UT10. BASES DE DATOS

Módulo: PROGRAMACIÓN

Curso 2022/2023. 1° DAM

Ruth Lospitao Ruiz

INTRODUCCIÓN

- Como se ha visto en la Unidad de Trabajo anterior, Java proporciona las herramientas suficientes para realizar una aplicación en la cual se utilice archivos para almacenar información que genera una aplicación.
- Sin embargo, la utilización de archivos conlleva diversos problemas como la redundancia, la dificultad de mantenimiento, la rigidez en las búsquedas, la seguridad, etc.
- Para paliar estos problemas surgieron las bases de datos, y la herramienta que Java nos proporciona para poder manipular la información que hay contenida en las BD es el API JDBC (Java DataBase Connectivity)
- Este API de Java nos permite ejecutar sentencias SQL

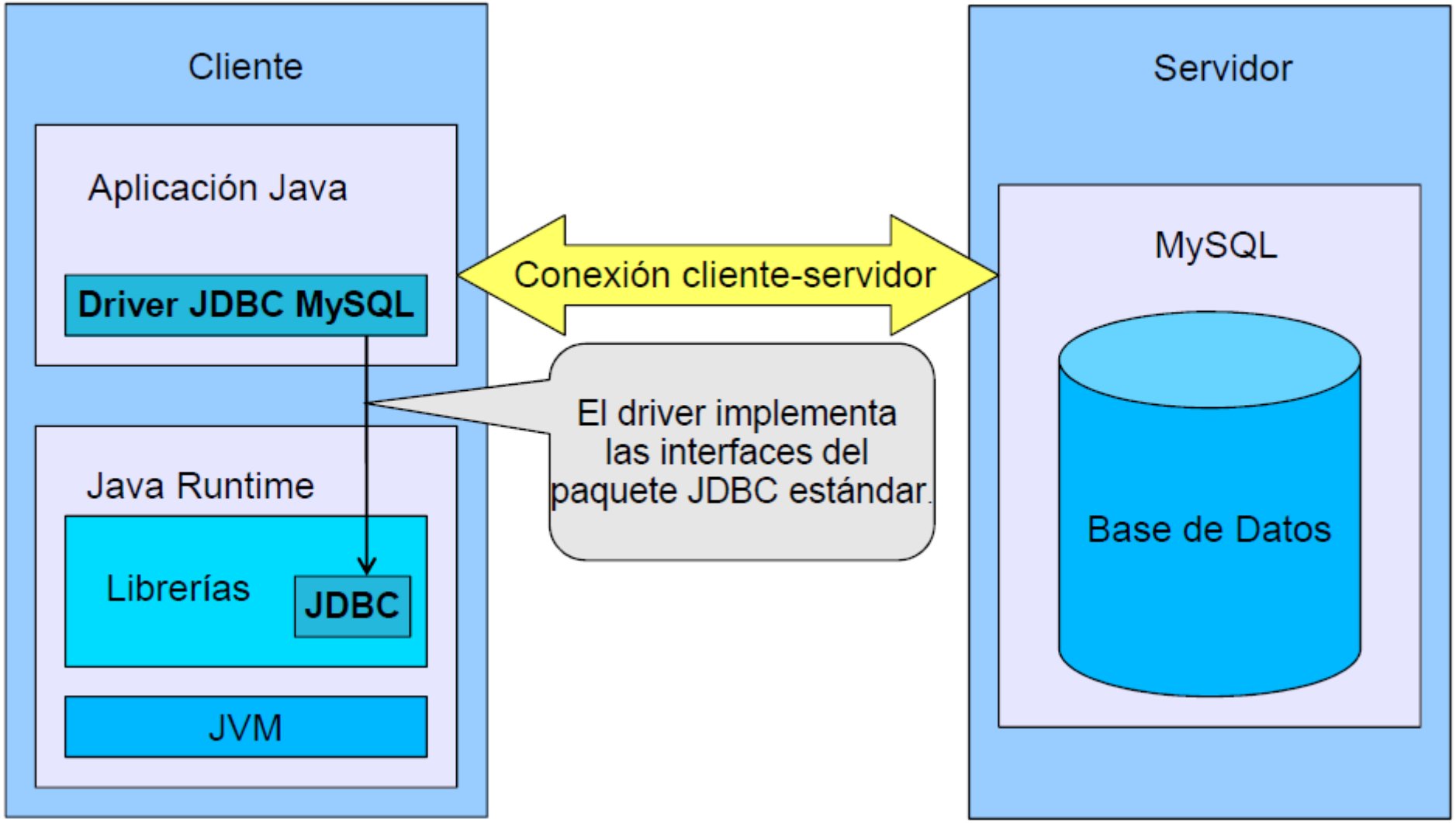
INTRODUCCIÓN

- Una aplicación podrá hacer uso de un BD si establece una conexión con la misma.
- Para ello, deberá usar un driver, que será el encargado de convertir el lenguaje de alto nivel SQL a sentencias entendibles por el sistema gestor de bases de datos.
- Una vez establecida la conexión con la BD podrá enviar sentencias SQL a esa BD y procesar los resultados obtenidos.

