto Statement creado se ha de cerrar, mediante su método close, cuando se finalice su uso.

⁵ <https://docs.oracle.com/javase/8/docs/api/java/sql/SQLException.html>

⁶ <https://docs.oracle.com/javase/8/docs/api/java/sql/Statement.html>

	RAMA:	Informática	CICLO:	Desenvolvemento de Aplicacions Multiplataforma				
	MÓDULO:	Acceso a datos					CURSO:	2º
	PROTOCOLO:	Apuntes clases		AVAL:	2	DATA:	2022/2023	
	UNIDAD		COMPETENCIA:					

3.1. executeQuery

ExecuteQuery permite ejecutar consultas que devuelven un conjunto ordenado de filas, como registros, dentro de un objeto de tipo ResultSet^{7 8}.

Para recuperar los datos desde el objeto ResultSet utilizaremos un cursor interno que apunta a la fila actual de datos, al principio el cursor esta posicionado antes de la primera fila. El método next permite avanzar el cursor a la siguiente fila devolviendo falso cuando no queden más filas para obtener. Por esto motivo usando next y un bucle while se pueden obtener todas las filas del ResultSet.

Por defecto un objeto ResultSet no es actualizable y solo permite avanzar pero dispone de métodos que permiten mover en cursor dentro del ResultSet.

Una vez que, usando el método next, hemos desplazado el cursor podemos acceder a los datos de la fila actual (que es a la que apunta el cursor) mediante el uso de los métodos getter que dispone: getInt, getBoolean, getString, ... JDBC intenta convertir los datos devueltos al tipo Java especificado en el método getter (deben ser compatibles). Estos métodos aceptan dos tipos de parámetros para indicar la columna de la que recuperar los datos.

- String: nombre de la columna desde la que se recuperan los datos. Si varias columnas comparten nombre se recupera la primera. No se distingue entre mayúsculas y minúsculas.
- Int: Posición de la columna dentro del resultado obtenido. La primera columna tiene el número 1.

En el siguiente ejemplo consultaremos datos de la tablas aulas (tabla usada en los ejercicios):

```

Ejemplo 3: Consulta de datos
1  public void consultaAlumnos(String bd) {
2      abrirConexion("add", "localhost", "root", "");
3
4      // El Statement se cierra solo al salir de catch
5      try (Statement stmt=this.conexion.createStatement()){
6          // Consulta a ejecutar
7          String query="select * from aulas";
8
9          // Se ejecuta la consulta
10         ResultSet rs= stmt.executeQuery(query);
11
12         //Mientras queden filas en rs (el método next devuelve true) recorreremos las filas
13         while (rs.next ()) {
14             // Se obtiene datos en función del número de columna o de su nombre
15             System.out.println (rs.getInt(1) + "\t" +
16                 rs.getString("nombreAula") + "\t" + rs.getInt("puestos")); //
17         }
18     } catch (SQLException e) {
19         System.out.println("Se ha producido un error: " +e.getMessage());
20     } finally {
21         cerrarConexion(); // Se cierra la conexión
22     }
23 }

```

⁷ <https://docs.oracle.com/javase/8/docs/api/java/sql/ResultSet.html>

⁸ https://es.wikipedia.org/wiki/Result_Set

<div>COLEXIO</div> <div>VIVAS S.L.</div>	RAMA:	Informática	CICLO:	Desenvolvemento de Aplicacions Multiplataforma				
	MÓDULO:	Acceso a datos					CURSO:	2º
	PROTOCOLO:	Apuntes clases		AVAL:	2	DATA:	2022/2023	
	UNIDAD		COMPETENCIA:					

```

19     }
20 }

```

3.2. executeUpdate

Como ya se comento executeUpdate se utiliza para aquellas sentencias DDL y DML que no devuelven filas de datos, sino un entero con las filas que se han visto afectadas por la sentencia.

En los siguientes ejemplos veremos cómo añadir datos a una tabla así como modificar la estructura de la tabla añadiéndole una columna.

Ejemplo 4: Inserción de datos

```

1  public void insertarFila(){
2      try (Statement sta = this.conexion.createStatement()){
3          String query="INSERT INTO aulas VALUES (5, 'Física', 23), (6, 'Química', 34)";
4          // Se ejecuta la sentencia de inserción mediante executeUpdate
5          int filasAfectadas=sta.executeUpdate(query);
6          System.out.println("Filas insertadas: "+filasAfectadas);
7      } catch (SQLException e) {
8          System.out.println("Se ha producido un error: "+e.getLocalizedMessage());
9      }
10 }

```

Podemos observar que executeUpdate no devuelve un objeto ResultSet sino un entero.

