

Tutorial Java con acceso a datos e Interfaces gráficas

Jazna Meza Hidalgo

Mayo 2010

1 OBJETIVOS

- Utilizar MySQL para manipulación de datos
- Trabajar con SQL como lenguaje de consulta
- Construir una aplicación Java capaz de acceder a una base de datos y ejecute las operaciones básicas de recuperación, actualización de registros usando interfaces gráficas estilo Windows.

2 REQUERIMIENTOS

SOFTWARE	LINK
Java Development Kit (JDK)	http://java.sun.com
Netbeans	http://www.netbeans.org
MySQL	http://dev.mysql.com
Conector MySQL	http://dev.mysql.com

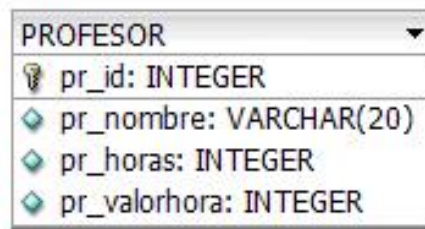
Table 1: Requerimientos

3 DESCRIPCIÓN

Considerando el modelo de datos revisado en uno de los tutoriales anteriores vamos a construir una aplicación que permita manipular los datos de la tabla y obtener los resultados de las consultas. Se recuerda que el modelo (que cuenta con una sola tabla) es el que se describe en la Figura 1 y Figura 2.

4 ACTIVIDAD 1 - CREANDO EL PROYECTO NETBEANS

Una vez que se ha creado la base de datos se va a revisar la forma de conectar la base de datos a un proyecto NetBeans de forma de poder construir aplicaciones que puedan



PROFESOR
pr_id: INTEGER
pr_nombre: VARCHAR(20)
pr_horas: INTEGER
pr_valorhora: INTEGER

Figure 1: Tabla que utiliza la aplicación

PROFESOR						
Representa los datos de los profesores y sus horas de docencia mensuales						
ColumnName	DataType	PrimaryKey	NotNull	Flags	Default Value	Comment
pr_id	INTEGER	PK	NN	UNSIGNED		ID del profesor
pr_nombre	VARCHAR(20)					Nombre del profesor
pr_horas	INTEGER			UNSIGNED		Número de horas de docencia al mes
pr_valorhora	INTEGER			UNSIGNED		Valor de hora de docencia
IndexName	IndexType		Columns			
PRIMARY	PRIMARY		pr_id			

Figure 2: Detalle de los atributos de la tabla

trabajar sobre la base de datos. En esta actividad se debe crear un proyecto NetBeans del tipo Aplicación Desktop Java.

5 ACTIVIDAD 2 - MODIFICANDO VENTANA INICIAL

Modificar la ventana propuesta hasta obtener una apariencia como la que se indica en la Figura 3. El detalle de los elementos de la ventana son los que se encuentran en la Figura 4:

6 ACTIVIDAD 3 - AGREGANDO PACKAGES

Agregar al proyecto los paquetes relacionados con la capa de servicio y la capa de negocio, de acuerdo a lo que se revisó en el tutorial de acceso a datos.

7 ACTIVIDAD 4 - AGREGANDO NUEVAS VENTANAS

Vamos a agregar las ventana que aparece en las Figura 5, la clase DEBE llamarse *FEdicion*. A través de la Paleta de elementos que se muestra en la Figura 6 será posible agregar los componentes de la ventana. Ahora vamos a revisar el código para implementar nuestra ventana:

- Aspecto del constructor de acuerdo a lo que aparece en la Figura 8.
- Programación del evento del botón Guardar (ver Figura 9)