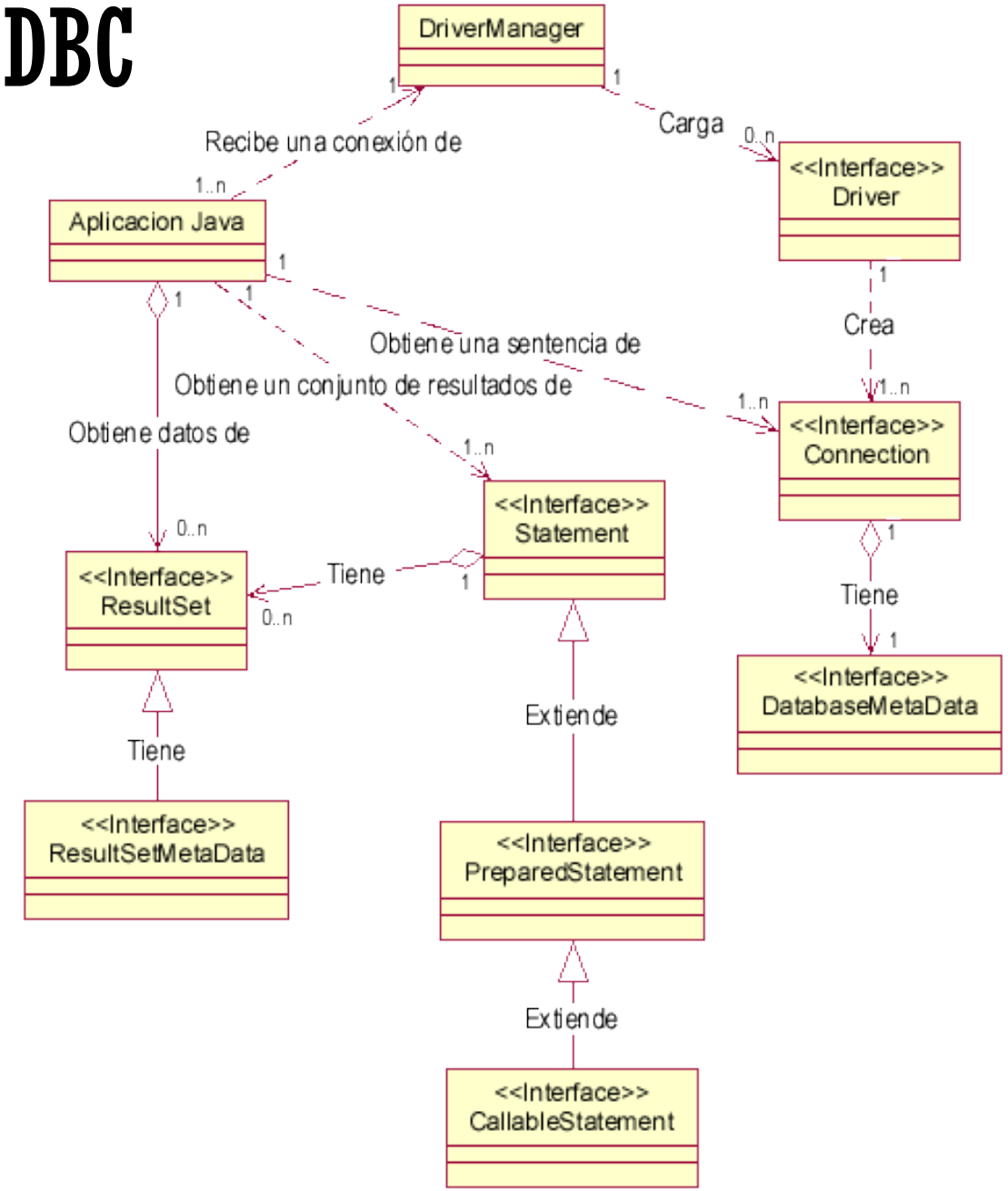
JDBC

- Es una herramienta muy potente para desarrollar aplicaciones que interactúan con la BD, en las que tenemos independencia del tipo de servidor de la BD que usemos.
- Para conseguir esta independencia del servidor, se ha construido una jerarquía de clases que hacen posibles las tareas necesarias para el trabajo con BD tales como: la conexión la BD, intercambio de información entre la aplicación y la BD, etc.

