CONEXIÓN A MYSQL DEDE JAVA

1. Descarga del conector JDBC (Java Database Connectivity)

mysql-connector-java-8.0.29.jar

Paquete java.sql [editar]

JDBC ofrece el paquete `java.sql`, en el que existen clases muy útiles para trabajar con bases de datos.

Clase	Descripción
DriverManager	Para cargar un driver
Connection	Para establecer conexiones con las bases de datos
Statement	Para ejecutar sentencias SQL y enviarlas a las BBDD
PreparedStatement	La ruta de ejecución está predeterminada en el servidor de base de datos que le permite ser ejecutado varias veces
CallableStatement	Para ejecutar sentencias SQL de Procedimientos Almacenados.
ResultSet	Para almacenar el resultado de la consulta

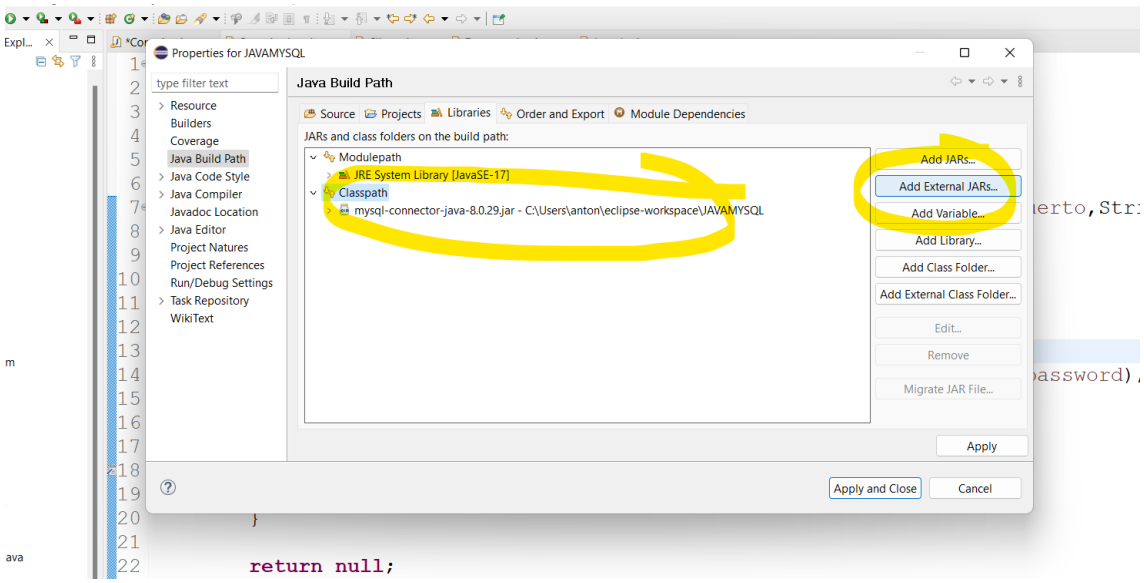
Tabla 1 Paquete Java - Fuente: Wikipedia

Tanto **java.sql.Connection** como **java.sql.Statement** son interfaces del API de Java. Es decir, el lenguaje los incorpora como punto de extensibilidad en el sistema a la hora de conectarnos a una base datos

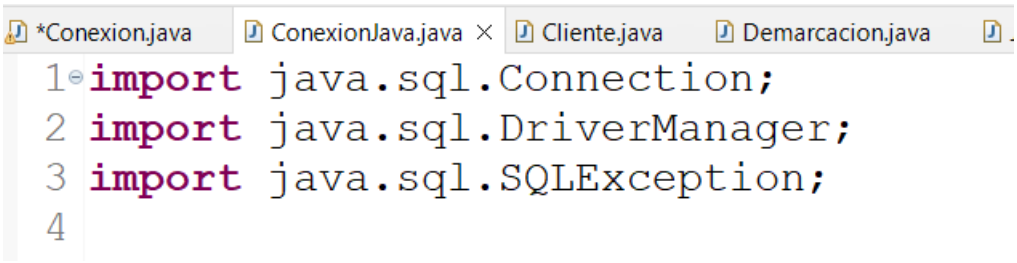
Cada sistema de base de datos debe aportar sus propias implementaciones y es ahí donde el ***Driver JDBC*** realiza sus aportes. El concepto de Driver hace referencia al conjunto de clases necesarias que implementa de forma nativa el protocolo de comunicación con la base de datos en un caso será Oracle y en otro caso será MySQL.

