JDBC: EL PUENTE ENTRE JAVA Y LAS BASES DE DATOS

JDBC: The bridge between Java and the databases

Carlos Alberto Vanegas*

Resumen

En este articulo se expone la API JDBC (Java DataBase Connetivity), la cual permite el puente de comunicación entre un programa realizado en el lenguaje de programación Java y una Base de Datos. Se explica los diferentes tipos de driver, la forma como JDBC accesa a una base de datos, así como también las diferentes clases que contiene dicha API. Por último se realiza un ejemplo práctico donde se hace una conexión a una base de datos de Access y se manipulan los registros de una tabla realizando los procesos de: consulta, modificación, borrado e inserción de un registros.

Palabras clave: objeto, subprograma, contenedor, bases de datos, navegador, servidor, clave.

Abstract

In this paper the API JDBC it is exposed (Java DataBase Connetivity), which allows the communication bridge between a program carried out in the programming language Java and a database. Explain the different driver types, the form like JDBC access to a database, as well as the different class that this API contains. Lastly, is carried out a practical example where a connection is made to a database of Access and the register are

* Ingeniero de Sistemas de la Universidad Incca de Colombia. Especialista en Ingeniería de Software de la Universidad Distrital Francisco José de Caldas, Magíster en Ingeniería de Sistemas de la Universidad Nacional de Colombia. Docente de Planta de la Universidad Distrital Francisco José de Caldas adscrito a la Facultad Tecnológica. Correo electrónico: cavanegas@udistrital.edu.co

manipulated by chart carrying out the processes of: consults, modification, erased and insert of a register.

Key words: object, applets, driver, databases, browser, server, password

1. Introducción

Al navegar en el World Wide Web, es fácil darse cuenta de que existe mucha información. Muchas compañías están usando bases de datos relacionales[1] para manejar la información en sus sitios Web. Por ejemplo, la mayoría de los buscadores de Internet usan este tipo de base de datos. Las bases de datos relacionales son ideales para el almacenamiento de grandes cantidades de información, la cual puede ser accesada por muchos usuarios.

Hoy en día, la mayoría de los manejadores de bases de datos relacionales tienen soporte para la utilización de interface HTML. Para algunas aplicaciones, las páginas HTML favorecen la interfaz, pero, en aplicaciones más complejas se presentan ciertas limitaciones que no permiten generar un buen trabajo. En estos casos, es conveniente la utilización de lenguajes de programación como Java, que permitan elaborar aplicaciones para generar una mejor interfaz con la base de datos.

2. La API JDBC

JDBC (Java Database Connectivity) es un API(Application Programming Interface) de Java, que contiene un conjunto de objetos y métodos que permiten crear una interfaz para

[1] conjunto de tablas relacionadas entre sí, cada tabla esta definida por una serie de campos.

realizar la comunicación con una base de datos por medio de aplicaciones y/o applets[2] creados en Java.

JDBC permite que programas escritos en el lenguaje de programación Java ejecuten instrucciones en el lenguaje estándar de acceso a bases de datos SQL[3] (Structured Query Language, Lenguaje estructurado de consultas). Dado que casi todos los sistemas de administración de bases de datos relacionales (DBMS) soportan SQL, y que los programas realizados en Java se ejecutan en la mayoría de las plataformas; JDBC hace posible escribir una sola aplicación que permita la interacción con una base de datos que se pueda ejecutar en diferentes plataformas e interactuar con distintos DBMS. En otras palabras, con el API JDBC no es necesario escribir un programa para accesar a Access, otro programa para accesar a Oracle, y otro para accesar a MySQL; con esta API, se puede crear un sólo programa que sea capaz de enviar sentencias SQL a la base de datos apropiada.

Con la ayuda de JDBC, la habilidad de Java para integrarse con DBMS comerciales y su naturaleza orientada al manejo de la Red, es posible crear un ambiente ideal tipo cliente-servidor.

De una manera muy simple, al usar JDBC se pueden hacer tres cosas:

- Establecer la conexión a una base de datos, ya sea remota o no
- Enviar sentencias SQL a esa base de datos
- Procesar los resultados obtenidos de la base de datos.

[2] Son subprogramas de Java independientes del dispositivo, es decir un mismo Applet puede ejecutarse desde un PC con Windows/x, Linux, ya que el compilador de Java genera código de maquina virtual el cual se puede ejecutar desde cualquier tipo de procesador.

[3] Es un lenguaje de base de datos normalizado, utilizado por los diferentes motores de bases de datos para realizar determinadas operaciones sobre los datos o sobre la estructura de los mismos.

La aplicación de Java debe tener acceso a un controlador (driver[4]) JDBC adecuado. Este controlador es el que implementa la funcionalidad de todas las clases de acceso a datos y proporciona la comunicación entre el API JDBC y la base de datos real.

Los distribuidores de bases de datos suministran los controladores que implementan el API JDBC y que permiten acceder a sus propias implementaciones de bases de datos. De esta forma JDBC proporciona a los programadores de Java una interfaz de alto nivel y les evita el tener que tratar con detalles de bajo nivel para acceder a las bases de datos.

