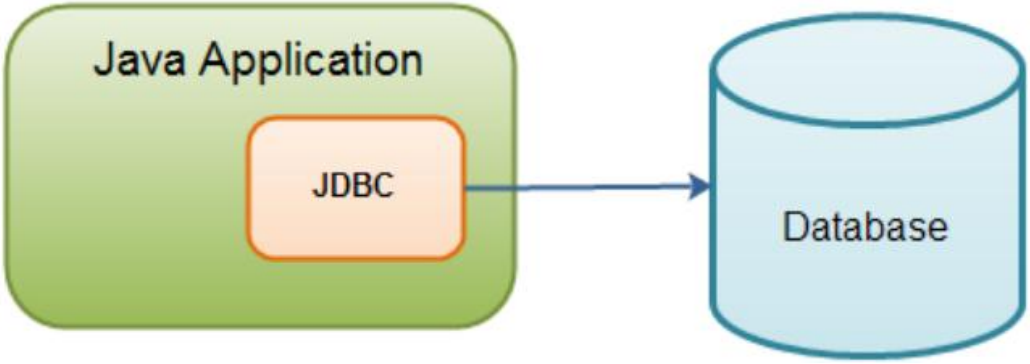
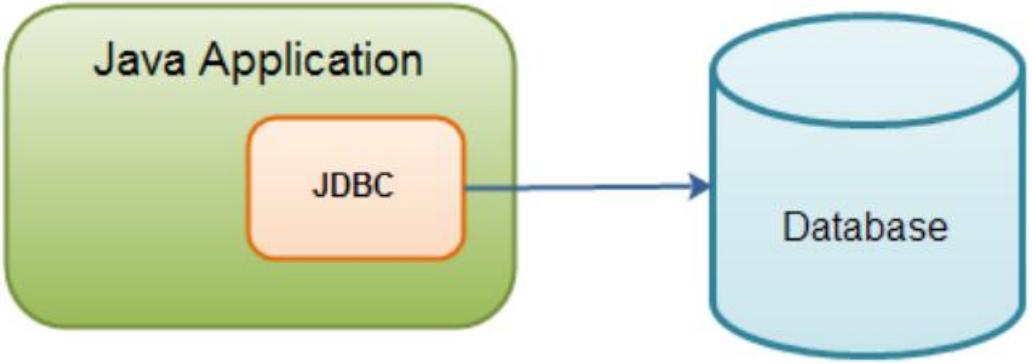

JDBC
Java Database Connection

Una de las principales aplicaciones de cualquier lenguaje moderno es la posibilidad de utilizar datos pertenecientes a un sistema de base de datos. La dificultad del manejo de archivos y las facilidades de manejo de datos que ofrecen los sistemas gestores de base de datos (SGBDs) son los causantes de esta necesidad.



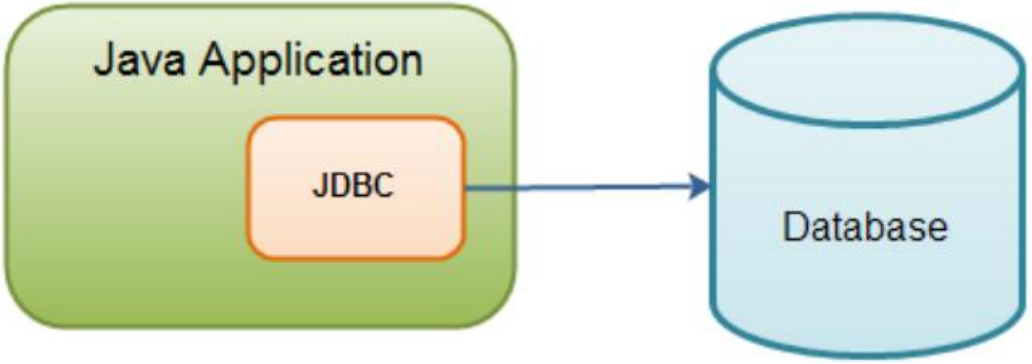
La API JDBC de Java (Java Database Connectivity) permite que las aplicaciones Java se conecten a bases de datos relacionales como MySQL, PostgreSQL, MS SQL Server, Oracle, H2 Database, etc. La API JDBC permite consultar y actualizar bases de datos relacionales. La API JDBC de Java es parte del SDK de Java SE, lo que hace que JDBC esté disponible para todas las aplicaciones Java que quieran usarlo.