El siguiente ejemplo muestra cómo se modifica una tabla añadiéndole una nueva columna:

Ejemplo 5: Modificación de la estructura de una tabla

```

1  public void addColumna(){
2      Statement sta = null;
3      try {
4          sta = this.conexion.createStatement() ;
5          String query="ALTER TABLE aulas MODIFY nombreAula VARCHAR(25) DEFAULT NULL";
6          // Se ejecuta la modificación de la tabla
7          int filasAfectadas=sta.executeUpdate(query);
8          System.out.println("Filas afectadas: "+filasAfectadas);
9      } catch (SQLException e) {
10         System.out.println("Se ha producido un error: "+e.getLocalizedMessage());
11     } finally {
12         if (sta!=null) {
13             try {
14                 sta.close(); // Se cierra el Statement
15             } catch (SQLException e) {
16                 e.printStackTrace();
17             }
18         }
19     }
20 }
21 }

```


COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	Desenvolvemento de Aplicacions Multiplataforma				
	MÓDULO:	Acceso a datos					CURSO:	2º
	PROTOCOLO:	Apuntes clases		AVAL:	2	DATA:	2022/2023	
	UNIDAD		COMPETENCIA:					

4. Sentencias preparadas

4.1. MariaDB/MySQL

Las sentencias preparadas o prepared statement son un mecanismo, presentes en casi todos los SGDB, que permiten la optimización de consultas mediante plantillas. Se utilizan en aquellas consultas que se ejecutan de forma repetida en la base de datos.

En el momento de la definición de la sentencia se realiza parte del proceso de verificación, compilación y optimización de la sentencia (estos pasos se realizan una única vez) y es en el momento de la ejecución de la sentencia cuando se rellenan los "huecos" de la plantilla con los datos necesarios evitándose realizar, en cada sentencia ejecutada, los tres pasos anteriores. Por el contrario si una sentencia preparada se ejecuta una o muy pocas veces puede ser menos eficiente que una sentencia normal.

Las sentencias preparadas evitan, además, los problemas de inyecciones SQL.

Por ejemplo si deseamos insertar varias aulas nuevas sobre la tablas aulas podemos ejecutar las sentencias siguientes:


```
insert into aulas (numero, nombreaula, puestos) values (20, 'Informatica', 30),
insert into aulas (numero, nombreaula, puestos) values (31, 'Aula Magna', 56);
```

O ejecutar una sentencia preparada de la siguiente manera:

```
PREPARE insertar FROM "INSERT INTO aulas VALUES ( ?, ?, null)";
SET @numero=20;
SET @nombreAula='Informática';
EXECUTE insertar USING @numero,@nombreAula;
SET @numero=31;
SET @nombreAula='Aula Magna';
EXECUTE insertar USING @numero,@nombreAula;
DEALLOCATE PREPARE insertar;
```

En la primera línea se define la sentencia preparada con nombre insertar. Esta sentencia preparada realiza una inserción de datos en la tabla aulas. Se puede observar que los dos primeros valores a insertar se indican con un ? indicando que sus valores se especificarán en el momento de la ejecución de la sentencia.

Mediante el comando execute se ejecuta la sentencia asignando, de forma ordenada los valores necesarios. En la primera ? se insertará el valor contenido en la variable @numero y en la segunda ? el valor contenido en la variable @nombreAula.

	RAMA:	Informática	CICLO:	Desenvolvemento de Aplicacions Multiplataforma				
	MÓDULO:	Acceso a datos					CURSO:	2º
	PROTOCOLO:	Apuntes clases		AVAL:	2	DATA:	2022/2023	
	UNIDAD		COMPETENCIA:					

4.2. Java

En java para poder usar sentencias preparadas deberemos, en el momento de crear la conexión, añadir el parámetro `?useServerPrepStmts=true`⁹ a la URL de conexión. Si la base de datos no soporta procedimientos almacenados este parámetro no se usa.

```
Connection conexion = DriverManager.getConnection(
    "jdbc:mariadb://localhost:3306/ad?useServerPrepStmts=true","usuario", "contraseña");
```

En este momento el objeto de tipo Connection acepta sentencias preparadas.