JDBC JAVA DATABASE CONNECTIVITY



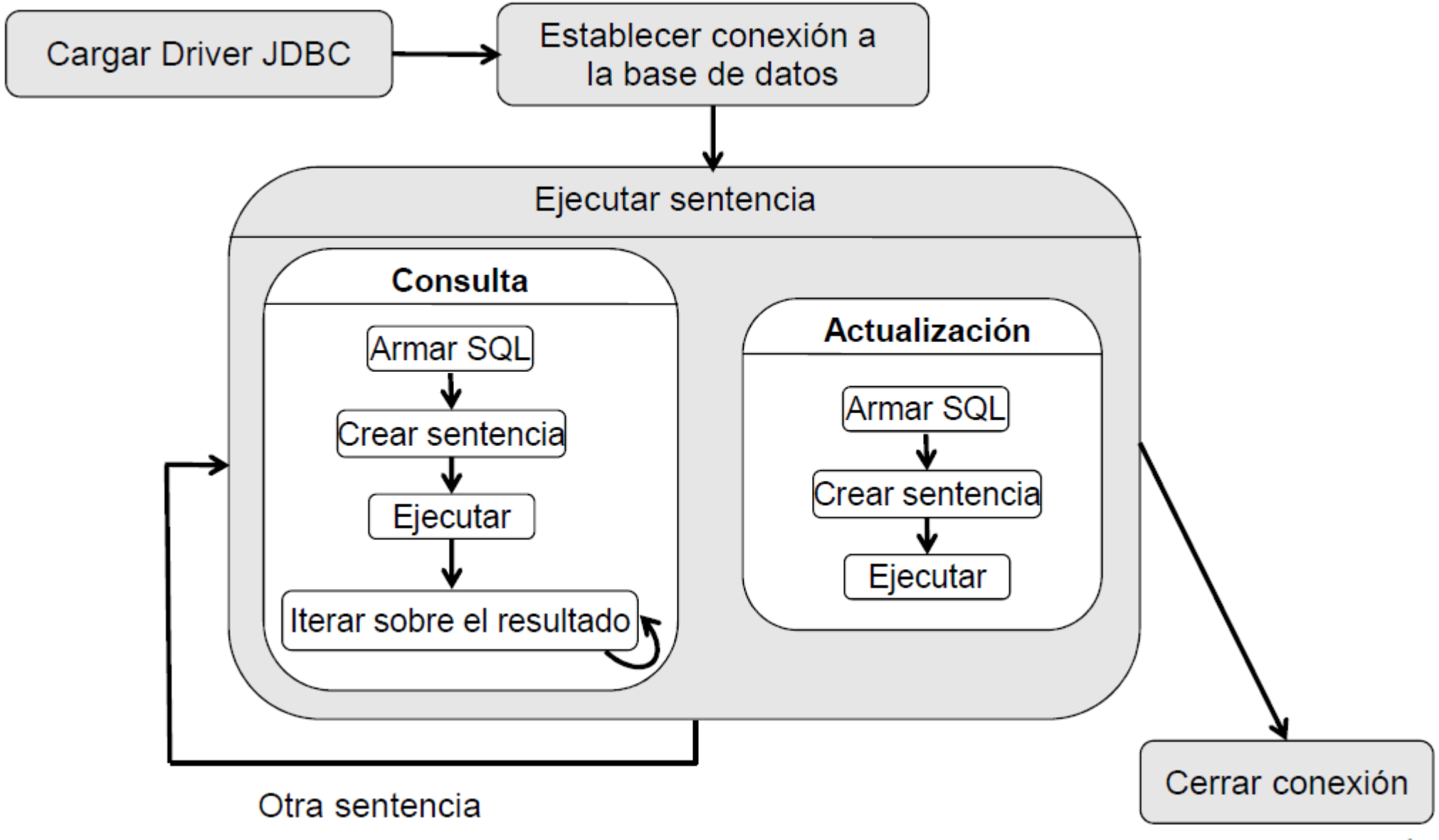
MODELO DE CLASES DEL JDBC



MODELO DE CLASES DEL JDBC

Clase / Interface	Descripción
Driver	Permite conectarse a una BD: cada gestor de BD requiere de un driver distinto
DriverManager	Permite gestionar todos los drivers instalados en el sistema
DriverPropertyInfo	Proporciona diversa información acerca de un driver
Connection	Representa una conexión con una BD. Una aplicación puede tener más de una conexión a más de una BD
DatabaseMetadata	Proporciona información acerca de una BD (como las tablas que la componen)
Statement	Permite ejecutar sentencias SQL sin parámetros
PreparedStatement	Permite ejecutar sentencias SQL con parámetros de entrada
CallableStatement	Permite ejecutar sentencias SQL con parámetros de entrada y salida, típicamente procedimientos almacenados
ResultSet	Contiene los registros obtenidos al ejecutar un SELECT
RsultSetMetadata	Permite obtener información sobre un ResultSet (nombre columnas, número…)

CICLO DE EJECUCIÓN



CLASE DRIVER MANAGER

DriverManager: clase estática (no requiere instanciación de objetos) que administra los drivers jdbc disponibles para iniciar conexiones.

- Connection getConnection(String url): intenta iniciar una conexión a una base de datos según los parámetros especificados en el url. Generalmente el string de conexión tiene el siguiente formato:

  jdbc:<driver>:<propiedades de la conexión>

  Para MySQL:

  jdbc:mysql://<servidor>:<puerto>/<base_datos>?<parametros>

  Por ejemplo: jdbc:mysql://localhost:3306/curso?&user=admin&password=Password1234
- **Connection getConnection(String url, String usuario, String clave):** idem al anterior pero por compatibilidad y seguridad, el usuario y la clave de acceso, son parámetros individuales.
- **setLoginTimeout(int segundos):** configura la cantidad de segundos de espera para intentar establecer la próxima conexión a una base de datos

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

CLASE CONNECTION