JDBC es independiente de la base de datos

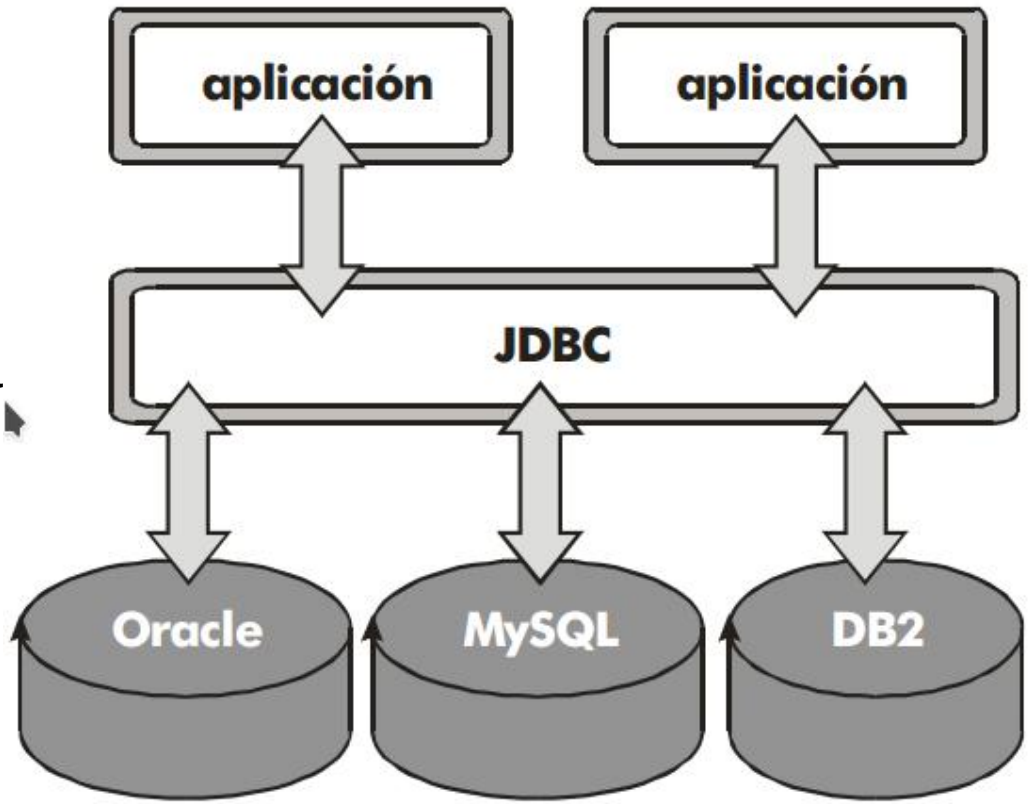
JDBC no es independiente de SQL

JDBC no es para bases de datos no relacionales

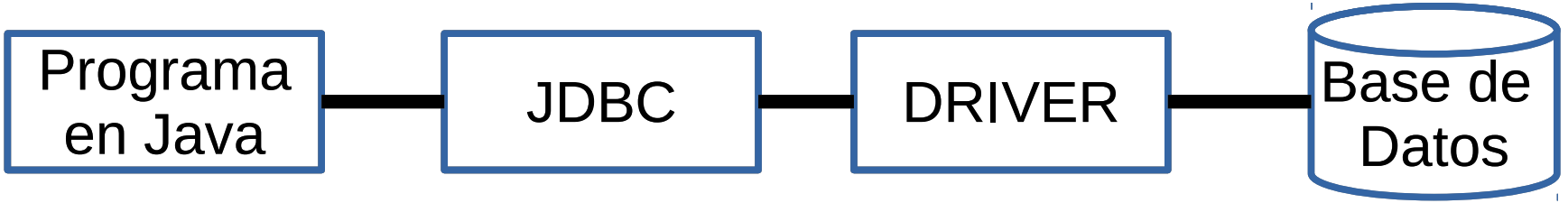


Estructura JDBC

Las aplicaciones sólo se comunican con la interfaz JDBC. Ésta es el encargada de comunicarse con los sistemas de base de datos.
Es necesario adquirir un controlador JDBC para el sistema gestor de base de datos que utilicemos. La comunicación fundamental entre las aplicaciones y JDBC se realiza a través de instrucciones SQL.



Relación entre una base de datos y un programa en Java



Nuestro programa envia las instrucciones SQL por medio de las clases JDBC

Hace parte de Java y nos facilita el acceso a la base de datos y la manipulación de las respuestas

Permite utilizar la base de datos desde JDBC

Empezar con JDBC

Instalar MySQL

Instalar Java y el JDBC en nuestra máquina

Instalar un driver en nuestra máquina.

Instalar nuestro Controlador de Base de Datos si es necesario.

Instalar Mysql

Si utilizas Linux Ubuntu:

Ir al gestor de paquetes e instalar mysql y mysql-workbench.

Luego acomodar los permisos de root y de tu usuario.

Configurar en ubuntu

→ ir a url con explicación completa ←

1) Ir al gestor de paquetes e instalar mysql y mysql-workbench. Usar esto evita muchos pasos.

2) Luego ajustar la autenticación y los privilegios del usuario.
Cambiar su método de autenticación de auth_socket a mysql_native_password para usar una contraseña para conectarse a MySQL como root. Para hacerlo, abra la indicación de MySQL desde su terminal:

```
$ sudo mysql
```

Configurar en ubuntu

3) Consulte cuál método de autenticación usa cada una de sus cuentas de usuario de MySQL usando el siguiente comando:

```
mysql> SELECT user, authentication_string, plugin, host FROM mysql.user;
```

Output

```
+------------------+-------------------------------------------+-----------------------+-----------+
| user             | authentication_string                     | plugin                | host      |