Veamos el siguiente ejemplo:

Ejemplo 6: Consulta preparada

```

1  private PreparedStatement ps=null; // atributo de instancia
2  public void consultaAlumnosPS(String patron, int numResultados )
3                                     throws SQLException{
4      String query="select * from aulas where nombreAula like ? limit ?";
5      if (this.ps == null) this.ps=this.conexion.prepareStatement(query);
6      ps.setString(1, patron);
7      ps.setInt(2,numResultados);
8      ResultSet resu= ps.executeQuery();
9      while (resu.next ()) {
10         System.out.println (resu.getInt(1) + "\t" + resu.getString("nombreAula"));
11     }
12 }
```

Donde:

- Línea 1: la ventaja de los procedimientos almacenados es crear la consulta parametrizada una única vez. Para ello usamos un atributo de instancia para almacenar la consulta y no tener que crearla de nuevo cada vez queramos ejecutar la sentencia.
- Línea 3: definimos la consulta a ejecutar. En esta caso tenemos dos valores, indicados con ?, a los que se asignará su valor en el momento de la ejecución de la sentencia. Solo se puede usar ? para representar datos y no se puede usar con nombres de tablas o columnas.
- Línea 4: si la consulta preparada no fue creada antes se crea. Se puede ver que desde el objeto Connection no se crea un Statement sino un prepareStatement al cual se le pasa la consulta de la línea uno.
- Líneas 5 y 6: mediante setters se le asignan valores a las ?. El primer parámetro es el número de ? a la que se le va a asignar el valor. El segundo parámetro es el valor a asignar. El tipo de dato que se asigna debe ser compatible con el valor esperado en la sentencia.
- Línea 7: Se ejecuta la consulta. Desde este punto el tratamiento de los datos, ya sea el resultado un ResultSet o un int se hace de forma idéntica a una consulta normal.

⁹ <http://dev.mysql.com/doc/connector-j/en/connector-j-reference-configuration-properties.html>

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	Desenvolvemento de Aplicacions Multiplataforma				
	MÓDULO:	Acceso a datos					CURSO:	2º
	PROTOCOLO:	Apuntes clases		AVAL:	2	DATA:	2022/2023	
	UNIDAD		COMPETENCIA:					

5. Ejecución de procedimientos almacenados^{10 11}

CallableStatement permite la ejecución de procedimientos (y funciones) almacenadas desde Java. Su uso es similar al uso de los preparedStatement pero en este caso se usara el método prepareCall de connection para crear la consulta parametrizada. Este método tiene como parámetro la cadena con la consulta a realizar que, como mínimo, debe contener el nombre del procedimiento almacenado y en caso de este poseer argumentos todos estos.

Al igual que con la ejecución de consultas preparadas se puede usar ? para establecer valores en tiempo de ejecución.

Imaginemos el siguiente procedimiento almacenado creado en MariaDB/MySQL:

```
DELIMITER //
CREATE PROCEDURE getAulas (IN num INT , IN cad VARCHAR(10))
BEGIN
    SELECT * FROM aulas WHERE puestos >= num
                                AND nombreAula LIKE CONCAT("%",cad,"%");
END //
DELIMITER ;
```

En el siguiente código se muestra un ejemplo de ejecución del procedimiento almacenado llamado getAulas (num, cad). Este procedimiento devuelve todas las aulas que tenga igual o más puestos que num y en el nombre del aula contenga la cadena cad.

Ejemplo 7: Ejecución de un procedimiento almacenado

```
1 CallableStatement cs = this.conexion.prepareCall("CALL getAulas(?,?)");
2 // Se proporcionan valores de entrada al procedimiento
3 cs.setInt(1, 10);
4 cs.setString(2, "o");
5 ResultSet resultado = cs.executeQuery();
6 while (resultado.next ()) {
7     System.out.println(resultado.getInt(1)+"\t"+
8         resultado.getString("nombreAula")+"\t"+resultado.getInt("puestos"));
9 }
```

¹⁰ <https://www.arquitecturajava.com/jdbc-prepared-statement-y-su-manejo/>

¹¹ <https://docs.oracle.com/javase/8/docs/api/java/sql/PreparedStatement.html>

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	Desenvolvemento de Aplicacions Multiplataforma				
	MÓDULO:	Acceso a datos					CURSO:	2º
	PROTOCOLO:	Apuntes clases		AVAL:	2	DATA:	2022/2023	
	UNIDAD		COMPETENCIA:					

6. Obtención de información de la BD y de un ResultSet.

JDBC dispone de dos métodos que nos permiten obtener información de la base de datos:

6.1. DatabaseMetaData¹²:

DatabaseMetaData permite obtener información, a través de varios métodos, de los distintos elementos de la base de datos: bases de datos, tablas, vistas, procedimientos almacenados, claves primarias y foráneas, ...

Para obtener un objeto de este tipo se usa el método getMetada de Connection. El siguiente ejemplo muestra información sobre las tablas que pertenecen a una base de datos pasadas como parámetro.