Connection: interfaz para implementar una sesión cliente-servidor con una base de datos.

- **Statement createStatement():** crea una nueva sentencia para ejecutar código SQL en forma directa en el servidor a traves de la conexión.
- **PreparedStatement prepareStatement(String sql):** crea una sentencia preparada con una estructura predeterminada dada por parámetros, para luego enviar los datos efectivos.
- **boolean isValid(int timeout):** verifica que la conexión está abierta y disponible para ejecutar una operación. Es necesaria para determinar si la conexión de red aun permanece activa desde la ejecución del último SQL.
- **close():** cierra la conexión y libera los recursos utilizados.
- **setAutoCommit(boolean autoCommit), commit(), rollback():** utilizados para el manejo de transacciones en la conexión actual.

https://docs.oracle.com/javase/8/docs/api/java/sql/Connection.html

CLASE STATEMENT

Statement: se utiliza para ejecutar una sentencia SQL en base a un string estático, ya sea un comando o una consulta.

- **boolean execute(String sql):** ejecuta cualquier tipo de SQL. Si es una consulta se debe recuperar el resultado mediante el método ResultSet getResultSet();
- **int executeUpdate(String sql):** sólo para comandos de actualización de datos (insert, delete, update) o configuración dinámica de la sesión (transacciones, concurrencia, etc).
- **ResultSet executeQuery(String sql):** sólo para consultas que retornan un resultado en filas o registros.
- **addBatch(String sql),** int[] executeBatch(), clearBatch(): permiten ejecutar una secuencia de comandos enviados en un lote.
- **void setQueryTimeout(int seconds):** impone un límite de espera por el resultado de la ejecución de la sentencia.
- **close():** cierra la sentencia liberando los recursos utilizados

https://docs.oracle.com/javase/8/docs/api/java/beans/Statement.html

CLASE RESULSET

ResultSet: contiene el conjunto resultado de una consulta SQL, estructurado en filas y columnas, con el comportamiento de un iterador.

