# Tema 1: Manejo de conectores

## 1.1-Protocolos de acceso a Bases de datos

**Podemos encontrarnos con dos normas de conexión a BD SQL:**

**-ODBC:(Open Database Connectivity) define una API que pueden usar las aplicaciones para abrir una conexión con una BD ,enviar consultas,actualizaciones…..fué desarrollado por Microsoft con el objeto de acceder a cualquier dato desde cq aplicación.**

**-JDBC:(Java Database Connectivity) define una API que pueden usar los programas ó aplicaciones Java para conectarse a los servidores de BD relacionales , no solo provee de una interfaz sino que define una arquitectura estándar, para que los fabricantes puedan crear los drivers que permitan a las aplicaciones java el acceso a los datos.JDBC dispone de una interfaz distinta para cada BD es lo que llamamos driver(controlador ó conector).JDBC consta de un conjunto de clases e interfaces que nos permiten escribir aplicaciones Java para gestionar tareas con una BD relacional:**

**-Conectarse a la BD.**

**-Enviar consultas y actualizaciones a la BD.**

**-Recuperar y procesar los resultados recibidos de la BD en respuesta a las consultas.**

## 1.2-Arquitecturas JDBC:

La API JDBC es compatible con los modelos:

**1.2.1-Modelo de dos capas**:es decir un applet ó aplicación java "hablan" directamente con la BD, lo cual requiere un driver JDBC residiendo en el mismo lugar que la aplicación.Desde el programa java se envían sentencias SQL al SGBD para que la procese y los resultados se envían de vuelta al programa,la BD puede encontrarse en otra máquina diferente a la de la aplicación y las solicitudes se hacen a través de la red, el driver será el encargado de manejar las comunicaciones a través de la red de forma trasparente al programa.

**1.2.2-Modelo de tres capas**,los comandos se envían a una capa intermedia,que se encargará de envíar los comandos SQL a la BD y recoger los resultados, en este caso tenemos un applet en una máquina ejecutándose y accediendo a un driver de Bd situado en otra máquina ,en este caso los drivers no tienen que residir en la máquina cliente.

**Conector:**

Se define como un conjunto de clases encargadas de implementar la interfaz de programación de aplicaciones(API) y facilitar con ello el acceso a una base de datos para poder conectarse a ella y lanzar consultas,esto sería inviable sin el uso de un conector.

## 1.3-El desfase Objeto-Relacional.

Los sistemas gestores de bases de datos poseen lenguajes propios que permiten gestionar los datos,pero cuando se quiere acceder a los datos desde lenguajes de programación, con independencia del sistema gestor, es necesario usar conectores que faciliten esas operaciones.

A medida que evolucionaron los LP y surgió la programación orientada a objetos, se produjo un problema: **el desfase objeto-relacional**,que se genera cuando en las aplicaciones se gestionan objetos, pero se tienen que almacenar filas y tablas ya que las BD utilizan el modelo (E/R) entidad-relación,lo cual implica que el desarrollador tenga que diseñar dos diagramas diferentes para el diseño de la aplicación.

## 1.4-Protocolos de acceso a Bases de Datos:

Muchos servidores de BD utilizan protocolos de comunicación específicos que facilitan el acceso a los mismos, lo que obliga a aprender un lenguaje nuevo para trabajar con cada uno de ellos.Es posible reducir esa diversidad de protocolos mediante el uso de alguna interfaz de alto nivel que ofrezca al programador una serie de métodos para acceder a la base de datos.

Estas interfaces ofrecen facilidades para:

-Establecer una conexión a una BD.DriverManager

-Ejecutar consultas sobre la BD.

-Procesar los resultados de las consultas.

**Las tecnologías disponibles,proporcionan una interfaz común basada en el lenguaje de consulta(SQL) para el acceso homogéneo a los datos .Podemos encontrarnos con dos normas de conexión a una base de datos SQL . ODBC de Microsoft ,JDBC de Sun, en este curso nos centraremos en JDBC, porque además de ser la más utilizada, usa java como punto de partida.**

Al conjunto de clases encargadas de implementar la interfaz de programación de aplicaciones(API) y facilitar con ello el acceso a una BD se denomina conector ó driver.Para poder conectarse a una BD y lanzar consultas, una aplicación siempre necesita tener un conector asociado.

Un ejemplo de conector muy extendido es JDBC ,es una capa de software intermedia situada entre los programas java y los sistemas de gestión de bases de datos relacionales que utilizan SQL.Esta capa es independiente de la plataforma y del gestor de BD utilizado.