Ejemplo 8: Obtención de información de la base de datos

```

1 public void getInfo(String bd){
2     DatabaseMetaData dbmt;
3     ResultSet tablas, columnas;
4     try {
5         dbmt = this.conexion.getMetaData();
6         tablas=dbmt.getTables(bd, null, null, null);
7         while (tablas.next()){
8             System.out.println(String.format("%s %s",
9                 tablas.getString("TABLE_NAME"), tablas.getString("TABLE_TYPE")));
10            columnas=dbmt.getColumns(bd, null,
11                tablas.getString("TABLE_NAME"), null);
12            while (columnas.next()){
13                System.out.println(String.format("  %s %s %d %s %s",
14                    columnas.getString("COLUMN_NAME"),
15                    columnas.getString("TYPE_NAME"),
16                    columnas.getInt("COLUMN_SIZE"),
17                    columnas.getString("IS_NULLABLE"),
18                    columnas.getString("IS_AUTOINCREMENT")
19                ));
20            }
21        }
22    } catch (SQLException e) {
23        System.out.println("Error obteniendo datos "+e.getLocalizedMessage());
24    }
25 }

```

¹² <https://docs.oracle.com/javase/8/docs/api/java/sql/DatabaseMetaData.html>

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	Desenvolvemento de Aplicacions Multiplataforma				
	MÓDULO:	Acceso a datos					CURSO:	2º
	PROTOCOLO:	Apuntes clases		AVAL:	2	DATA:	2022/2023	
	UNIDAD		COMPETENCIA:					

6.2. ResultSetMetadata¹³

Permite obtener información de un ResultSet, es decir, de las filas devueltas por una consulta. Su funcionamiento es idéntico a DatabaseMetaData pero en este caso la información se obtiene del ResultSet.

En el siguiente ejemplo obtendremos información de una consulta.

Ejemplo 9: Obtención de información de una consulta

```

1 public void getInfoConsulta(String consulta) throws SQLException{
2     Statement st=this.conexion.createStatement();
3     ResultSet filas=st.executeQuery(consulta);
4     ResultSetMetaData rsmd=filas.getMetaData();
5     System.out.println("Num\tNombre\tAlias\tTipoDatos");
6     for (int i=1;i<=rsmd.getColumnCount();i++){
7         System.out.println(String.format("%d \t %s \t %s \t %s", i,
8             rsmd.getColumnLabel(i),
9             rsmd.getColumnLabel(i),
10            rsmd.getColumnTypeName(i)
11        ));
12    }
13 }
```

7. Gestión de transacciones¹⁴

Una transacción en un SGBD es un conjunto de instrucciones que se tienen que ejecutar de forma atómica, es decir, o se ejecutan todas o no se ejecuta ninguna. Normalmente deben cumplir con las propiedades ACID¹⁵ ¹⁶ (atomicidad, consistencia, aislamiento y durabilidad) garantizando que la base de datos se mantiene en un estado consistente de tras su ejecución.

Toda transacción termina con una de las siguientes sentencias:

- Commit: se confirman los cambios producidos por la transacción.
- Rollback: se aborta la transacción deshaciéndose los cambios producidos por esta.
- Sentencias que realizan un commit de forma implícita: alter table, truncate table, drop index, create table, create database, start transaction / begin, rename table, drop database, lock tables, drop table, create index, set autocommit=1.

Por defecto, y tal como se vio el año pasado, MariaDB/MySQL tiene activado el auto-commit. Por lo que cuando se realiza una inserción o modificación en la base de datos estos cambios son confirmados automáticamente.

¹³ <https://docs.oracle.com/javase/8/docs/api/java/sql/ResultSetMetaData.html>

¹⁴ <https://docs.oracle.com/javase/tutorial/jdbc/basics/transactions.html>

¹⁵ <https://es.wikipedia.org/wiki/ACID>

¹⁶ <https://www.arquitecturajava.com/transacciones-acid-y-sus-propiedades/|Transacciones>

<div>COLEXIO</div> <div>VIVAS S.L.</div>	RAMA:	Informática	CICLO:	Desenvolvemento de Aplicacions Multiplataforma				
	MÓDULO:	Acceso a datos					CURSO:	2º
	PROTOCOLO:	Apuntes clases		AVAL:	2	DATA:	2022/2023	
	UNIDAD		COMPETENCIA:					

Para usar transacciones se deberá deshabilitar el auto-commit implícito y ejecutar las sentencias que deseemos incluir en la transacción. Tras su ejecución, si no se ha producido algún error, se ejecuta commit para confirmar los cambios realizados. En caso contrario se debe ejecutar un rollback para deshacer estos cambios.