+------------------+-------------------------------------------+-----------------------+-----------+
| root             |                                           | auth_socket           | localhost |
| mysql.session    | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE | mysql_native_password | localhost |
| mysql.sys        | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE | mysql_native_password | localhost |
| debian-sys-maint | *CC744277A401A7D25BE1CA89AFF17BF607F876FF | mysql_native_password | localhost |
+------------------+-------------------------------------------+-----------------------+-----------+
4 rows in set (0.00 sec)
```

Configurar en ubuntu

4) Para configurar la cuenta root para autenticarse usando una contraseña, ejecute el siguiente comando ALTER USER.

```
mysql> ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'password';
```

Donde password es tu clave elegida

5) Luego, ejecute FLUSH PRIVILEGES (purgar privilegios)

```
mysql> FLUSH PRIVILEGES;
```

Configurar en ubuntu

6) Vuelva a verificar los métodos de autenticación que usa cada uno de sus usuarios para confirmar que root ya no se autentica usando el plugin auth_socket:

```
mysql> SELECT user, authentication_string, plugin, host FROM mysql.user;
```

Output

```
+------------------+-------------------------------------------+-----------------------+-----------+
| user             | authentication_string                     | plugin                | host      |
+------------------+-------------------------------------------+-----------------------+-----------+
| root             | *3636DACC8616D997782ADD0839F92C1571D6D78F | mysql_native_password | localhost |
| mysql.session    | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE | mysql_native_password | localhost |
| mysql.sys        | *THISISNOTAVALIDPASSWORDTHATCANBEUSEDHERE | mysql_native_password | localhost |
| debian-sys-maint | *CC744277A401A7D25BE1CA89AFF17BF607F876FF | mysql_native_password | localhost |
+------------------+-------------------------------------------+-----------------------+-----------+
4 rows in set (0.00 sec)
```