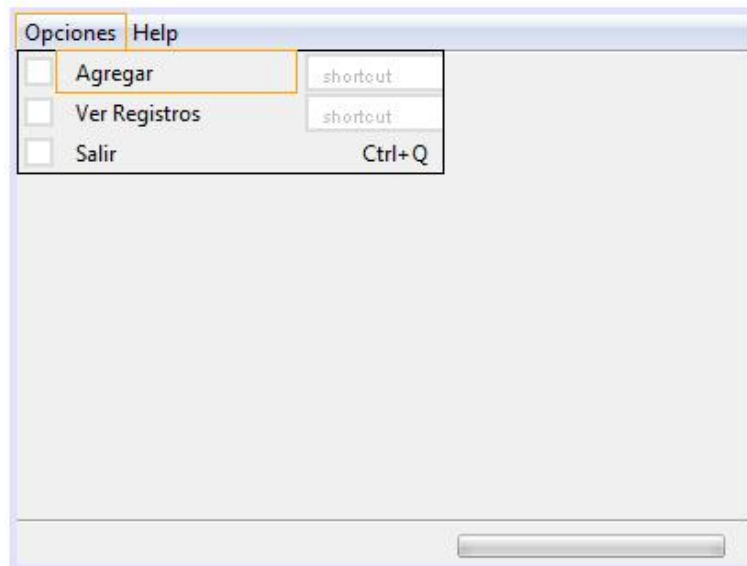


Figure 3: Apariencia de la ventana

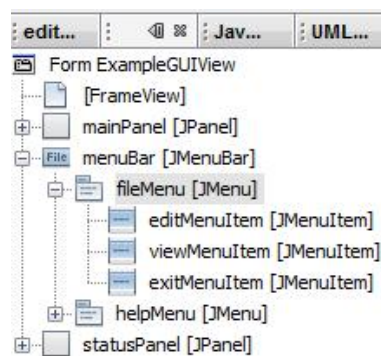


Figure 4: Elementos de la ventana



Figure 5: Frame para agregar datos a la tabla

- Programación del evento del botón Limpiar (ver Figura 10)

8 ACTIVIDAD 5 - VINCULANDO LA NUEVA VENTANA

Vamos a unir la ventana que se ha creado a la aplicación existente, para lo cual vamos a modificar el código de la ventana principal de la aplicación y se agrega el código que aparece en la Figura 10. Luego vamos a programar el evento asociado a la opción de menú para poder abrir la ventana y agregamos el código que aparece en la Figura 11. Ahora probemos el proyecto para verificar que se agreguen nuevos registros a la tabla a través de nuestra nueva interfaz gráfica.

9 ACTIVIDAD 6 - CREANDO Y VINCULANDO LA VENTANA PARA VER LA TABLA

Ahora vamos a crear un nuevo *JFrame* que tenga el aspecto de la Figura 12, la nueva ventana DEBE llamarse *FVerTabla*. Una vez que se haya diseñado la ventana se van a programar los eventos para poder mostrar los datos de la tabla en el *JTable* que se ha agregado al *Frame* y se tienen las siguientes acciones:

- Agregar al final de la clase *FVerTabla* el código que aparece en la Figura 13.
- Agregar atributos y modificar el constructor (ver Figura 14)
- Programación del evento para seleccionar un registro de la tabla (ver Figura 15)

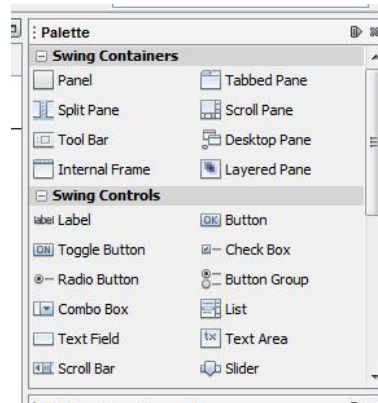


Figure 6: Paleta de componentes que pueden ser agregadas al Frame

```

23 public class FEdition extends javax.swing.JFrame {
24
25     private Profesor p;
26     /** Creates new form FEdition */
27     public FEdition() throws ClassNotFoundException, java.langInstantiationException,
28                                     java.lang.IllegalAccessException,
29                                     java.sql.SQLException,
30                                     MyError {
31         initComponents();
32         p = new Profesor();
33         p.setId(p.generarCodigo());
34         this.txt_id.setText(String.valueOf(p.getId()));
35     }

```

Figure 7: Atributos y constructor de la clase FEdition

```

178 private void btGuardarActionPerformed(java.awt.event.ActionEvent evt) {
179     String nombre;
180     int horas, valor_hora;
181
182     /* Obtiene los datos desde la interfaz */
183     nombre = this.txt_nombre.getText();
184     horas = Integer.parseInt(this.txt_horas.getText());
185     valor_hora = Integer.parseInt(this.txt_valor.getText());
186
187     /* Setea los valores del objeto */
188     p.setNombre(nombre); p.setHoras(horas); p.setValorHora(valor_hora);
189     try{
190         p.grabar();
191         JOptionPane.showMessageDialog(null, "Profesor ha sido agregado", "Registro profesor",
192                                     JOptionPane.INFORMATION_MESSAGE);
193         /* Setea el contenido de la ventana */
194         p.setId(p.generarCodigo());
195         this.txt_id.setText(String.valueOf(p.getId()));
196         this.txt_nombre.setText("");
197         this.txt_horas.setText("0"); this.txt_valor.setText("0");
198     }
199     catch (java.lang.ClassNotFoundException e) { JOptionPane.showMessageDialog(null, "Descripción : " + e.getMessage()); }
200     catch (java.lang.IllegalAccessException e) { JOptionPane.showMessageDialog(null, "Descripción : " + e.getMessage()); }
201     catch (java.langInstantiationException e) { JOptionPane.showMessageDialog(null, "Descripción : " + e.getMessage()); }
202     catch (java.sql.SQLException e) { JOptionPane.showMessageDialog(null, "Descripción : " + e.getMessage()); }
203     catch (MyError e) { JOptionPane.showMessageDialog(null, "Descripción : " + e.getMessage()); }
204 }