Veamos un ejemplo de su uso:

Ejemplo 10: Transacciones en java

```

1  public void insertarConTransaccion () {
2      try {
3          this.conexion.setAutoCommit(false);
4          Statement st = this.conexion.createStatement();
5          st.executeUpdate("INSERT INTO aulas VALUES (5, 'Física', 23)");
6          st.executeUpdate("INSERT INTO aula VALUES (6, 'Química', 34)");
7          // Si no se produce ningún error se confirman los cambios
8          this.conexion.commit();
9      } catch (SQLException e) {
10         System.out.println("Se ha producido un error: "+e.getLocalizedMessage());
11         try{
12             if (this.conexion!=null) {
13                 System.out.println("Se deshacen los cambios mediante un rollback");
14                 // Si se ha producido un error los cambios se deshacen
15                 this.conexion.rollback();
16             }
17         } catch (SQLException e2) {
18             System.out.println("Error en el rollback: "+e.getLocalizedMessage());
19         }
20     }
21 }

```


8. El desfase Objeto-Relacional

El desfase objeto-relacional surge debido a la diferente naturaleza de los lenguajes orientados a objetos que se utilizan para construir aplicaciones y las bases de relacionales que se utilizan para almacenar los datos usados por estas.

Cuando una aplicación desea almacenar un objeto, estructura básica en los lenguajes orientados a objetos, debe realizar una conversión de este objeto a una tabla, que es la estructura básica de almacenamiento en el modelo relacional.

El desfase objeto-relacional son los problemas que surgen, entre otros aspectos, en la realización de esta conversión de estructuras de almacenamiento, de la correcta asignación de tipos de datos entre el lenguaje de programación y el SGBD, de la necesidad de conocer dos lenguajes distintos, ...

Estos aspectos provocan que la complejidad en el desarrollo de los programas aumente y con ello el esfuerzo en la programación. Todo ello conlleva que el código desarrollado aumente.

	RAMA:	Informática	CICLO:	Desenvolvemento de Aplicacions Multiplataforma				
	MÓDULO:	Acceso a datos					CURSO:	2º
	PROTOCOLO:	Apuntes clases		AVAL:	2	DATA:	2022/2023	
	UNIDAD		COMPETENCIA:					

9. Bases de datos embebidas: SQLite¹⁷.

Cuando desarrollamos pequeñas aplicaciones, en las que no vamos a almacenar grandes cantidades de información, podemos utilizar una base de datos embebidas, en las cuales el motor esta incrustado en la aplicación y es exclusivo para ella. La base de datos se inicia cuando se ejecuta la aplicación y termina cuando se cierra la aplicación.

SQLite es un sistema gestor de base de datos relacional multiplataforma escrito en C que implementa un sistema autónomo, sin servidor y sin necesidad de configuración. Es un proyecto de dominio público.

Algunas características de SQLite son:

- No tiene un proceso servidor independiente.
- La base de datos completa se almacena en un único archivo y es multiplataforma.
- Toda la base de datos ocupa muy poco espacio.
- Muy rápido en ciertas condiciones.
- Es autónomo, es decir no tiene dependencias externas.
- No es necesario ninguna configuración o administración.
- Tiene soporte para transacciones cumpliendo las propiedades ACID.

Para instalar SQLite podemos descargar el fichero ZIP `sqlite-tools-XXXXXXX.zip` desde la página <http://www.sqlite.org/download.html> (desde Linux se puede instalar mediante el comando `apt-install sqlite3`), el cual al descomprimirse se obtendrá, entre otros el fichero ejecutable con el SGBD (`sqlite3.exe` para Windows y `sqlite` para Linux). Podemos ejecutarlo desde la línea de comandos añadiéndole como parámetro una base de datos SQLite: si la base de datos existe de abre para su uso y si no existe se crea vacía.