Con este conector no hay que escribir un programa diferente para acceder a una BD Access y otro para acceder a una BD Oracle,sino que se puede escribir un único programa utilizando el API JDBC y es ese programa

el que se encarga de enviar las consultas a la BD utilizada en cada caso.

Otro ejemplo de conector es el de Microsoft ODBC la diferencia con JDBC es que tiene una interfaz desarrollada en C,la mayoría de sistemas gestores de bases de datos disponen de drivers para trabajar con ODBC y JDBC.Además tb existe en Java un driver JDBC-ODBC,para convertir llamadas JDBC a ODBC y poder acceder a BD que ya tienen un driver ODBC y todavía no tienen un conector JDBC.

## 1.4 Acceso a Bases de Datos Relacionales Mediante Un Conector JDBC.

**1.4.1_PRIMER PASO**:

Registrar un Driver JDBC. Esto consiste en **CARGAR LA CLASE QUE CORRESPONDA A DICHO DRIVER** en la JVM usando la **clase Class.**

**JDBC**:es una **interfaz desarrollada por Oracle** que permite la conexión de un programa escrito en Java y el SGBD que opera sobre la BD.

**ESTE EJEMPLO ES PARA UNA BD ACCESS**

```
try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
}
```

También se puede poner creando un objeto String:

```
String driverJDBC="sun.jdbc.odbc.JdbcOdbcDriver"
try
```

{

Class.forName(driverJDBC);

}

**EJEMPLO PARA MySQL**

```
String driver= "com.mysql.jdbc.Driver";
 try
     {
      Class.forName(driver);
     }
    catch(ClassNotFoundException ex)
    {
     System.out.println("No se encontró el Driver"+driver);
     System.exit(1);
    }
```

**En la versión 4.0 de JDBC ,el driver ya se carga con la clase DriverManager y no haría falta la clase Class con el método forName(…)**

**1.4.2_SEGUNDO PASO:**

Establecer la conexión: Par esto hay que usar el método **getConnection()** es un método estàtico de la clase **DriverManage**r.**Ej DriverManager.getConnection(url);**Este método retorna un objeto perteneciente a la **clase Connection.**

Este método requiere un parámetro fundamental que es la **URL** con el fín de saber con que base de datos se va a trabajar.

**FORMATO URL:protocolo:subprotocolo:identificacion de la base de datos**:

1-Protocolo: en una URL JDBC es siempre **jdbc**.

2-Subprotocolo:es el mecanismo de conexión que se va a usar ó la indicación del driver JDBC que se va a usar.**Ej odbc,db2etc**…

3-Identificación de la BD.Es necesario:

-El driver que se va a usar.

-El nombre de la BD.

## EJEMPLO para USBwebserver:

```
String url= "jdbc:mysql://localhost:3307/Alumnos?user=root&password=usbw";
  try
    {
    Connection  conexion=DriverManager.getConnection(url);
    }
     catch(SQLException e)
        {
            System.out.println("No hay ningún Driver registrado que reconozca la URL especificada");
            System.exit(2);
        }
     catch(Exception e)
     {
     System.out.println("\n\t Se ha producido algún otro error.");
     System.exit(3);
     }
```

## 1.5_PARA CREAR UN DRIVER Y CONECTARSE A UNA BD MySQL SERÍA:

a) **Utilizando Xampp**

```
String driverJDBC="com.mysql.jdbc.Driver";
String url="jdbc:mysql://localhost:3306/test?user=root&password=root";
```

b) **Utilizando USBwebserber**

String driver= "com.mysql.jdbc.Driver";

String url= "jdbc:mysql://localhost:3307/Alumnos?user=root&password=usbw";

**Si la BD no está creada ,entonces todavía no tiene un nombre por lo que habría que sustituir en la instrucción anterior después de escribir el Puerto una ? es decir 3307/? User=root& password =usbw**

```
 try
{
 Class.forName(driverJDBC);
}
catch(ClassNotFoundException ex)
{
 System.out.println("No se encontró el Driver"+driverJDBC);
 System.exit(1);
}
   try
{
 conexion=DriverManager.getConnection(url);
}
 catch(SQLException e)
    {
        System.out.println("No hay ningún Driver registrado que reconozca la URL especificada");
        System.exit(2);
    }
 catch(Exception e)
{
System.out.println("\n\t Se ha producido algún otro error.");
System.exit(3);
}
```

```
try
 {
  sentencia=conexion.createStatement();
 }
catch(Exception e)
{
System.out.println("Error al conectarse con el driver que maneja la BD"+e);
System.exit(4);
 }
```

Estos métodos lanzan el error **SQLException ,** si el cursor no apunta a ninguna fila.