Configurar en ubuntu

7) En este resultado de ejemplo, puede ver que, ahora, el usuario root de MySQL se autentica usando una contraseña. Una vez que confirme esto en su propio servidor, puede salir del shell de MySQL:

```
mysql> exit
```

8) Cree un nuevo usuario y use una contraseña sólida

```
$ mysql -u root -p
```

```
mysql> CREATE USER 'sammy'@'localhost' IDENTIFIED BY
'password';
```

Configurar en ubuntu

9) Dele a su nuevo usuario los privilegios adecuados

```
mysql> GRANT ALL PRIVILEGES ON *.* TO 'sammy'@'localhost'
WITH GRANT OPTION;
```

10) Salga del Shell de MySQL

```
mysql> exit
```

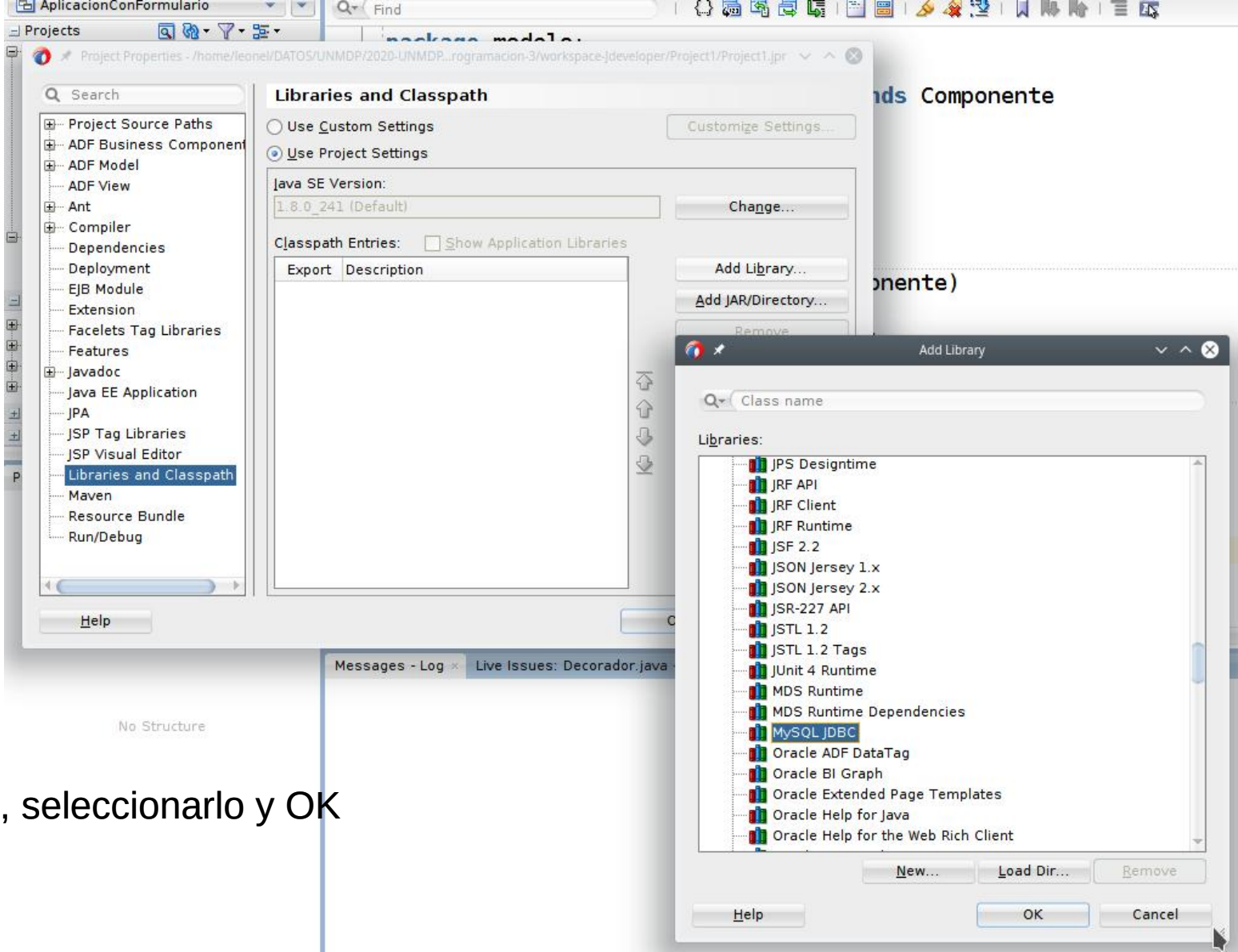
Instalar JDBC

En el proyecto, *botón derecho → propiedades.*

Luego *buscar Libraries and Classpath*

Luego *add Library*

Buscar MySQL JDBC, seleccionarlo y OK

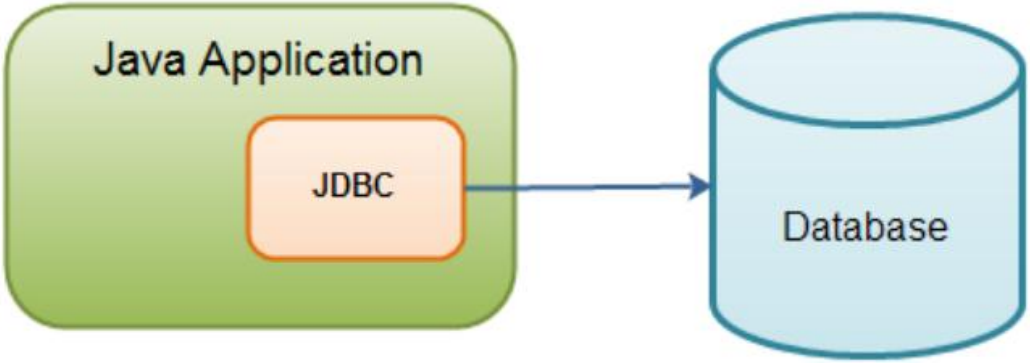


Usar JDBC

El código Java debe incluir ciertas instrucciones para establecer una conexión y comunicación con una base de datos a través de JDBC.

Dichas instrucciones deben considerar los siguientes aspectos:

1) Cargar los Drivers
2) Hacer una conexión
3) Crear sentencias JDBC
4) Ejecutar sentencias
5) Cerrar conexión



EJEMPLO

Parte 1

1) Cargar los Drivers
2) Hacer una conexión
3) Crear sentencias JDBC
4) Ejecutar sentencias
    1) Crear una tabla
    2) Insertar filas
5) Cerrar conexión
    1) Se verá en la segunda parte

Las partes más interesantes del código son las que se van a revisar a continuación, profundizando en cada uno de los pasos.

Lo primero que se hace es importar toda la funcionalidad de JDBC, a través de la primera sentencia ejecutable del programa.

```
import java.sql.*;
```

EJEMPLO
Parte 1


```
23  public class BaseDeDatos extends Observable
24  {
25      Connection conexion;
26      Statement sentencia;
27      PreparedStatement ps;
28      ResultSet resultado;
29
30      public BaseDeDatos()
31      {
32      }
33
34      public void cargarDriver()
35      {
36          try
37          {
38              // Se carga el driver jdbc
39              Class.forName("com.mysql.cj.jdbc.Driver");
40          }
41          catch (ClassNotFoundException ex)
42          {
43              Logger.getLogger(BaseDeDatos.class.getName()).log(Level.SEVERE, null, ex);
44          }
      }
```