El siguiente ejemplo crea la tabla `alumnos` con tres filas de datos en la base de datos `ejemplo.db` situada en la carpeta actual (la extensión de la base de datos es opcional):

```
sqlite3 ejemplo.db


BEGIN TRANSACTION;

CREATE TABLE alumnos (
  cod int(5) not null primary key,
  nombre varchar(25),
  direccion varchar(30)
);

INSERT INTO ALUMNOS VALUES
  (1,'juan', 'Vigo'),(2,'Paco', 'Lugo'),(3,'Laura', 'Ourense');

COMMIT;
```

¹⁷ <https://www.sqlite.org/index.html>


	RAMA:	Informática	CICLO:	Desenvolvemento de Aplicacions Multiplataforma				
	MÓDULO:	Acceso a datos					CURSO:	2º
	PROTOCOLO:	Apuntes clases		AVAL:	2	DATA:	2022/2023	
	UNIDAD		COMPETENCIA:					

Algunos comandos útiles del SGBD son:

- `.help`: Muestra la ayuda del cliente.
- `.tables`: Ver tablas de una base de datos SQLite.
- `.schema [nombreTabla]`: Muestra la sentencia de creación de las tablas de la base de datos o de la tabla especificada por nombreTabla.
- `.show`: Muestra información sobre la configuración de la base de datos.
- `.read fichero`: Carga el fichero con código SQL.
- `.separator cad`: Personaliza la separación de columnas en la presentación de los datos.
- `.headers [on|off]`: Muestra o oculta las cabeceras en la presentación de los datos.
- `.mode [tipo]`: Cambia el modo de presentar los datos. Tipo puede ser: `ascii`, `csv`, `column`, `html`, `insert`, `line`, `list`, `quote`, `tabs`, `tcl`.
- `.width [anchoColumna1 [anchoColumna2 anchoColumna3]]`: Nos permite definir los anchos de visualización de las columnas.
- `.dump [tabla]`: Realiza un volcado de las tablas (o tabla si se especifica una) junto con sus datos.
- `.quit` o `.exit`: Sale de SQLite y vuelve a consola.
- `attach bd`: Permite añadir bases de datos al cliente SQLite; por ejemplo: `attach database 'bd' as e1;`
- `.database`: Muestra las bases de datos.

Comparación de algunos comandos del cliente de consola entre SQLite y MariaDB/MySQL:

Tarea	SQLite	MySQL/MariaDB
Conectarse a una base de datos	<code>sqlite3/sqlite <file></code>	<code>mysql <nombreBD></code>
Ayuda del cliente	<code>.help</code>	<code>help contents</code>
Ayuda SQL	n/a	<code>help contents</code>
Listar bases de datos	<code>.databases</code>	<code>SHOW DATABASES;</code>
Cambiar de base de datos	n/a	<code>USE <dbname></code>
Listar tabla	<code>.tables</code>	<code>SHOW TABLES;</code>
Mostar información de una tabla	<code>.schema <tablename></code>	<code>DESCRIBE <tablename>;</code>
Importar datos	<code>.import <file> <table></code>	<code>LOAD DATA INFILE '<file>'</code>
Exportar datos	<code>.dump <table></code>	<code>SELECT ... INTO OUTFILE '<file>'</code>
Salir del cliente	<code>.quit</code> o <code>.exit</code>	<code>quit</code> o <code>exit</code>

	RAMA:	Informática	CICLO:	Desenvolvemento de Aplicacions Multiplataforma				
	MÓDULO:	Acceso a datos					CURSO:	2º
	PROTOCOLO:	Apuntes clases		AVAL:	2	DATA:	2022/2023	
	UNIDAD		COMPETENCIA:					

9.1. Conexión a SQLite

Para conectarnos a SQLite necesitamos el driver JDBC para SQLite. Desde la URL <https://github.com/xerial/sqlite-jdbc> podemos descargarnos el JAR `sqlite-jdbc-x.x.x.jar` con el driver. Al igual que con el driver JDBC para MariaDB/MySQL tenemos dos opciones para usar el driver. Añadirlo al proyecto o incluirlo en el classpath.

Para usar el driver deberemos cargarlo y crear una conexión a la base de datos:


```
Class.forName ("org.sqlite.JDBC");
String url="jdbc:sqlite:/rutaBaseDeDatos/baseDeDatos.db";
Connection conexion = DriverManager.getConnection(url);
```

Una vez establecida la conexión, para trabajar con la bases de datos SQLite se podrán usar los mismos comandos que los usados para el driver JDBC de MariaDB/MySQL.