- **boolean next():** avanza el índice interno del iterador a la proxima fila. Retorna false si no hay mas filas.
- String getString(int columnIndex), String getString(String columnLabel): permiten recuperar los valores de las columnas como un String, según su posición en la fila (la primer columna es 1) o mediante su nombre respectivamente.
- int getInt(...), long getLong(...), float getFloat(...), double getDouble(...), boolean getBoolean(...), Date getDate(...), Timestamp getTimestamp(...), Object getObject(...), etc...: una función para cada tipo de dato (java) de las columnas.
- **boolean wasNull():** verifica si el último valor recuperado de una columna correpondía al valor NULL de SQL.
- **boolean previous(), boolean first(), boolean last(), boolean absolute(int row), Boolean relative(int rows):** funciones para navegar en el conjunto resultado.
- **ResultSetMetaData getMetaData():** para recuperar los meta-datos (cantidad de columnas, tipos, ...) del conjunto resultado y de las columnas

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

CLASE RESULTSETMEDATA

ResultSetMetaData: permite obtener los tipos y propiedades de las columnas de un conjunto resultado (ResultSet):

- **int getColumnCount():** cantidad de columnas en el resultado.
- **String getColumnLabel(int column), String getColumnName(int column):** recuperar el nombre modificado o el nombre real de una columna respectivamente.
- **String getColumnClassName(int column):** recuperar la clase de Java determinada como predefinida para el tipo de dato de una columna.
- **int getColumnType(int column), String getColumnTypeName(int column):** recuperar el tipo de dato SQL estándar y SQL específico de una columna respectivamente.
- **int isNullable(int column), boolean isAutoIncrement(int column):** propiedades particulares de una columna.
- **boolean isSearchable(int column):** determina si es posible ejecutar un filtro en el WHERE de una consulta sobre la columna indicada.

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

PASOS DEL CICLO EJECUCIÓN

PASO 1: CONEXIÓN A LA BD

PASO 2: DIÁLOGO CON LA BD

PASO 3: CIERRE CONEXÓN

CONEXIÓN A LA BASE DE DATOS

- A la hora de conectarnos a una BD, debemos tener en cuenta que cada una tiene su propio lenguaje, es decir, necesitamos un traductor que convierta órdenes que le transmitimos en comandos entendibles a esa BD concreta.
- A este tipo de conectaros se les llama controladores.
- Existen gran cantidad de controladores JDBC que nos permiten conectarnos a bases de datos de distintos fabricantes
- Estos controladores se distribuyen como clases de Java. Por lo tanto, lo único que tendremos que hacer es cargarlas en nuestra aplicación con una sentencia similar a:

```
Class.forName(“controlador”);
```

EJEMPLOS DE DRIVERS

Loading class `com.mysql.jdbc.Driver'. This is deprecated. The new driver class is `com.mysql.cj.jdbc.Driver'. The driver is automatically registered via the SPI and manual loading of the driver class is generally unnecessary.

- Ejemplo de Driver para Oracle:

```
String driver = new String (“oracle.jdbc.driver.OracleDriver”);
Class.forName(driver);
```

- Ejemplo de Driver para MySQL:

Cambiar por → `Class.forName("com.mysql.cj.jdbc.Driver");`

```
String driver = new String (“com.mysql.jdbc.Driver”);
Class.forName(driver);
```

- Ejemplo de Driver para SQL Server:

```
Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
```

- Ejemplo de Driver para PostgresSQL:

```
Class.forName("org.postgresql.Driver");
```

CONEXIÓN A LA BASE DE DATOS

- Una vez que hemos indicado y cargado el controlador es necesario abrir la conexión con la BD, indicando a que ordenador van dirigidas las ordenemos que lancemos.
- Para realizar esto es necesario crear un objeto tipo **`Connection`** utilizando le método **`getConnection()`** de la clase **`DriverManager`** y que tiene como argumento un URL JDBC.
- La sintaxis de la URL JDBC tiene la siguiente forma:

```
jdbc::subprotocolo://servidor:puerto/basedatos
```