Un objeto **ResultSet,es similar a una lista en la que está el resultado de la consulta ,tiene internamente un puntero que apunta al primer registro de la lista,mediante el método next() el puntero avanza al siguiente registro,para recorrer la lista de de registros usaremos dicho método dentro de un bucle while que se ejecutará mientras next() devuelva true,es decir mientras haya registros** será diferente en función de los **distintos parámetros** que se utilicen del mismo, pero siempre estará formada por un conj. De filas.

**Importante recordar que si cualquier operación que se vaya a realizar sobre la BD no es posible,lanzará el error:SQLException.**

# 1.6_Ejecución de sentencias de definición de datos.

## 1.6.1_Creación de tablas:

**Sentencia** es un objeto de la clase **Statement** que tiene un método que es **execute** ó **executeUpdate**,cualquiera de los dos métodos sirven **para crear la tabla** co la siguiente sintáxis:

**Sentencia.execute("CREATE TABLE empleados(nombre VARCHAR(70) NOT NULL,edad INTEGER,sueldo FLOAT);");**

**El método execute () devuelve un booleano true ó false.**

# 1.7_Ejecución de sentencias de manipulación de datos.

## 1.7.1_Inserción de registros a través de la clase Statement:

**Sentencia.executeUpdate("INSERT INTO acreedores VALUES(21,'David','Camino de Serrano2')");**

## 1.7.2_Consultas de la información anterior a través del objeto ResultSet:

El método executeQuery() devueve un objeto de la clase ResultSet.

```
ResultSet rs=Sentencia.executeQuery("Select *from acreedores");

While(rs.next())

{

Con++;

System. ….("\n\tNúmero del acreedor "+rs.getInt("numacred"));

……

}

rs.close();

}
```

# 1.8_Ejecución de consultas para acceso a los datos.

## 1.8.1_ RECUPERACIÓN DE INFORMACIÓN.

Una vez que estamos conectados a la BD gracias al objeto **Connection**(recordar se trata de una interfaz) podremos acceder a la Bd a través de instrucciones SQL.Esto se consigue gracias a que el objeto Connection tiene un **método createStatement()**,que permite la conexión con el driver que gestiona la BD este método retorna un objeto perteneciente a la clase **Statement Y dicha clase tiene un método** a través del cual se podrán ejecutar las

sentencias SQL llamado **executeQuery (),**la información obtenida de la BD con el método anterior **se** gestionará a través de un objeto de tipo **ResultSet.**

Ej:

**Connection** con =**DriverManager.getConnection(URL):**

**Statement** smt =con.**createStatement();**

**ResultSet** resul =smt.**executeQuery**("SELECT nombre,edad FROM Alumnos")

La **clase ResultSet** tiene **métodos get** con el fín de obtener los valores de los distintos campos que forman la fila donde está posicionado el cursor.

## 1.8.2_Métodos:

**boolean next():**desplaza el cursor a la siguiente fila.retorna true si hay una fila o false en caso contrario.

**boolean first():**El cursor pasa a apuntar a la primera fila,retorna true si hay al menos una fila.

**Void beforeFirst**(): Posiciona el cursor delante de la primera fila.

**Boolean previous();** Desplaza el cursor a la fila anterior.

**Boolean last():**El cursor pasa a apuntar a la última fila.

## Establecimiento de conexiones:

Conexión através de jdbc a bd embebidas:

1-SQLite: para conectarnos necesitamos la librería:sqlite-jdbc-3.7.2.jar.Buscar en web

www.xerial.org/trac/Xerial/wiki/SQLiteJDBC

```
Class.forName("org.sqlite.JBBC");
```

Connection
conexión=DriverManager.getConnection("jdbc:sqlite:D:/db/SQLite/nombre.db");

En la misma carpeta donde está laBD creamos nuestro programa.

2- Apache Derby: necesitamos la librería derby.jar .Copiamos el mismo programa y

Cambiamos la carga del driver

Descargar de http://db.apache.org/derby/derby downloads.html

Class.forName("org.apache.derby.jdbc.EmbeddedDriver");

Connection
conexión=DriverManager.getConnection("jdbc:derby:D:/db/Derby/nombre.db");