Las siguientes líneas son las que cargan el driver jdbc mediante el método *forName()* de la clase **Class**.

```
33      public void cargarDriver()
34      {
35          try
36          {
37              // Se carga el driver jdbc
38              Class.forName("com.mysql.cj.jdbc.Driver");
39          }
40          catch (ClassNotFoundException ex)
41          {
42              Logger.getLogger(BaseDeDatos.class.getName()).log(Level.SEVERE, null, ex);
43          }
44      }
```

El método ***forName()*** localiza, lee y enlaza dinámicamente una clase determinada. La sintaxis recomendada de ***forName()*** es *nombreEmpresa.nombreBaseDatos.nombreDriver*

El *driver* deberá estar ubicado en el directorio *nombreEmpresa\nombreBaseDatos\ nombreDriver.class* a partir del directorio indicado por la variable de entorno *CLASSPATH*.

Si por cualquier motivo no es posible conseguir cargar **JdbcOdbcDriver.class**, se intercepta la excepción y se sale del programa.

A continuación, se solicita a **DriverManager** que proporcione una conexión para una fuente de datos JDBC. El parámetro **jdbc:mysql://localhost/Agenda** especifica que la intención es acceder a la fuente de datos con nombre **Agenda**

```
public void conectar()
{
    try
    {
        // Se establece la conexión con la base de datos
        conexion = DriverManager.getConnection("jdbc:mysql://localhost/Agenda", "root", "leonel127");
        System.out.println("Agenda Conectada");
    }
    catch (SQLException ex)
    {
        Logger.getLogger(BaseDeDatos.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

El segundo y tercer parámetro son el nombre del usuario y la clave con la cual se intentará la conexión. Esto depende de cada instalación.

Luego de hacer una operación contra la base de datos se procede a su desconexión.

```
public void desconectar()
{
    try
    {
        conexion.close();
        System.out.println("Desconectando de BD Agenda");
    }
    catch (SQLException ex)
    {
        Logger.getLogger(BaseDeDatos.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Cerrando las conexiones se garantiza que se actualice la información en la base de datos. Hasta este momento no hay seguridad de que esté actualizada.
Se va utilizando una memoria cache.

Una vez establecida la conexión contra la base de datos, se puede crear una Tabla:

```
public void crearTabla() throws SQLException
{
   conectar();
   System.out.println("se crea la tabla vacía");
   sentencia = conexion.createStatement();
   // Esto es código SQL
   boolean execute =
      sentencia.execute("CREATE TABLE AMIGOS (" + " NOMBRE VARCHAR(15) NOT NULL, " +
                        " APELLIDOS VARCHAR(30) NOT NULL, " + " CUMPLE DATETIME) ");
   sentencia.close();
   desconectar();
}
```

Se ejecuta una sentencia SQL que crea la tabla *AMIGOS* con tres campos: *NOMBRE, APELLIDOS* y *CUMPLE*. De ellos, únicamente el tercero, correspondiente al cumpleaños, es el que puede ser desconocido, es decir, puede contener valores nulos.

Se solicita que proporcione un objeto de tipo *Statement* para poder ejecutar sentencias a través de esa conexión.

Para ello se dispone de los métodos *execute(String sentencia)* para ejecutar una petición SQL que no devuelve datos o *executeQuery(String sentencia)* para ejecutar una consulta SQL. Este último método devuelve un objeto de tipo *ResultSet*.

Una vez que se tiene el objeto *Statement* ya se pueden lanzar consultas y ejecutar sentencias contra el servidor. A partir de aquí el resto del programa realmente es SQL «adornado»

```java
public void destruirTabla()
{
    conectar();
    try
    {
        sentencia = conexion.createStatement();
        sentencia.execute("DROP TABLE AMIGOS");
        sentencia.close();
    }
    catch (SQLException e)
    {
        System.out.println("no existe tabla");
    }
    finally
    {
        desconectar();
    }
}
```

se ejecuta *DROP TABLE AMIGOS* para borrar la tabla.