2. Importar el conector al proyecto

(Propiedades del proyecto - Java Build Path - Pestaña ***Libraries Add external Jars***



3. **Importar las clases**



4. **Registro del driver. Es necesario registrar el driver, esto inicializa la clase estáticamente.**

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

Qué hace exactamente esta línea: https://foroayuda.es/que-hace-exactamente-esto-class-forname-com-mysql-jdbc-driver-newinstance/

El registro del driver ha de ir dentro de un bloque try-catch, será el propio IDE quien nos recordará añadir la captura de la excepción.

```
String url="jdbc:mysql://"+direccion+":"+puerto+'/'+bd;

try {
    Class.forName("com.mysql.cj.jdbc.Driver");
    Connection conexion=DriverManager.getConnection(url, user, password);
    return conexion;

} catch (ClassNotFoundException | SQLException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

5. **Crear la URL de conexión del JDBC. Para establecer la conexión es necesario crear la URL de conexión con la siguiente sintaxis:**

jdbc:mysql://hostname:port/databasename

- ***hostname***: El nombre de host donde está instalado el servidor MySQL. Si la BD está en local podemos utilizar la palabra ***localhost*** o la dirección IP local ***127.0.0.1.***
- ***port:*** *El puerto TCP / IP donde escucha el servidor MySQL, por defecto 3306.*
- ***databasename: (opcional)*** *El nombre de la base de datos a que queremos conectarnos.*

6. Crear objeto Connection

Una vez cargado el driver es necesario crear un objeto del tipo **Connection**, para administrar la conexión. A partir de la clase ***DriverManager*** se obtendrá una objeto de tipo conexión, Connection, con una los datos indicados en la función ***getConnection***.

7. Establecimiento de la conexión

La siguiente línea de código muestra cómo se realiza la conexión empleando la URL del punto 5:

Connection con = DriverManager.getConnection(url, "myLogin", "myPassword");

```
String url="jdbc:mysql://"+direccion+":"+puerto+'/'+bd;

try {
    Class.forName("com.mysql.cj.jdbc.Driver");
    Connection conexion=DriverManager.getConnection(url, user, password);
    return conexion;

} catch (ClassNotFoundException | SQLException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

return null;
```

Si uno de los drivers que hemos cargado reconoce la URL suministada por el método ***DriverManager.getConnection***, el driver establecerá una conexión con el controlador de base de datos especificado en la URL del JDBC.

La clase DriverManager , como su nombre indica, maneja todos los detalles del establecimiento de la conexión.

La conexión devuelta por el método DriverManager.getConnection es **una conexión abierta que se puede utilizar para crear sentencias JDBC que pasen nuestras sentencias SQL al controlador de la base de datos.**

8. Ejecución de consultas contra la BD a partir de un *Statement*

Recurriremos a un objeto Statement que será con el que ejecutemos la consulta mediante su método ***execute()*** o ***executeQuery().*** El objecto Statement se crea a partir de la conexión obtenida en el punto anteri,or de la siguiente forma

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

En caso de que necesitemos que el cursor que devuelve la consulta, permita moverse en ambos sentidos sobre el conjunto de resultados, debemos de indicárselo al método

createStatement() como parámetro. Recordar que es necesario controlar y capturar la excepción **SQLException** al invocar al método ***createStatement***()

```
try {
    s = conexion.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, 0);
} catch (SQLException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

Más información de la clase Statement en http://www.euskalnet.net/jaoprogramador

Para ejecutar la sentencia SQL que será enviada a la base de datos es pasada como un argumento a uno de los métodos de ejecución del objeto Statement.

```
ResultSet rs = s.executeQuery ("SELECT customerID,companyName FROM customers");
```

La interfaz Statement nos ofrece tres métodos diferentes para ejecutar sentencias SQL, ***executeQuery***, ***executeUpdate*** y ***execute***. El método a usar está determinado por el tipo de la sentencia SQL, nosotros emplearemos el método ***executeQuery*** cuando las consultas lanzadas devuelvan resultados.