2.1 Tipos de Driver

- **Puente JDBC-ODBC**: este driver convierte todas las llamadas JDBC a llamadas ODBC y realiza la conversión correspondiente de los resultados.
- **Java / Binario:** este driver se salta la capa ODBC y se conecta directamente con la librería nativa del fabricante del sistema DBMS (como pudiera ser DB-Library para Microsoft SQL Server). Este driver es 100% Java pero aún así necesita la existencia e un código binario (la librería DBMS) en la máquina del cliente.**100% Java / Protocolo Nativo:** es un driver realizado completamente en Java que se comunica con el servidor DBMS utilizando el protocolo de red nativo del servidor. El driver no necesita intermediarios para hablar con el servidor. Convierte todas las peticiones JDBC en peticiones de red contra el servidor.
- **100% JAVA / Protocolo Independiente:** en este caso, el driver JDBC hace las peticiones de datos al intermediario con un protocolo de red independiente del

[4] Son los encargados de actuar como interfaz entre el sistema operativo y los dispositivos que componen un computador. De esta forma todos los componentes del PC se entienden y trabajan conjuntamente.

servidor DBMS. El intermediario (ubicado en el lado del servidor) a su vez, convierte las peticiones JDBC en peticiones nativas del sistema DBMS.

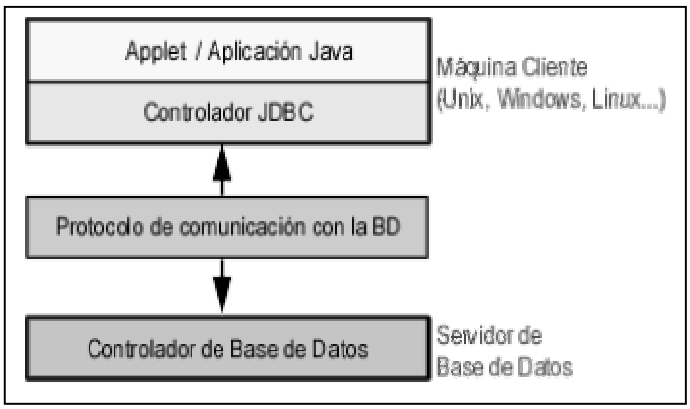
2.2 Acceso de JDBC a Bases de Datos

El API JDBC soporta dos modelos diferentes de acceso a Bases de Datos, los modelos de dos y tres capas.

Modelo de dos capas

Este modelo se basa en que la conexión entre la aplicación Java o el applet que se ejecuta en el navegador, se conectan directamente a la base de datos. Esto significa que el driver JDBC específico para conectarse con la base de datos, debe residir en el sistema local. La base de datos puede estar en cualquier otra máquina y se accede a ella mediante la red. Esta es la configuración típica *Cliente/Servidor*: el programa cliente envía instrucciones SQL a la base de datos, ésta las procesa y envía los resultados de vuelta a la aplicación.

Figura 1: Modelo de dos Capas



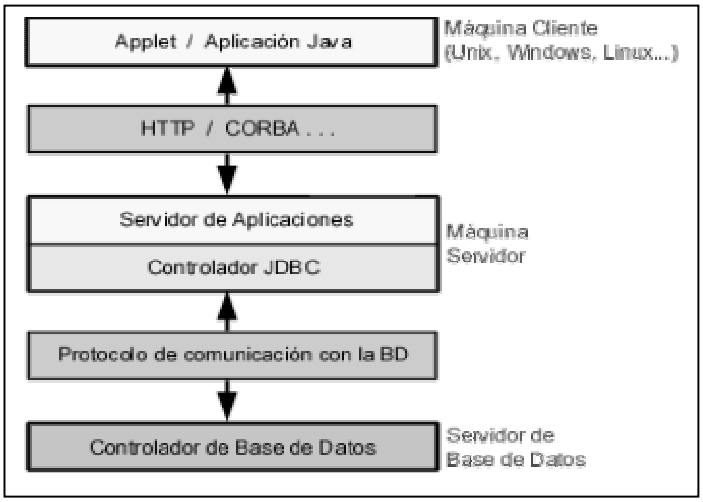
Fuente: http://www.itapizaco.edu.mx/paginas/JavaTut/froufe/parte21/cap21-3.html

Modelo de tres capas

En este modelo de acceso a las bases de datos, las instrucciones son enviadas a una capa intermedia entre Cliente y el Servidor, que es la que se encarga de enviar las sentencias SQL a la base de datos y recoger el resultado desde la base de datos. En este caso el usuario no tiene contacto directo, ni a través de la red, con la máquina donde reside la base de datos.

La ventaja adicional de este modelo es que los drivers JDBC no tienen que residir en la máquina cliente, lo cual libera al usuario de la instalación de cualquier tipo de driver.