9.2. Diferencias entre SQLite y MariaDB/MySQL

Al crear un proyecto nuevo nos podemos preguntar que SGBD se adapta mejor al mismo: SQLite o MariaDB/MySQL. Estas preguntas pueden ayudar a escoger el SGBD correcto:

- ¿Hay un límite de espacio de almacenamiento?
 - La biblioteca SQLite ocupa 250kb, que es perfecto para los dispositivos embebidos debido a que no tienen mucho espacio de almacenamiento.
- ¿Velocidad?
 - SQLite es generalmente más rápido que MariaDB/MySQL bajo ciertas circunstancias.
- Acceso remoto
 - MariaDB/MySQL es accesible de forma remota, SQLite no.
- ¿Se necesita que la base de datos sea portable?
 - La base de dato SQLite se guardan directamente en un solo archivo, lo que facilita que puedan ser movidas o copiadas.
 - Una base de datos MariaDB/MySQL, por norma general, no se puede copiar u mover directamente.
- ¿Son necesarios aspectos de seguridad y autenticación en la base de datos?
 - SQLite no proporciona ni gestión de usuarios ni sistema de autenticación. La seguridad se basa en el sistema de permisos de ficheros establecido por el sistema operativo.
 - MariaDB/MySQL gestiona usuarios con diferentes niveles de acceso y requiere un nombre de usuario y contraseña para conectarse a ella.
- ¿Pueden escalarse?
 - SQLite no tiene capacidad de escalamiento.
 - MariaDB/MySQL se puede escalar.

	RAMA:	Informática	CICLO:	Desenvolvemento de Aplicacions Multiplataforma				
	MÓDULO:	Acceso a datos				CURSO:	2º	
	PROTOCOLO:	Apuntes clases		AVAL:	2	DATA:	2022/2023	
	UNIDAD		COMPETENCIA:					

- ¿Se puede optimizar el rendimiento?
 - En SQLite no hay mejoras en el rendimiento. No realiza caché de consultas, no optimiza las selecciones, etc.
 - MariaDB/MySQL tiene opciones para la optimización del rendimiento.
- ¿Limitaciones de escritura?
 - SQLite solo permite una operación de escritura al mismo tiempo.
 - MariaDB/MySQL, dependiendo del tipo de tabla, restringe las escrituras al nivel de tabla o de fila lo que puede aumentar la concurrencia.
- Mayúsculas, minúsculas y acentos
 - SQLite al comparar la igualdad de dos cadenas (=) distingue entre mayúsculas y minúsculas o entre vocales acentuadas o sin acentuar. En cambio si se busca si dos cadenas son similares (like) no distingue entre mayúsculas y minúsculas aunque sí que lo hace entre vocales acentuadas o sin acentuar
 - MariaDB/MySQL no distingue entre mayúsculas y minúsculas o entre vocales acentuadas o sin acentuar.
- Longitud de los campos
 - SQLite: Si se guarda un dato más largo que el tamaño del campo, SQLite lo almacena en su totalidad.
 - MariaDB/MySQL trunca el dato a la longitud del campo.
 - Por ejemplo, si se crea una tabla con un campo nombre VARCHAR(4) y se guarda la cadena abcdef, en MariaDB/MySQL se guardará la cadena abcd, mientras que en SQLite se guardará la cadena abcdef.
- Fecha vacía
 - SQLite guarda una cadena vacía al insertar una fecha vacía.
 - MariaDB/MySQL guarda la cadena "0000-00-00".

En este enlace <http://www.sqlite.org/whentouse.html> se puede ver, según el creador del SQLite las situaciones donde es y no es recomendable usar SQLite.

9.3. Enlaces de interés

- FAQ: <https://www.sqlite.org/faq.html>
- Acerca de: <https://www.sqlite.org/about.html>
- Quickstart: <https://www.sqlite.org/quickstart.html>

<div>COLEXIO</div> <div>VIVAS</div> <div>S.L.</div>	RAMA:	Informática	CICLO:	Desenvolvemento de Aplicacions Multiplataforma				
	MÓDULO:	Acceso a datos					CURSO:	2º
	PROTOCOLO:	Apuntes clases		AVAL:	2	DATA:	2022/2023	
	UNIDAD		COMPETENCIA:					


9.4. Editores gráficos para SQLite

Algunos de los editores gráficos que podemos usar para trabajar con SQLite son:

Editor	Web	Soporte
DBbeaver	https://dbeaver.io/	Soporte para Linux, Windows y Mac.
SQLiteStudio	http://sqlitestudio.pl	Soporte para Linux, Windows y Mac.
DB Browser for SQLite	http://sqlitebrowser.org/	Soporte para Windows y Mac.
SQLiteSpy	http://www.yunqa.de/delphi/doku.php/products/sqlitespy/index	Soporte para Windows.
SQLiteManager	http://www.sqlabs.com/sqlitemanager.php	Soporte para Windows y Mac.
SQLite Administrator	http://sqliteadmin.orbmu2k.de/	Soporte para Windows.