9. Uso del método *preparedStatement* para crear sentencias que acepten parámetros. Fuente: Chuidiang , acervolima.com y arquitecturajava

Una Prepared Statement es una sentencia SQL de base de datos precompilada que acepta parámetros. Al estar precompilada, su ejecución será más rápida que una SQL normal, por lo que es adecuada cuando vamos a ejecutar la misma sentencia SQL (con distintos valores) muchas veces. Por ejemplo con un filtro o en una inserción de registros.

Desde java tenemos posibilidad de crear y usar estas PreparedStatement. Tienen dos ventajas sobre el createStatement() que hemos mencionado en el punto anterior.

Son más rápidas en ejecución y la segunda ventaja es que no debemos preocuparnos de revisar los datos que introduzca el usuario, como en el caso de las Statement. Tampoco nos tenemos que preocupar de formatear adecuadamente los Date u otros tipos de datos. A una PreparedStatement le pasamos directamente las clases java que representan nuestros datos (String, Number, Integer, Date, etc) y ella solita se encarga de ponerlo todo correctamente.

```
try {
    String consulta="SELECT companyName FROM customers WHERE CustomerID = ?";
    PreparedStatement psConsulta=conexion.prepareStatement(consulta);
    psConsulta.setString(1,"ALFKI");
    rs=psConsulta.executeQuery();
    //Statement s1 = conexion.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, 0);
} catch (SQLException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

Con las preparedStatement, en lugar de codificar consultas como

select * from students where age>10 and name ='Chhavi'

Se establecen marcadores de posición de parámetros (se usa un signo de interrogación para los marcadores de posición) como,

select * from students where age> ? and name = ?

PreparedStatement myStmt;

myStmt = myCon.prepareStatement(select * from students where age> ? and name = ?);

Para configurar los parámetros que sustituirán a los comodines representados por asteriscos se utilizan los métodos set???, por ejemplo.

myStmt.setInt(1,10);

myStmt.setString(2,"Chhavi");

Aquí se muestran los métodos que se pueden emplear con las consultas preparadas, en los métodos set el primer parámetro indica el orden de los parámetros a reemplazar por valores concretos.

setInt (int, int): este método se puede utilizar para establecer un valor entero en el índice de parámetro dado.

setString (int, string): este método se puede utilizar para establecer el valor de la string en el índice de parámetro dado.

setFloat (int, float): este método se puede utilizar para establecer el valor flotante en el índice de parámetro dado.

setDouble (int, double): este método se puede utilizar para establecer un valor doble en el índice de parámetro dado.

executeUpdate(): este método se puede utilizar para crear, soltar, insertar, actualizar, eliminar, etc. Devuelve el tipo int.

executeQuery(): devuelve una instancia de ResultSet cuando se ejecuta una consulta de selección.

10. Manejo de los datos obtenidos a partir de un ResultSet (Fuente: ArquitecturaJava y programacion.net)

Trabajando con un programa que acceda a una base de datos con Java JDBC es necesario controlar los posibles fallos a través de un SQLException. SQLException es la excepción que se lanza cuando hay algún problema entre la base de datos y el programa Java JDBC.

Por ejemplo, cuando realizamos una consulta debemos de poner nuestro código que maneje el ResultSet dentro de de un bucle try-catch.