Figura 2: Modelo de tres capas



Fuente: http://www.itapizaco.edu.mx/paginas/JavaTut/froufe/parte21/cap21-3.html

2.3 Herramientas y conceptos básicos para el acceso de JDBC a una Bases de Datos

Para que una aplicación de Java pueda interactuar con una base de datos es necesario contar con las siguientes herramientas:

- Tener instalado en el computador el kit de desarrollo de Java jdk1.2 o superior el cual incluye el API JDBC, este kit se encuentra en la dirección electrónica *java.sun.com*.
- Una base de datos (Access, Oracle, Mysql, etc.)
- Un controlador JDBC que permita realizar la conexión con la base de datos.

Teniendo en cuenta todas las herramientas mencionadas, y suponiendo que el usuario tiene los conocimientos básicos de la compilación y la ejecución de un programa Java, conoce las sentencias de SQL, y además maneja una base de datos (es necesario contar con el driver de la base de datos que va a utilizar). Se debe tener en cuenta lo siguiente:

- **Importar el paquete sql**: Este paquete contiene las diferentes clases para la manipulación de la base de datos como son:
    - Driver: Permite conectarse a una Base de Datos: cada gestor de Base de Datos requiere 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 Base de Datos. Una aplicación puede tener más de una conexión a más de una Base de Datos.
    - DatabaseMetadata: Proporciona información acerca de una Base de Datos, como las tablas que contiene, etc.
    - 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 las filas o registros obtenidos al ejecutar un SELECT.
- ResultSetMetadata: Permite obtener información sobre un ResultSet, como el número de columnas, sus nombres, etc.

- **Establecer una Conexión**: para establecer una conexión con el controlador de base de datos que queremos utilizar, se deben realizar dos pasos:
  - **Cargar los Drivers**: para cargar el driver o drivers que queremos utilizar se requiere de una línea de código. Si, por ejemplo, queremos utilizar el puente JDBC-ODBC, se cargaría la siguiente línea de código:

    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  - **Hacer la Conexión:** El segundo paso para establecer una conexión corresponde a tener el driver apropiado conectado al controlador de base de datos. Este proceso puede resolverse como se observa en la siguiente línea de código:

Connection conexion= DriverManager.getConnection(url, "Login", "Password");

Si estamos utilizando el puente JDBC-ODBC, el URL empezará con **jdbc:odbc:,** el resto de la URL normalmente es la fuente de nuestros datos o el sistema de base de datos. Por eso, si estamos utilizando ODBC para acceder a una fuente de datos ODBC llamada "**Mijdbc**" por ejemplo, nuestro URL podría ser **jdbc:odbc:Mijdbc**. En lugar de "**Login**" colocaríamos el nombre utilizado para entrar en el controlador

de la base de datos; en lugar de "**Password**" colocaríamos nuestra clave para el controlador de la base de datos. Por eso si entramos en el controlador con el nombre "**Mibase**" y el password "**Datos,**" estas dos líneas de código establecerán una conexión:

```
String url = "jdbc:odbc:Mijdbc";
Connection conexion= DriverManager.getConnection(url, "Mibase", "Datos");
```

Si uno de los drivers que hemos cargado reconoce la URL suministrada por el método **DriverManager.getConnection**, dicho 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.

- **Seleccionar las Tablas:** Un objeto Statement es el que envía nuestras sentencias SQL al controlador de la base de datos. Simplemente creamos un objeto Statement y lo ejecutamos, suministrando el método SQL apropiado con la sentencia SQL que queremos enviar. Para una sentencia SELECT, el método a ejecutar es executeQuery. Para sentencias que crean o modifican tablas, el método a utilizar es executeUpdate.Se toma un ejemplar de una conexión activa para crear un objeto Statement. En el siguiente ejemplo, utilizamos nuestro objeto Connection: *conexion* crear el objeto Statement: declaracion

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

En este momento el objeto **declaracion** existe, pero no tiene ninguna sentencia SQL que pasarle al controlador de la base de datos. Necesitamos suministrarle el método que utilizaremos para ejecutar el objeto **declaracion**. Por ejemplo, en el siguiente fragmento de código, suministramos **executeUpdate** con la sentencia SQL para crear una tabla:

declaracion.executeUpdate*("CREATE TABLE Clientes " +"(IDENTIFICACION VARCHAR(12), NOMBRE VARCHAR(20), CREDITO INTEGER, CIUDAD VARCHAR(20))");*

3. Ejercicios práctico de acceso a una Base de Datos con JDBC

A continuación se realizara un ejemplo de un aplicación Java que permita el acceso a una base de datos, donde los programas de Java permitirán la conexión, visualización, modificación, eliminación e inserción de un registro. También se visualizarán los campos y el tipo de dato de la tabla. Para realizar este proceso ges necesario conocer conceptos básicos de Java, la forma como se crea una tabla en Access, como también conocer las diferentes sentencias SQL.

Ejemplo 1: Hacer un programa que utilice JDBC, que permita realizar la conexión a una base de datos realizada en Access.