Puede ser que la tabla AMIGOS no exista, dando como resultado una excepción, se aísla la *sentencia.execute()* mediante un *try* y un *catch*.

```java
public boolean existeTablaAmigos()
{
    boolean rta = false;
    try
    {
        conectar();
        DatabaseMetaData md = conexion.getMetaData();
        ResultSet rs = md.getTables(null, null, "AMIGOS", null);
        if (rs.next())
        {
            rta = true;
        }
    }
    catch (SQLException ex)
    {
        Logger.getLogger(BaseDeDatos.class.getName()).log(Level.SEVERE, null, ex);
    }
    return rta;
}
```

Este código permite determinar si existe la tabla Amigos

```
public void guardar(Amigo amigo) throws SQLException
{
   conectar();
   String sql =
      "INSERT INTO AMIGOS (`NOMBRE`, `APELLIDOS`, `CUMPLE`) VALUES ('" + amigo.getNombre() + "','" +
      amigo.getApellido() + "','" + amigo.getCumple() + "')";
   System.out.println(sql);
   sentencia = conexion.createStatement();
   sentencia.execute(sql);
   desconectar();
   setChanged();
   notifyObservers();
}
```

Se ejecutan sentencias *INSERT* para rellenar con datos la tabla. En todo momento se ha colocado un *try ... catch* exterior para interceptar cualquier excepción que puedan dar las sentencias. En general, para *java.sql* está definida una clase especial de excepciones que es ***SQLException***. Se obtendrá una excepción de este tipo cuando ocurra cualquier error de proceso de JDBC.

EJEMPLO

Parte 2

1)Cargar los Drivers

2)Hacer una conexión

3)Crear sentencias JDBC

4)Ejecutar sentencias

   1) Obtener información a partir de una BD creada.

5)Cerrar conexión

EJEMPLO

Parte 2

Solamente se mostrarán las partes de código significativas

obtener información a partir de una base de datos ya creada

Se utiliza *executeQuery* para obtener el resultado de *SELECT * FROM AMIGOS*. Mediante *resultado.next()* la posición se situará en el «siguiente» elemento del resultado, o bien sobre el primero si todavía no se ha utilizado. La función *next()* devuelve *true* o *false* si el elemento existe, de forma que se puede iterar mediante *while (resultado.next())* para tener acceso a todos los elementos.

```
public ArrayList listar()
{
   ArrayList listaDeAmigos = null;
   try
   {
      conectar();
      listaDeAmigos = new ArrayList();
      Amigo amigo = null;
      String sql = "SELECT * FROM AMIGOS";
      sentencia = conexion.createStatement();
      ResultSet rows = sentencia.executeQuery(sql);
      while (rows.next())
      {
         amigo = new Amigo();
         amigo.setNombre(rows.getString("NOMBRE"));
         amigo.setApellido(rows.getString("APELLIDOS"));
         amigo.setCumple(rows.getDate("CUMPLE"));
         listaDeAmigos.add(amigo);
      }
      return listaDeAmigos;
   }
   catch (SQLSyntaxErrorException ex)
```

El método anterior se completa con las siguientes líneas de código:

```
    catch (SQLSyntaxErrorException ex)
    {
        Logger.getLogger(BaseDeDatos.class.getName()).log(Level.SEVERE, null, ex);
    }
    catch (SQLException ex)
    {
        Logger.getLogger(BaseDeDatos.class.getName()).log(Level.SEVERE, null, ex);
    }
    finally
    {
        desconectar();
        return listaDeAmigos;
    }
}
```

Aclaraciones:

Los métodos conectar y desconectar se han utilizado de manera tal vez exagerada, con el fin de modularizar cada aspecto (alta, baja y consulta)

Notar que a veces se produce una desconexión seguida de una conexión, sin nada en medio. Todo depende del orden en que se efectúan los eventos.

El resto del código corresponde a otros aspectos que no tienen que ver con JDBC.

Podrían haberse aplicado más patrones, pero la intención fue solamente destacar el uso de JDBC.