```
ResultSet rs;
try {
    rs = s.executeQuery ("SELECT customerID,companyName FROM customers");
} catch (SQLException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

En este ejemplo la variable ***e*** perteneciente a SQLException dispone de los siguientes métodos que nos permitiría conocer información acerca del error que lanza la consulta

- ***.getMessage*()**, nos indica la descripción del mensaje de error.
- ***.getSQLState*()**, devuelve un código SQL estándar definido por ISO/ANSI y el Open Group que identifica de forma unívoca el error que se ha producido.
- ***.getErrorCode*()**, es un código de error que lanza la base de datos. En este caso el código de error es diferente dependiendo del proveedor de base de datos que estemos utilizando.
- ***.getCause*()**, nos devuelve una lista de objetos que han provocado el error.
- ***.getNextException*()**, devuelve la cadena de excepciones que se ha producido. De tal manera que podemos navegar sobre ella para ver en detalle de esas excepciones.

Existen diferentes **ResultSet Types**

```
1  try (Connection conexion = getConnection();
2  Statement sentencia = conexion.createStatement();
3  ResultSet resultSet = sentencia.executeQuery(sql)) {
4  while (resultSet.next()) {
5  //gestionar resultset
6  }
7  } catch (SQLException e) {
8
9  }
```

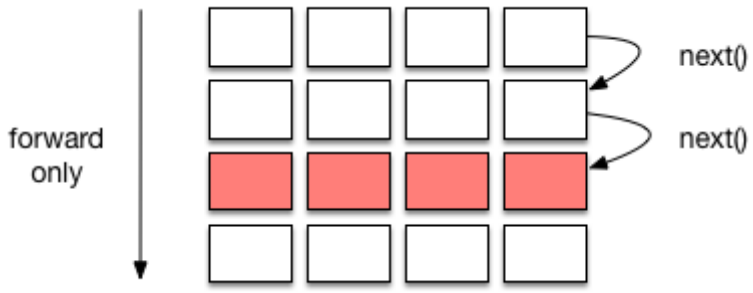
. Lo más importante en este código a nivel de ResultSets es la creación de un Statement :

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

Este código es idéntico al siguiente en cuanto a funcionalidad:

```
1  Statement sentencia = conexion.createStatement(ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_READ_ONLY);
```

En JDBC por defecto un ResultSet es **TYPE_FORWARD_ONLY** y **CONCUR_READ_ONLY**. ¿Qué quiere decir esto? . Quiere decir que el ResultSet solo se puede mover hacia delante y solo puede leer los datos. No tiene ninguna capacidad para actualizarlos. Como se vio anteriormente, con **TYPE_SCROLL_SENSITIVE** podemos recorrer el resultSet en ambos sentidos.



Utilizar el Método next

La variable **rs**, que es un ejemplar de **ResultSet**, contiene el resultado de la consulta lanzada. Necesitamos acceder a las columnas de cada fila y recuperar los valores de acuerdo con sus tipos.

Emplearemos el método **next()** mueve algo llamado cursor a la siguiente fila y hace que esa fila (llamada fila actual) sea con la que podamos operar. Como el cursor inicialmente se posiciona justo encima de la primera fila de un objeto ResultSet, primero debemos llamar al método next para mover el cursor a la primera fila y convertirla en la fila actual.

Sucesivas llamadas del método next moverán el cursor de línea en línea de arriba a abajo. (Es habitual ver el método next dentro de un bucle)

Observa que con el JDBC 2.0, cubierto en la siguiente sección, se puede mover el cursor hacia

Utilizar los métodos getXXX

Los métodos getXXX del tipo apropiado se utilizan para recuperar el valor de cada columna. Por ejemplo, la primera columna de cada fila de un rs podría ser ***customerID*** que almacena un valor del tipo ***VARCHAR*** de SQL.

El método para recuperar un valor VARCHAR es getString. La segunda columna de cada fila podría almacenar, como ejemplo, un valor del tipo FLOAT de SQL, y el método para recuperar valores de ese tipo es getFloat. Puedes consultar los distintos tipos de métodos get en el siguiente enlace. https://docs.oracle.com

JDBC ofrece dos formas para identificar la columna de la que un método getXXX obtiene un valor. Una forma es dar el nombre de la columna, o se puede indicar el índice de la columna (el número de columna), con un 1 significando la primera columna, un 2 para la segunda, etc.