Para implementar el programa se deben realizar los siguientes pasos:

1. Crear una Base de Datos en Access llamada **Ingeniería:** dentro de la base de datos se crea una tabla llamada **Clientes,** con la siguiente estructura:

- identificacion Texto 13
- nombres Texto 25
- apellidos Texto 25
- direccion Texto 25
- telefono Texto 15
- ciudad Texto 20
- credito numérico

La tabla con 6 registros quedara de la siguiente forma:

Figura 3: Registros de la tabla Clientes

Clientes : Tabla

identificacion	nombres	apellidos	direccion	telefono	ciudad	credito
1	Carlos Alberto	Rubiano	calle 2c # 45-35	4714747	bogota	100000
2	Rosa	Cetina	Cra 59 # 24-11 sur	6598471	tunja	500000
3	Cristian	Vanegas	calle 2e # 44-11	2145893	bogota	250000
5	Rocio	Varela	calle 24 # 55-55	3214568	cali	300000
6	Claudia	Pinzon	calle 31 # 45-88	1255478	tunja	50000
8	Pedro	Celio	Cra 25 # 22-15	2154869	Manizales	215000
						0

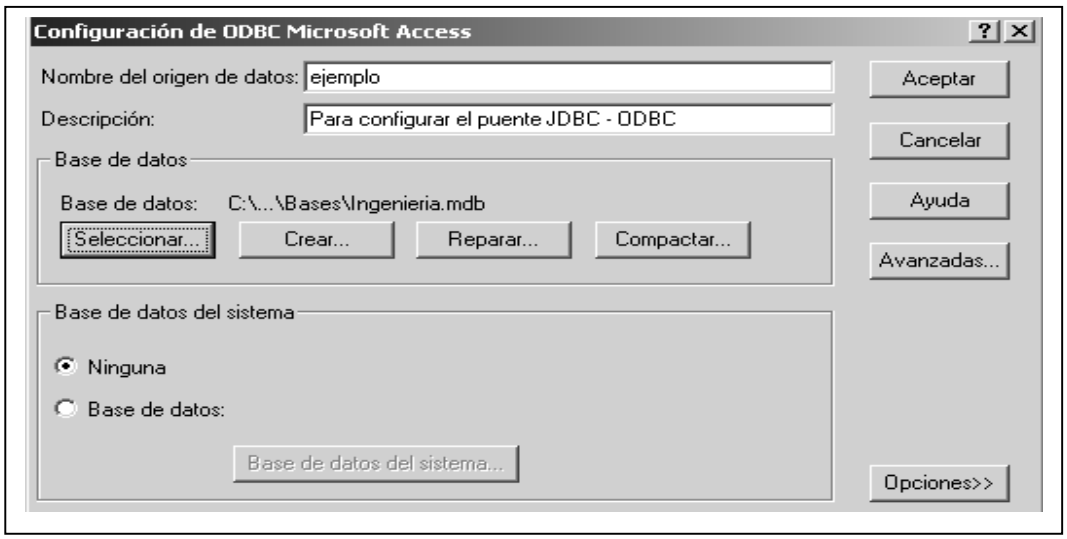
fuente: elaboración propia

2. Crear el puente ODBC::JDBC: se debe abrir el panel de control y escoger el icono "Fuente de Datos ODBC", el cual mostrará la ventana "Administrador de orígenes de datos ODBC".

Figura 4: Administrador de orígenes de datos ODBC

En la página DSN de Usuario se pulsa el botón "Agregar", se selecciona el driver que se necesita, en este caso "Controlador para Microsoft Access (*.mdb)". A continuación se vera la ventana de configuración de ODBC Microsoft Access:

Figura 5: Configuración de ODBC



En la opción origen de datos se escribe "ejemplo" (para nuestro caso). En el botón "Seleccionar", se selecciona la base de datos "Ingeniería" y se pulsa el botón "Aceptar". Si desea en la opción "Avanzadas" puede colocar el usuario y el password. Por último se cierra el Administrador ODBC con la opción “Aceptar”.

3. Crear el programa fuente: PruebaConexion.java

```
1  import java.sql.*;
2  public class PruebaConexion {
3    public static void main(String[] args){
4      try
      {
5       Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
6       String basededatos = "jdbc:odbc:ejemplo";
7       Connection conexion = DriverManager.getConnection(basededatos);
8       Statement sentencia= conexion.createStatement();
9     ResultSet clientes = sentencia.executeQuery("SELECT * FROM
Clientes");
10      clientes.close();
11      conexion.close();
12      sentencia.close();
      }
13    catch (ClassNotFoundException e) {
14      System.out.println("Clase no encontrada");
     }
15  catch (SQLException e) {
16  System.out.println(e);
       }
17    System.out.println("Sin error al conectar");
      }
   }
```

Línea 1: se importa el paquete sql

Línea 5: se carga el drive JDBC-ODBC

Línea 6: se crea un objeto *basededatos* de tipo String para guardar el nombre de los datos de origen del administrador ODBC.

Línea 7: se crea un objeto *conexión* de tipo Connection que permitirá la conexión a la base de datos. Para esto de utiliza el método getConnection (el cual recibe como parámetro el nombre de origen de los datos) perteneciente a la clase DriveManager.

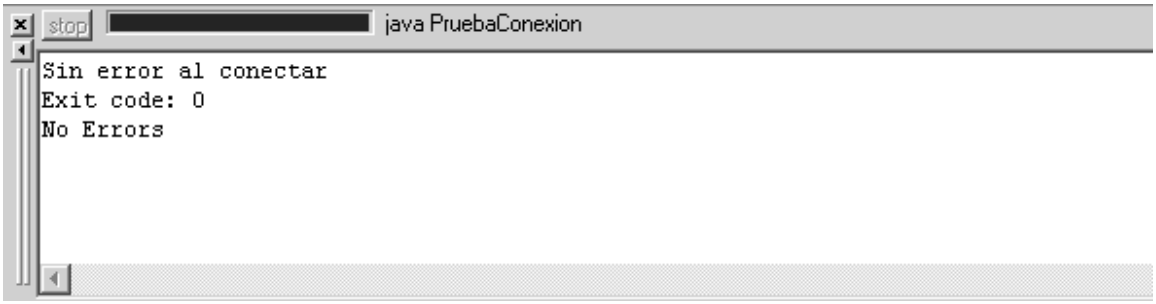
Línea 8: se crea un objeto *sentencia* de tipo Statement que sirve para consultar la base de datos.

Línea 8: se crea un objeto *clientes* de tipo ResulSet, donde se ejecuta el método executeQuery de la interfaz Statement para ejecutar una consulta de todos los datos que contenga la tabla clientes.

Línea 10, 11, 12: se liberar todos los recursos empleados utilizando el método close() de la interfaz ResulSet.

Al compilar y ejecutar el programa se visualizará la siguiente ventana:

Figura 6: Ventana de la prueba de la conexión a la base de datos.



Fuente: elaboración propia

Ejemplo 2: Hacer un programa que utilice JDBC, que permita la consulta de todos los registros de la tabla Clientes.

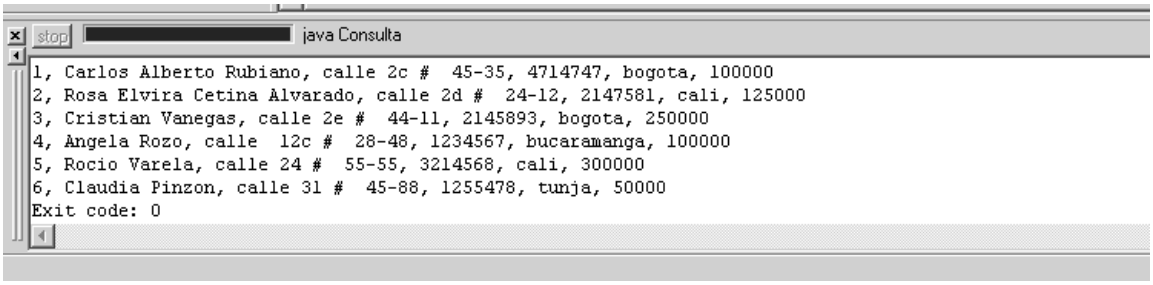
```
1  import java.sql.*;
2  public class Consulta{
3     public static void main(String[] args){
4      String id,nombre,apellido,ciudad,telefono,direccion;
5      int credito;
6      try{
7       Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
8       String basededatos = "jdbc:odbc:ejemplo";
9       Connection conexion = DriverManager.getConnection(basededatos);
10      Statement sentencia = conexion.createStatement();
11      ResultSet clientes = sentencia.executeQuery("SELECT * FROM
      Clientes");
12      while (clientes.next()) {
13       id = clientes.getString("identificacion");
14       nombre = clientes.getString("nombres");
15       apellido = clientes.getString("apellidos");
```

```
16       direccion = clientes.getString("direccion");
17       telefono = clientes.getString("telefono");
18       ciudad = clientes.getString("ciudad");
19       credito = clientes.getInt("credito");
20    system.out.println(id+", "+nombre+" "+apellido+", "+direccion+",
   "+telefono+", "+ciudad+", "+credito);
         }
21     clientes.close();
22     conexion.close();
23     Sentencia.close();
       }
24     catch (ClassNotFoundException e) {
25      System.out.println("Clase no encontrada");
        }
26     catch (SQLException e) {
27      System.out.println(e);
    }
  }
}
```

Línea 12-20: se crea un ciclo donde se utilizan las variables de tipo String y int para asignarle el valor correspondiente con el método getString() y getInt() de la tabla Clientes.

Al compilar y ejecutar el programa se visualizará la siguiente ventana:

Figura 7: Ventana de la consulta de los registros de la tabla Clientes.



Fuente: elaboración propia

Ejemplo 3: Hacer un programa que utilice JDBC, que permita realizar una modificación a un registro de la tabla Clientes.

```
1  import java.sql.*;
2  public class Modificacion {
3    public static void main(String[] args){
4         String id,nombre,apellido,direccion, telefono, ciudad;
5         int credito=0;
6      try{
```

```
7       Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
8       String basededatos = "jdbc:odbc:ejemplo";
9       Connection conexion = DriverManager.getConnection(basededatos);
10      Statement sentencia = conexion.createStatement(
        ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
11     ResultSet clientes = sentencia.executeQuery("SELECT * FROM
       Clientes");
12      while (clientes.next()) {
13        nombre = clientes.getString("nombres");
14        apellido = clientes.getString("apellidos");
15        id = clientes.getString("identificacion");
16        direccion = clientes.getString("direccion");
17        telefono = clientes.getString("telefono");
18        credito = clientes.getInt("credito");
19        ciudad = clientes.getString("ciudad");
20       if (apellido.equalsIgnoreCase("Cetina Alvarado")&&
            nombre.equalsIgnoreCase("Rosa Elvira"))
         {
21         clientes.updateString("identificacion","2");
22         clientes.updateString("nombres","Rosa");
23         clientes.updateString("apellidos","Cetina");
24         clientes.updateString("direccion","Cra 59 # 24-11 sur");
25         clientes.updateString("telefono","6598471");
26         clientes.updateString("ciudad","tunja");
27         clientes.updateInt("credito",500000);
28         clientes.updateRow();
29         System.out.println(id+", "+nombre+" "+apellido+",
         "+direccion+", "+telefono+", "+ciudad+", ");
30         System.out.println("Registro modificado");
31         break;
         }
32     Else
 {
           System.out.println("Registro no encontrado");
 }
33    }
     clientes.close();
     conexion.close();
     sentencia.close();
    }
    catch (ClassNotFoundException e) {
      System.out.println("Clase no encontrada");
    }
    catch (SQLException e) {
      System.out.println(e);
    }
  }
}
```

Línea 4, 5: se crean las variables para obtener la información de un registro

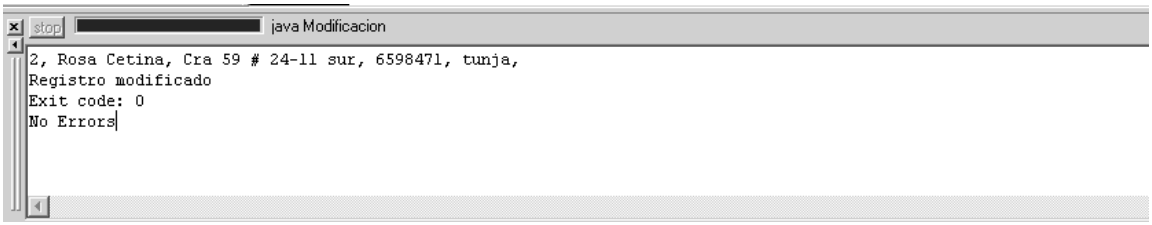
Línea 10: el método createStatement recibe dos parámetros: el primero *ResultSet.TYPE_SCROLL_INSENSITIVE*, especifica que el cursor puede desplazarse en

cualquier dirección y que los cambios realizados al objeto *ResulSet* se reflejaran cuando se consulte la base de datos por segunda vez; el segundo *ResultSet.CONCUR_UPDATABLE* permite la actualización del registro a ser modificado.

Línea 12-33: se crea un ciclo que permite obtener por cada registro el contenido de este, se verifica si existe el registro a ser modificado. Si existe se utilizada el método *updateString* y el método *updateInt* para modificar cada campo del registro; el método *updateRow()* de la clase *ResulSet* actualiza todos los campos del registro. Por último la instrucción *break* permite salir del ciclo.

Al compilar y ejecutar el programa se observará la siguiente ventana:

Figura 8: Ventana de la modificación de un registro en la tabla Clientes.



Fuente: elaboración propias

Ejemplo 4: Hacer un programa que utilice JDBC, que permita insertar un registro en la tabla Clientes.

```
1  import java.sql.*;
2  public class Insertar {
3    public static void main(String[] args){
4      try{
5       Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
6       String basededatos = "jdbc:odbc:ejemplo";
7       Connection conexion = DriverManager.getConnection(basededatos);
8       Statement sentencia = conexion.createStatement(
9       ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
10     ResultSet clientes = sentencia.executeQuery("SELECT * FROM
      Clientes");
11      clientes.moveToInsertRow();
12      clientes.updateString("identificacion","8");
13      clientes.updateString("nombres","Pedro");
14      clientes.updateString("apellidos","Celio");
15      clientes.updateString("direccion","Cra 25 # 22-15");
16      clientes.updateString("telefono","2154869");
```

```
17      clientes.updateString("ciudad","Manizales");
18      clientes.updateInt("credito",215000);
19      clientes.insertRow();
20      System.out.println("Registro insertado");
21      clientes.close();
22      conexion.close();
23      sentencia.close();
    }
    catch (ClassNotFoundException e) {
      System.out.println("Clase no encontrada");
    }
    catch (SQLException e) {
      System.out.println(e);
    }
  }
}
```

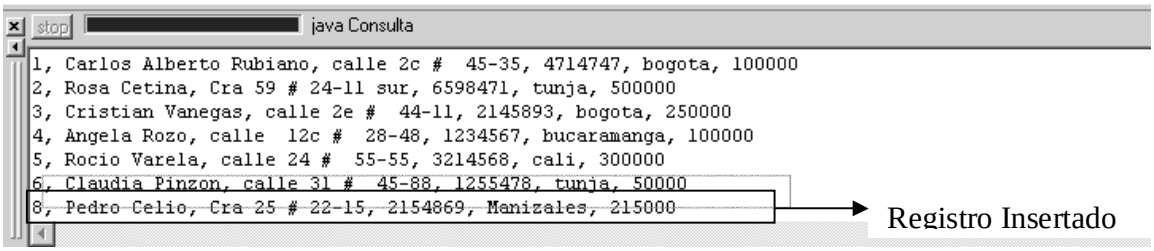
Línea 11: se utiliza el método *moveToInsertRow()* de la clase *ResulSet* el cual inserta un registro en blanco.

Línea 12-19: se actualiza cada campo del registro con el valor deseado. Por último se utiliza el método *insertRow()* de la clase *ResulSet* inserta el registro.

Utilizando el programa de consulta se visualiza el registro que ha sido insertado

Al compilar y ejecutar el programa se visualizará la siguiente ventana:

Figura 9: Ventana de la inserción de un registro en la tabla Clientes.



Fuente: elaboración propia

Ejemplo 5: Hacer un programa que utilice JDBC, que permita eliminar un registro en la tabla Clientes.

```
1  import java.sql.*;
2  public class Borrado {
3    public static void main(String[] args){
```

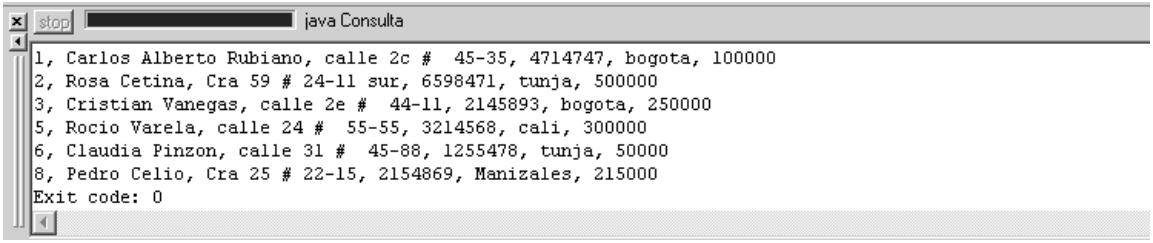
```
4    String nombre,apellido;
5    try{
6     Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
7     String basededatos = "jdbc:odbc:ejemplo";
8     Connection conexion = DriverManager.getConnection(basededatos);
9     Statement sentencia = conexion.createStatement(
10    ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
11    ResultSet clientes = sentencia.executeQuery("SELECT * FROM
        clientes");
12     while (clientes.next()) {
13       nombre = clientes.getString("Nombres");
14       apellido = clientes.getString("Apellidos");
15       if (apellido.equalsIgnoreCase("Rozo")&&
             nombre.equalsIgnoreCase("Angela")){
16        System.out.println("Registro eliminado");
17        clientes.deleteRow();
18        break;
       }
     }
20     clientes.close();
21     conexion.close();
22     sentencia.close();
    }
    catch (ClassNotFoundException e) {
      System.out.println("Clase no encontrada");
    }
    catch (SQLException e) {
      System.out.println(e);
    }
  }

}
```

Línea 12-18: se crea un ciclo para recorrer la tabla, buscando un nombre y un Apellido especifico, si se encuentra los datos se elimina el registro utilizando el método *deleteRow()* de la clase *ResulSet.*

Utilizando el programa de consulta se visualiza que el registro ha sido eliminado.

Figura 9: Ventana de la tabla Clientes, con un registro eliminado.



Fuente: elaboración propia

Ejemplo 6: Hacer un programa que utilice JDBC, y que permita visualizar los campos y el tipo de dato de cada campo de la tabla Clientes.

```
1  import java.sql.*;
2  public class CamposTabla {
3    public static void main(String[] args){
4      try{
5        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
6        String basededatos = "jdbc:odbc:ejemplo";
7        Connection conexion = DriverManager.getConnection(basededatos);
8        Statement sentencia = conexion.createStatement();
9      ResultSet clientes = sentencia.executeQuery("SELECT * FROM
      Clientes");
10       ResultSetMetaData campos = clientes.getMetaData();
11       int numerodecolumnas = campos.getColumnCount();
12       for (int columna=1;columna<=numerodecolumnas;columna++) {
13         String Nombre = campos.getColumnName(columna);
14         String Tipo = campos.getColumnTypeName(columna);
15         int Tamano = campos.getColumnDisplaySize(columna);
16         System.out.println(Nombre+", "+Tipo+", "+Tamano);
          }
17      clientes.close();
18      conexion.close();
19      sentencia.close();
       }
20     catch (ClassNotFoundException e) {
21     System.out.println("Clase no encontrada");
    }
22     catch (SQLException e) {
23     System.out.println(e);
       }
     }
   }
```

Línea 10: se crea un objeto *campos* de la interfaz *ResultSetMetaData*. A este objeto se le asigna los valores de los campos de la tabla por medio del objeto clientes, utilizando el método *getMetaData*.

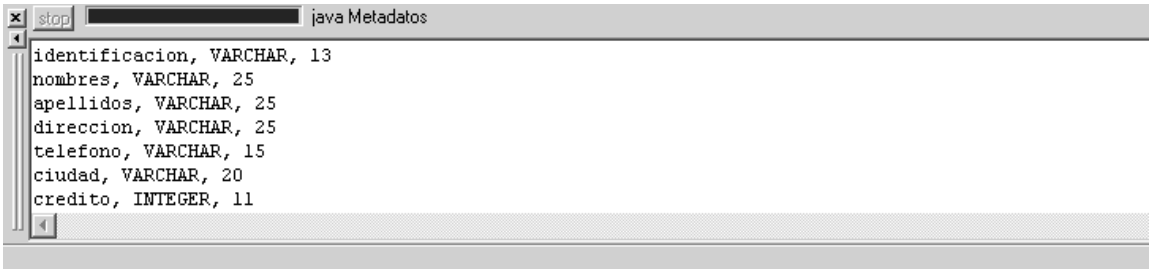
Línea 11 se crea una variable entera numerocolumnas a la cual se le asigna el número de campos que tiene la tabla utilizando el método *getColumnCount()*.

Línea 12-16: se crea un ciclo que permite recorrer cada uno de los campos de la tabla Clientes, obteniendo el nombre del campo utilizando el método getColumnName, tipo de

datos con el método *getColumnTypeName* y el tamaño del campo utilizando el método *getColumnDisplaysize*.

Al compilar y ejecutar el programa se visualizará la siguiente ventana:

Figura 10: Ventana que visualiza los campos que contiene la tabla Clientes.



Fuente: elaboración propia

Conclusiones

1. La API JDBC es muy simple de implementar, ya que solo se necesita contar con los driver de la base de datos con la que se quiere trabajar.
2. No es necesario implementar varios programas para manejar diversas bases de datos, tan solo es necesario cambiar el drive de la nueva base de datos.
3. Es necesario conocer las sentencias del lenguaje estándar SQL.
4. La interfaz entre la comunicación de la base de datos y JDBC es transparente para el usuario.
5. No es necesario ser un experto programador en Java para implementar las aplicaciones que tengan acceso a una base de datos.
6. Aunque realizar la comunicación entre la base de datos y JDBC es sencilla, si se desea realizar aplicaciones más robustas, es conveniente trabajar con motores de bases de datos que tengan mayor extensibilidad como mysql, oracle, progress, entre otras.

Referencias Bibliográficas

[1] Bobadilla Jesús, Comunicaciones y Bases de Datos con Java, Editorial Alfaomega, 2003.

[2] Deitel, H.M., Como Programar en Java, Editorial Prentice Hall, 1998.

[3] Deitel, H.M., Como Programar en Java, Editorial Prentice Hall, 2004.

[4] Lemay, Laura, Cadenhead, Rogers, Aprendiendo Java 2 en 21 días, Editorial Prentice Hall, 1999.

[5] Naughton, Patrick, Childt, Herbert, Java Manual de Referencia, Editorial McGraw Hill, 1997.

[6] Wang, Paul, Java con Programación orientada a Objetos y aplicaciones en la www, International Thomson Editores S.A.,2000.

[7] WU, Thomas, Introducción a la Programación Orientada a objetos con Java, Editorial McGraw Hill, 2001.

Infografía

http://www.desarrolloweb.com/articulos/497.php?manual=15

http://www.desarrolloweb.com/articulos/1670.php?manual=57

http://enciclopedia.us.es/index.php/Programaci%F3n_orientada_a_objetos

http://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos

http://programarenc.webcindario.com/Cplus/capitulo1.htm

http://www.monografias.com/trabajos/objetos/objetos.shtml

http://www.mysql-hispano.org/page.php?id=24

http://www.programatium.com/sql.htm

http://www.programacion.com/tutorial/jdbc/2/

http://www.ctisa.com/diccionario.htm

http://java.sun.com/j2se/1.4.2/docs/guide/jdbc/getstart/intro.html#1006158

http://www.itapizaco.edu.mx/paginas/JavaTut/froufe/parte21/cap21-3.html