- Ejemplo:

```
Connection conexión =
DriverManager.getConnection(“jdbc:mysql://172.29.2.70:1521/nominas”);
```

DIÁLOGO CON LA BD

- Una vez que se tiene abierta la conexión con la BD mediante el objeto de tipo **`Connection`**, se pueden hacer solicitudes de información mediante un objeto de la clase **`Statement`**
- Para crear las distintas órdenes utilizaremos el método **`createStatement`**() de la clase **`Connection`**. Este método crea un objeto Statement asociado a la conexión con la BD que estamos utilizando.

```
Statement sentencia = conexion.createStatement();
```

CIERRE DE LA CONEXIÓN

- Cuando terminamos de operar con la Base de Datos se debe cerrar la conexión establecida.
- Para ello, se utilziará el método `close()` que se encuentra definido en `Connection, ResultSet y Statement.`

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

EJEMPLO CONEXIÓN

```
import java.sql.*;
String servidor = "localhost:3306";
String baseDatos = “curso";
String usuario = “admin";
String clave = “Password1234";
String url = "jdbc:mysql://" + servidor + "/" +baseDatos;
Connection cnx;
try
{
        cnx =DriverManager.getConnection(url, usuario, clave);
}catch (SQLException ex){
   System.out.println("SQLException: " + ex.getMessage());
   System.out.println("SQLState: " + ex.getSQLState());
   System.out.println("VendorError: " + ex.getErrorCode());
}
```

DIÁLOGO CON LA BD

- Con el objeto sentencia que nos devuelve el método **`createStatement`** podemos realizar cualquier tipo de operación (consultas, modificaciones, etc) sobre la BD.
- Para realizarlas se suele utilizar dos métodos de la clase **`Statement`**:
  - **`execute (String sentencia)`** para ejecutar una SQL que no devuelve datos
  - y **`executeQuery(String sentencia)`** para ejecutar una consulta SQL que sí devuelve datos
- Cada una de estas sentencias puede devolver cero o más resultados, JDBC utiliza los objetos tipo **`ResultSet`** para capturar la información que recibe

```
ResultSet resultado = sentencia.executeQuery ("SELECT * FROM TABLA");
```

- De esta forma tendremos el objeto **`resultado`** con el contenido de la consulta, el cual podremos recorrer con los métodos definidos en la clase **`ResultSet`**. Por ejemplo, con su método **`next()`**
- El objeto ResultSet devuelto por el método **`executeQuery()`**, permite recorrer las filas pero no proporciona información sobre la estructura. Para ello se utiliza **`ResultSetMetaData`**. Para obtener un objeto **`ResultSetMetaData`** basta con llamar al método **`getMetadaData()`** del objeto **`ResultSet`**

EJEMPLO CONSULTAS

```
import java.sql.*;
try{
    Statement stmt = cnx.createStatement(); // Se crea una sentencia jdbc para realizar la consulta
    String sql = "SELECT depno, dnombre, localidad FROM deparamentos”; // Se prepara el string SQL de la consulta
    ResultSet rs = stmt.executeQuery(sql); // Se ejecuta la sentencia y se recibe un resultado
    while (rs.next()){// Se recorre el resultado
        String nombre = rs.getString("dnombre");
        int id = rs.getInt(“deono);
        String loc = rs.getString(“localidad");
    }
    rs.close();
    stmt.close();

}

catch (SQLException ex) {

            System.out.println("SQLException: " + ex.getMessage());

}
```

EJEMPLO CONSULTAS ACTUALIZACIÓN

```
try{ // Se crea una sentencia para realizar la consulta
        Statement stmt = cnx.createStatement();// Se prepara el string SQL de la inserción
        String sql = "INSERT INTO DEPARTAMENTOS (dep_no, dnombre, localizacion) VALUES (100, ‘SOPORTE TÉCNICO’, ‘Madrid’)";
        stmt.execute(sql); // Se ejecuta la inserción
        stmt.close();
}catch (java.sql.SQLException ex){
   System.out.println("Mensaje: " + ex.getMessage()); // Mensaje retornado por MySQL
   System.out.println("Código: " + ex.getErrorCode()); // Código de error de MySQL
}
```

UT10. BASES DE DATOS

Módulo: PROGRAMACIÓN

Curso 2022/2023. 1° DAM

Ruth Lospitao Ruiz