10. Anexo 1: Drivers JDBC

Bases de datos	Driver Class	JDBC URL
mariadb	org.mariadb.jdbc.Driver	jdbc:mariadb://host:puerto/bd
mysql	com.mysql.jdbc.Driver	jdbc:mysql://host:puerto/bd
sqlite	org.sqlite.JDBC	jdbc:sqlite:/ruta/bd
access	net.ucanaccess.jdbc.UcanaccessDriver	jdbc:ucanaccess://ruta/bd
h2	org.h2.Driver	jdbc:h2:/ruta/bd
db2	com.ibm.db2.jcc.DB2Driver	jdbc:db2://host:puerto/bd
apache derby	org.apache.derby.jdbc.EmbeddedDriver	jdbc:derby:/ruta/bd
frontbase	com.frontbase.jdbc.FBJDriver	jdbc:frontbase://host/bd
firebird	org.firebirdsql.jdbc.FBDriver	jdbc:firebirdsql://host:puerto/bd
hsqldb	org.hsqldb.jdbc.JDBCDriver	jdbc:hsqldb:file:/ruta/bd
sqlserver	com.microsoft.sqlserver.jdbc.SQLServerDriver	jdbc:sqlserver://host:puerto;database=bd
oracle	oracle.jdbc.driver.OracleDriver	jdbc:oracle:thin:@host:puerto:bd
pointbase	com.pointbase.jdbc.jdbcUniversalDriver	com.pointbase.me.jdbc.jdbcdriver
postgresql	org.postgresql.driver	jdbc:postgresql://host:puerto/bd
sybase	com.sybase.jdbc3.jdbc.SybDriver	jdbc:sybase:tds:host:puerto/bd
informix	com.informix.jdbc.IfxDriver	jdbc:informix-sqli://host:puerto/bd:informixserver=dbservername

	RAMA:	Informática	CICLO:	Desenvolvemento de Aplicacions Multiplataforma				
	MÓDULO:	Acceso a datos					CURSO:	2º
	PROTOCOLO:	Apuntes clases	AVAL:	2	DATA:	2022/2023		
	UNIDAD		COMPETENCIA:					

11. Anexo 2: Equivalencia entre tipos de datos

Tipo SQL	Tipo Java	Descripción
BINARY	byte[]	Matriz de bytes. Se utiliza para objetos binarios grandes.
BIT	boolean	Booleano. Valor 0 / 1
CHAR	String	Cadena de caracteres de longitud fija
DATETIME	java.sql.Date	Fecha y hora como: yyyy-mm-dd hh:mm:ss
DECIMAL	java.math.BigDecimal	De precisión arbitraria con signo y números decimales
FLOAT	double	Número de coma flotante, mapeado a double
INTEGER	int	Valores enteros de 32 bits
LONGVARBINARY	byte[]	Cadena de caracteres de longitud variable
LONGVARCHAR	String	Cadena de caracteres de longitud variable
NTEXT	String	Cadena de caracteres grande
REAL	float	Número de coma flotante, mapeado a float
SMALLINT	short	Valores enteros de 16 bits
TIME	java.sql.Time	Thin wrapper around java.util.Date
TIMESTAMP	java.sql.Timestamp	Composite of a java.util.Date and a separate nanosecond value
VARBINARY	byte[]	Matriz de bytes
VARCHAR	String	Cadena de caracteres de longitud variable

	RAMA:	Informática	CICLO:	Desenvolvemento de Aplicacions Multiplataforma				
	MÓDULO:	Acceso a datos					CURSO:	2º
	PROTOCOLO:	Apuntes clases		AVAL:	2	DATA:	2022/2023	
	UNIDAD		COMPETENCIA:					

12. Bibliografía

1. [JDBC](#)
2. [Tipos de drives JDBC](#)
3. Establecer conexion al SGBD
 1. [Class.forName](#)
 2. [DriverManager](#)
 3. [SQLException](#)
 4. [Statement](#)
 5. [ResultSet](#)
 6. [ResultSet wikipedia](#)
4. [Sentencias preparadas](#)
5. Ejecución de procedimientos almacenados
 1. [JDBC PreparedStatement-y-su-manejo](#)
 2. [PreparedStatement](#)
6. Metadatos
 1. [DatabaseMetaData](#)
 2. [ResultSetMetaData](#)
7. Transacciones
 1. [Transacciones](#)
 2. [Propiedades](#)
 3. [Transacciones acid y sus propiedades](#)
8. SQLite
 1. [HomePage](#)
 2. [Download](#)
 3. [Appropriate Uses For SQLite](#)
 4. [FAQ](#)
9. Conectores JDBC
 1. [MariaDB](#)
 2. [MySQL](#)
 3. [SQLite](#)