Podéis ver un ejemplo de cómo obtener como String el valor de la columna 1 de los registros que devuelve la consulta.

```
rs=s.executeQuery ("SELECT customerID FROM customers;");

while (rs.next())
{
    System.out.println (rs.getString(1));
}
```

11. Uso del ResultSetMetaData (fuente: Consulta JDBC sin conocer campos)

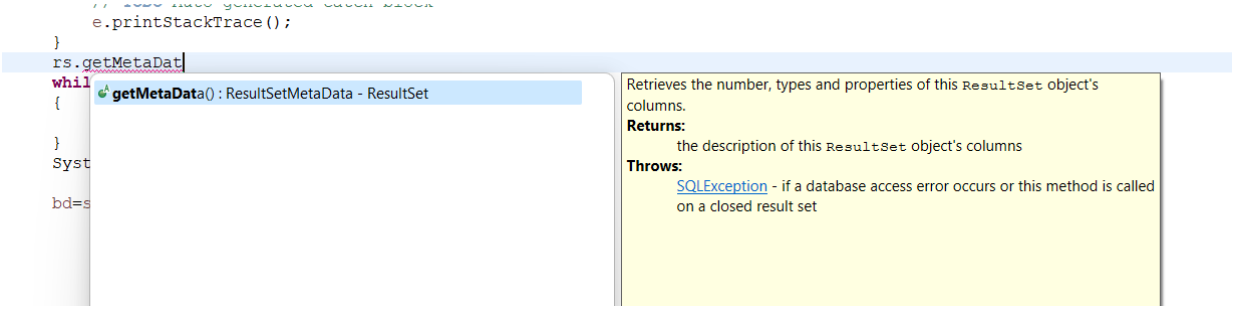
Como ejemplo tomemos la consulta ***SELECT * FROM customers*** de la BD de muestra Northwind.

Desconocemos la cantidad de campos y filas que puede tener una tabla de este calibre, imaginemos dentro de una gran empresa.

Nos sirve de ejemplo para mostrar todas las filas y columnas que ésta contiene.

La ejecución de la sentencia se hace con el método ***.executeQuery()*** y los resultados se almacenan en el ***ResultSet*** como se hizo en los puntos anterior.

Para conocer el número columnas de nuestra consulta necesitamos conocer los metadatos de la consulta. Los metadatos es información relacionada con el conjunto de resultados obtenidos. Para acceder a la información de los metadatos tenemos el método ***.getMetaData()*** dentro del ResultSet



Este método devolverá otro ResultSet, pero con meta-información, un RESULTSETMETADATA.

```
}
ResultSetMetaData rsmd=rs.getMetaData();
```

De esta meta-información tenemos varios métodos especialmente útiles:

- **getColumnCount()**, que nos dice el número de columnas que hay en la consulta.
- **getColumnName(id)**, que dado un número de columna nos devuelve su nombre.
- **getColumnLabel(id),** que dado un número de columna devuelve su alias o etiqueta.

Lo que vamos a hacer es recorrer mediante un bucle todas las columnas y mostrar sus nombres.

TIPOS DE DATOS

El controlador JDBC convierte el tipo de datos Java al tipo JDBC adecuado antes de enviarlo a la base de datos.
Emplea una asignación predeterminada para la mayoría de los tipos de datos. Por ejemplo, un int de Java se convierte en un INTEGER de SQL.
El mapeo de los tipos de datos Java a SQL es muy sencillo, tal como se muestra en la tabla

SQL	Java
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	java.Math.BigDecimal
DECIMAL	java.Math.BigDecimal
BIT	Boolean
TINYINT	Byte
SMALLINT	Short
INTEGER	Int
BIGINT	long
REAL	float
DOUBLE	double
FLOAT	double
BINARY	byte []
VARBINARY	byte []
LONGVARBINARY	byte []
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp