UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA
FACULTAD DE INGENIERÍA (UNIDAD MEXICALI)
DOCUMENTO DEL SISTEMA DE CALIDAD

Formato para prácticas de laboratorio

CARRERA	PLAN DE ESTUDIO	CLAVE ASIGNATURA	NOMBRE DE LA ASIGNATURA
IC	2003-1	5046	Bases de Datos

PRÁCTICA No.	LABORATORIO DE	Bases de Datos	DURACIÓN (HORA)
10	NOMBRE DE LA PRÁCTICA	Conexión Mysql-Java	2

1 INTRODUCCIÓN

JDBC es un API de Java para acceder a sistemas de bases de datos, y prácticamente a cualquier tipo de dato tabular. El API JDBC consiste de un conjunto de clases e interfaces que permiten a cualquier programa Java acceder a sistemas de bases de datos de forma homogénea. En otras palabras, con el API JDBC no es necesario escribir un programa para accesar a Sybase, otro programa para accesar a Oracle, y otro programa 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. Una aplicación de Java debe tener acceso a un controlador (driver) JDBC adecuado.

2 OBJETIVO (COMPETENCIA)

Elaborar una aplicación gráfica en Java utilizando el conjunto de clases e interfaces necesarias, que permitan al alumno conocer la forma correcta de realizar transacciones en una base de datos de Mysql.

Formuló	Revisó	Aprobó	Autorizó
Ing. Alicia del R. López Aguirre	M.C. Gloria Etelbina Chavez Valenzuela		M.C. Miguel Ángel Martínez Romero
Maestro	Coordinador de la Carrera	Gestión de la Calidad	Director de la Facultad

3 FUNDAMENTO

JDBC

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.

De una manera muy simple, al usar JDBC se pueden hacer tres cosas:
1.Establecer una conexión a una fuente de datos (ej. una base de datos).
2.Mandar consultas y sentencias a la fuente de datos.
3.Procesar los resultados.

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 bases de datos.

En el caso del manejador de bases de datos MySQL, **Connector/J** es el driver JDBC oficial. En el momento de escribir esta práctica, se pueden encontrar dos versiones de este driver, la versión estable (Connector/J 2), y la versión en desarrollo (Connector/J 3). Para los ejemplos que se mostrarán a continuación se hará referencia a la versión 2 del driver, y en particular a la versión 2.0.14. Los procedimientos descritos aquí deben de ser practicamente los mismos si se utiliza alguna otra versión del driver, incluso, si se usa alguna de las versiones en desarrollo. Por supuesto, se recomienda siempre utilizar la versión estable más reciente que se pueda obtener.

Cabe señalar que actualmente JDBC es el nombre de una marca registrada, y ya no más un acrónimo; es decir, JDBC ya no debe entenderse como "Java Database Conectivity".

Herramientas necesarias

1.Un ambiente de desarrollo para Java, tal como el Java 2 SDK, el cual está disponible en java.sun.com. La versión estándar del SDK 1.4 ya incluye el API JDBC.
2.Un servidor de bases de datos MySQL al que se tenga acceso con un nombre de usuario y contraseña.
3.El driver JDBC para MySQL, Connector/J

Cargar el controlador JDBC

Para trabajar con el API JDBC se tiene que importar el paquete java.sql, tal y como se indica a continuación:

import java.sql.;*

En este paquete se definen los objetos que proporcionan toda la funcionalidad que se requiere para el acceso a bases de datos. El siguiente paso después de importar el paquete java.sql consiste en cargar el controlador JDBC, es decir un objeto Driver específico para una base de datos que define cómo se ejecutan las instrucciones para esa base de datos en particular.

3 FUNDAMENTO

Hay varias formas de hacerlo, pero la más sencilla es utilizar el método forName() de la clase Class:

Class.forName("Controlador JDBC");

para el caso particular del controlador para MySQL, Connector/J, se tiene lo siguiente:

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

Debe tenerse en cuenta que el método estático forName() definido por la clase Class genera un objeto de la clase especificada. Cualquier controlador JDBC tiene que incluir una parte de iniciación estática que se ejecuta cuando se carga la clase. En cuanto el cargador de clases carga dicha clase, se ejecuta la iniciación estática, que pasa a registrarse como un controlador JDBC en el DriverManager. Es decir, el siguiente código:

Class.forName("Controlador JDBC");

es equivalente a:

```
Class c = Class.forName("Controlador JDBC");
Driver driver = (Driver)c.newInstance();
DriverManager.registerDriver(driver);
```

Algunos controladores no crean automáticamente una instancia cuando se carga la clase. Si forName() no crea por sí solo una instancia del controlador, se tiene que hacer esto de manera explícita:

Class.forName("Controlador JDBC").newInstance();

De nuevo, para el Connector/J:

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

Establecer la conexión

Una vez registrado el controlador con el DriverManager, se debe especificar la fuente de datos a la que se desea acceder. En JDBC, una fuente de datos se especifica por medio de un URL con el prefijo de protocolo jdbc: , la sintaxis y la estructura del protocolo es la siguiente:

jdbc:{subprotocolo}:{subnombre}

El {subprotocolo} expresa el tipo de controlador, normalmente es el nombre del sistema de base de datos, como db2, oracle o mysql.

El contenido y la sintaxis de {subnombre} dependen del {subprotocolo}, pero en general indican el nombre y la ubicación de la fuente de datos.

3 FUNDAMENTO

El formato general para conectarse a MySQL es:

jdbc:mysql:///[servidor][:puerto]/[base_de_datos][?param1=valor1][param2=valor2]...

Si queremos acceder de manera local a la base de datos, el URL sería :

String url = "jdbc:mysql://localhost/Nombre_de_la_base_de_datos";

Una vez que se ha determinado el URL, se puede establecer una conexión a una base de datos. El objeto Connection es el principal objeto utilizado para proporcionar un vínculo entre las bases de datos y una aplicación Java. Connection proporciona métodos para manejar el procesamiento de transacciones, para crear objetos y ejecutar instrucciones SQL y para crear objetos para la ejecución de procedimientos almacenados.

Se puede emplear tanto el objeto Driver como el objeto DriverManager para crear un objeto Connection. Se utiliza el método connect() para el objeto Driver, y el método getConnection() para el objeto DriverManager.

El objeto Connection proporciona una conexión estática a la base de datos. Esto significa que hasta que se llame en forma explícita a su método close() para cerrar la conexión o se destruya el objeto Connection, la conexión a la base de datos permanecerá activa.

La manera más usual de establecer una conexión a una base de datos es invocando el método getConnection() de la clase DriverManager. A menudo, las bases de datos están protegidas con nombres de usuario (login) y contraseñas (password) para restringir el acceso a las mismas. El método getConnection() permite que el nombre de usuario y la contraseña se pasen también como parámetros.

```
String login = "Alicia";
String password = "reprobaralumnos";
    Connection conn = DriverManager.getConnection(url,login,password);
```

En toda aplicación de bases de datos con MySQL es indispensable poder establecer la conexión al servidor para posteriormente enviarle las consultas. Los programas en Java no son la excepción. El siguiente código nos servirá para verificar que podemos establecer una conexión a una base de datos.

```
import java.sql.*;

public class TestConnection
{
  static String bd = "UABC";
  static String login = "Alicia";
  static String password = "reprobaralumnos";
  static String url = "jdbc:mysql://localhost/"+bd;
```

3 FUNDAMENTO

```
  public static void main(String[] args) throws Exception
  {
    Connection conn = null;
   try
    {
      Class.forName("com.mysql.jdbc.Driver").newInstance();
       conn = DriverManager.getConnection(url,login,password);
      if (conn != null)
      {
        System.out.println("Conexión a base de datos "+url+" ... Ok");
        conn.close();
      }
    }
    catch(SQLException ex)
    {
      System.out.println(ex);
    }
    catch(ClassNotFoundException ex)
    {
      System.out.println(ex);
    }

  }
}
```

Creación de sentencias

Como se mencionó en la sección anterior, el objeto Connection permite establecer una conexión a una base de datos. Para ejecutar instrucciones SQL y procesar los resultados de las mismas, debemos hacer uso de un objeto Statement.

Los objetos Statement envían comandos SQL a la base de datos, que pueden ser de cualquiera de los tipos siguientes:

1.Un comando de definición de datos como CREATE TABLE o CREATE INDEX.
2.Un comando de manipulación de datos como INSERT, DELETE o UPDATE.
3.Un sentencia SELECT para consulta de datos.

Un comando de manipulación de datos devuelve un contador con el número de filas (registros) afectados, o modificados, mientras una instrucción SELECT devuelve un conjunto de registros denominado conjunto de resultados (result set). La interfaz Statement no tiene un constructor , sin embargo, podemos obtener un objeto Statement al invocar el método createStatement() de un objeto Connection.

3 FUNDAMENTO

conn = DriverManager.getConnection(url,login,pasword);
Statement stmt = conn.createStatement();

Una vez creado el objeto Statement, se puede emplear para enviar consultas a la base de datos usando los métodos execute(), executeUpdate() o executeQuery(). La elección del método depende del tipo de consulta que se va a enviar al servidor de bases de datos:

execute( )
Se usa principalmente cuando una sentencia SQL regresa varios conjuntos de resultados. Esto ocurre principalmente cuando se está haciendo uso de procedimientos almacenados.
executeUpdate( )
Este método se utiliza con instrucciones SQL de manipulación de datos tales como INSERT, DELETE o UPDATE.
ExecuteQuery( )
Se usa en las instrucciones del tipo SELECT.

Es recomendable que se cierren los objetos Connection y Statement que se hayan creado cuando ya no se necesiten. Lo que sucede es que cuando en una aplicación en Java se están usando recursos externos, como es el caso del acceso a bases de datos con el API JDBC, el recolector de basura de Java (garbage collector) no tiene manera de conocer cuál es el estado de esos recursos, y por lo tanto, no es capaz de liberarlos en el caso de que ya no sean útiles. Lo que sucede en estos casos es que pueden quedar almacenados en memoria grandes cantidades de recursos relacionados con la aplicación de bases de datos que se está ejecutando. Es por esto que se recomienda que se cierren de manera explícita los objetos Connection y Statement.

De manera similar a Connection, la interfaz Statement tiene un método close() que permite cerrar de manera explícita un objeto Statement. Al cerrar un objeto Statement se liberan los recursos que están en uso tanto en la aplicación Java como en el servidor de bases de datos.

```
Statement stmt = conn.createStatement();
    ....
stmt.close();
```

Ejecución de consultas

Cuando se ejecutan sentencias SELECT usando el método executeQuery(), se obtiene como respuesta un conjunto de resultados, que en Java es representado por un objeto ResultSet.

```
Statement stmt = conn.createStatement();
ResultSet res  = stmt.executeQuery("SELECT * FROM nombre_de_la_tabla");
```

La información del conjunto de resultados se puede obtener usando el método next() y los diversos métodos getXXX() del objeto ResultSet. El método next() permite moverse fila por fila a través del ResultSet, mientras que los diversos métodos getXXX() permiten accesar a los datos de una fila en particular.

3 FUNDAMENTO

Los métodos getXXX() toman como argumento el índice o nombre de una columna, y regresan un valor con el tipo de datos especificado en el método. Así por ejemplo, getString() regresará una cadena, getBoolean() regresará un booleano y getInt() regresará un entero. Cabe mencionar que estos métodos deben tener una correspondencia con los tipos de datos que se tienen en el ResultSet, y que son a las vez los tipos de datos provenientes de la consulta SELECT en la base de datos, sin embargo, si únicamente se desean mostrar los datos se puede usar getString() sin importar el tipo de dato de la columna.

Por otra parte, si en estos métodos se utiliza la versión que toma el índice de la columna, se debe considerar que los índices empiezan a partir de 1, y no en 0 (cero) como en los arreglos, los vectores, y algunas otras estructuras de datos de Java.

Existe un objeto ResultSetMetaData que proporciona varios métodos para obtener información sobre los datos que están dentro de un objeto ResultSet. Estos métodos permiten entre otras cosas obtener de manera dinámica el número de columnas en el conjunto de resultados, así como el nombre y el tipo de cada columna.

```
ResultSet res  = stmt.executeQuery("SELECT * FROM nombre_de_la_tabla");
ResultSetMetaData metadata = res.getMetaData();
```

Un ejemplo completo

El siguiente programa realiza la inserción de renglones en la tabla tablaPrueba de la base de datos test utilizando una interfaz gráfica para captura de datos.Este programa ejecuta la inserción (Insert) a través de una instrucción de SQL precompilada que permite ejecutar sentencias de SQL de una manera más eficiente y fácil de codificar. De una conexión se obtiene, utilizando el método preparedStatement(), el llamado PreparedStatment ó SQL precompilado. Un PreparedStatement es más eficiente ya que se prepara una vez y se puede utilizar en múltiples ocaciones. Además resulta más fácil de codificar con éste, ya que libera al programador de escribir complejas concatenaciones de Strings y de usar métodos de formateo de datos propietarios para formular sentencias del SQL. En vez, se utiliza el signo de interrogación (?) para reservar un lugar para cada parámetro de entrada y programáticamente se especifican los valores para cada lugar reservado previo a la ejecución de la sentencia de SQL.

```
import java.awt.*; import java.awt.event.*; import java.sql.*;
public class JDBCInserta extends Frame implements ActionListener{
TextField texto; Button b1; TextArea data;  Label etiqueta;
 int rows = 0;
public JDBCInserta()
{  super("JDBC Inserta"); setSize(240, 280); setLayout(new FlowLayout());
   TextField("jdbc:mysql://127.0.0.1/test? user=Ala&password=", 20);
   etiqueta = new Label("Escriba el texto a insertar:"); add(etiqueta);
   texto=new TextField("algun texto", 18); add(texto);
   b1=new Button("Inserta Datos"); b1.addActionListener(this); add(b1);
   data=new TextArea(10, 30); add(data);
   addWindowListener(new WindowAdapter(){ public void windowClosing(WindowEvent e)
      { dispose(); System.exit(0); }}); show();
}
```

3 FUNDAMENTO

```
public void insertaDatos() throws SQLException
{ String usr, pass;
   String f1, f2;
   String url="jdbc:mysql://127.0.0.1/test? user=Ala&password=";
   String text1 = texto.getText();
    PreparedStatement pstmt = null;
    try{ // inicializar y cargar Driver de Mysql.
            Class.forName("com.mysql.jdbc.Driver");
       }catch(ClassNotFoundException e){System.out.println("Could not find driver!"); System.exit(1); }
    Connection con=DriverManager.getConnection(url);//, usr, pass);
    try{ pstmt = con.prepareStatement(  "INSERT into tablaPrueba ( "+ " identif,"+ "campoTexto)"+ " values ( "+ " ","+
          "   ?) ");
         }
          catch (SQLException e) {  System.err.println(e); }
          pstmt.setString(1,text1);
         rows = 0;
         rows=pstmt.executeUpdate();
         pstmt.close();
     if( rows == 1 ) { data.append("\nRegistro Insertado"); }
}

public void actionPerformed(ActionEvent e)
{
   data.replaceRange("Field 1\t\tField 2",0,59);
   data.append("\n-----\t\t-----");
   try{ insertaDatos(); }
   catch(SQLException ex)
   { data.append("\nError inserting in database:"+rows+"inserted.");
      data.append("\n"+ex.getMessage());
   }
}

public static void main(String args[])
{   new JDBCInserta(); }

}
```

3 FUNDAMENTO

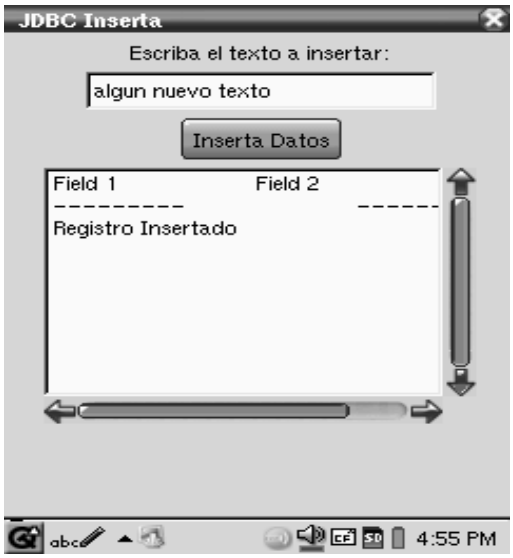

Figura del programa en ejecución

4 PROCEDIMIENTO (DESCRIPCIÓN)	
A EQUIPO NECESARIO	**MATERIAL DE APOYO**
Computadora con acceso al servidor Mysql	Práctica impresa
B DESARROLLO DE LA PRÁCTICA	

4 PROCEDIMIENTO (DESCRIPCIÓN)

Sistema de infomación ecológica para áreas protegidas (FORIS)

En el a?o 2000 la FAO elaboró nuevos mapas mundiales de los bosques y de las zonas ecológicas, los cuales definen espacialmente las estadísticas del área, que a su vez fueron producto del estudio realizado en cada país y en cada una de las regiones, proporcionando así un cuadro sinóptico de la **cubierta forestal en todo el mundo.** El mapa mundial de zonas ecológicas proporciona un medio importante para agregar la información mundial sobre los bosques u otros recursos naturales de acuerdo a su carácter ecológico. El mapa de la cubierta forestal fue desarrollado mediante la utilización de imágenes satelitares de resolución gruesa. En las evaluaciones mundiales previas, no existían los medios ni la tecnología para producir un mapa mundial basado en imágenes satelitares. Actualmente otro organismo forestal llamado FRA se dio a la tarea de dividir el mapa mundial en zonas ecológicas y recopilar cierta información sobre ellas. Por ejemplo entre otras cosas se recopilaron los datos de áreas protegidas tanto nacionales como internacionales. La infomación se encuentra en la sig. tabla

Datos internacionales y nacionales para las áreas protegidas

	POLIGONOS			PUNTOS		
Región	**Nacional**	**Internacional**	**Total**	**Nacional**	**Internacional**	**Total**
África	1 926	293	2 219	2 088	74	2 162
Asia	3 907	288	4 195	2 384	107	2 491
Europa	2 1468	1 587	23 055	19 478	1 915	21 393
Norte y Centro América	10 119	352	10 471	4 722	92	4 814
Oceanía	816	427	1 243	2 739	53	2 792
América del Sur	2 436	158	2 594	1 413	48	1 461
Antártica	0	28	28			
Otros	25	12	37	3 156	517	3 673

Nota: Se manejan dos tipos de area para cada región POLIGONOS y PUNTOS.

Elabore una aplicación gráfica en Java que permita realizar las opciones:

a) Almacenar toda la información anterior en una base de datos.
b) Modificaciones para los datos de región, superficie nacional y internacional (tanto para poligonos como para puntos).
c) Eliminar alguna región específica.
d) Opciones varias:
   1.- Total de áreas POLIGONOS protegidas tanto nacional como internacional para cada región
   2.- Nombre de la región con mayor área protegida de PUNTOS nacionalmente
   3.- Total de áreas protegidas tanto nacional e internacionalmente en todo el planeta.

C	CÁLCULOS Y REPORTE
El alumno deberá entregar impreso el código de la aplicación elaborado en esta práctica.	

5 RESULTADOS Y CONCLUSIONES

Al finalizar la práctica el alumno será capaz de ralizar mediante aplicaciones gráficas en Java conexiones y transacciones con el servidor de base de datos Mysql .

6 ANEXOS