```

Figure 8: Programación del evento sobre el botón Guardar

```

206 private void btLimpiarActionPerformed(java.awt.event.ActionEvent evt) {
207     /* Limpia el contenido de los textfields de la ventana */
208     this.txt_id.setText("SIN ASIGNAR");
209     this.txt_nombre.setText("");
210     this.txt_horas.setText("0"); this.txt_valor.setText("0");
211 }

```

Figure 9: Programación del evento sobre el botón Limpiar

```

27 public class ExampleGUIView extends FrameView {
28
29     private FEdition ventana;

```

Figure 10: Cambios en la clase de la ventana principal

- Programación del evento del botón Guardar (ver Figura 16)

Ahora vamos a vincular la tabla a nuestra ventana principal y se tiene el código asociado a la Figura 17

```

221 private void editMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
222     try{
223         if (ventana == null) ventana = new FEdition();
224         ventana.setLocationRelativeTo(null);
225         ventana.setVisible(true);
226     }
227     catch (java.lang.ClassNotFoundException e){ JOptionPane.showMessageDialog(null, "Descripción : " + e.getMessage()); }
228     catch (java.lang.IllegalAccessException e){ JOptionPane.showMessageDialog(null, "Descripción : " + e.getMessage()); }
229     catch (java.langInstantiationException e){ JOptionPane.showMessageDialog(null, "Descripción : " + e.getMessage()); }
230     catch (java.sql.SQLException e){ JOptionPane.showMessageDialog(null, "Descripción : " + e.getMessage()); }
231     catch (MyError e){ JOptionPane.showMessageDialog(null, "Descripción : " + e.getMessage()); }
232 }

```

Figure 11: Programación del evento de la opción de menú

Title 1	Title 2	Title 3	Title 4

Antecedentes de Profesor

ID

NOMBRE

HORAS

VALOR HORA

Guardar

Figure 12: Apariencia de la ventana para ver el contenido de la tabla

```

303 class MiModelo extends DefaultTableModel
304 {
305     public MiModelo(Object[][] data, Object[] columnNames) {
306         super(data, columnNames);
307     }
308     @Override
309     public boolean isCellEditable (int row, int column)
310     {
311         // Aquí devolvemos true o false según queramos que una celda
312         // identificada por fila,columna (row,column), sea o no editable
313         return false;
314     }
315 }

```

Figure 13: Agregar nueva clase para manejar el modelo del JTable

```

27 public class FVerTabla extends javax.swing.JFrame {
28     /* Array de String's con los titulos de las columnas */
29     private String[] columnNames = {"ID", "NOMBRE", "HORAS", "VALOR HORA"};
30     private MiModelo dtm;
31     private Profesor p;
32
33     /** Creates new form FVerTabla */
34     public FVerTabla() {
35         initComponents();
36         cargarTabla();
37     }

```

Figure 14: Atributos y constructor de la clase FEdicion

```

195 private void cargarTabla() {
196     ArrayList r;
197     Iterator iterador;
198     Profesor un_profesor;
199     Object[][] data;
200     int index = 0;
201     try{
202         p = new Profesor();
203         r = p.cargar();
204         iterador = r.iterator();
205         /* Pasa los datos a la tabla */
206         data = new Object[r.size()][4];
207         while (iterador.hasNext()){
208             un_profesor = (Profesor) iterador.next();
209             data[index][0] = un_profesor.getId();
210             data[index][1] = un_profesor.getNombre();
211             data[index][2] = un_profesor.getHoras();
212             data[index][3] = un_profesor.getValorHora();
213             index++;
214         }
215
216         /* Se crea el Modelo de la tabla con los datos anteriores */
217         dtm = new MiModelo(data, columnNames);
218         this.jTableProfesor.setModel(dtm);
219     }
220     catch(java.lang.ClassNotFoundException e){ JOptionPane.showMessageDialog(null, "Descripción : " + e.getMessage()); }
221     catch(java.lang.IllegalAccessException e){ JOptionPane.showMessageDialog(null, "Descripción : " + e.getMessage()); }
222     catch(java.langInstantiationException e){ JOptionPane.showMessageDialog(null, "Descripción : " + e.getMessage()); }
223     catch(java.sql.SQLException e){ JOptionPane.showMessageDialog(null, "Descripción : " + e.getMessage()); }
224     catch(MyError e){ JOptionPane.showMessageDialog(null, "Descripción : " + e.getMessage()); }
225 }

```

Figure 15: Método para cargar los datos en el JTable


```

227 private void jTableProfesorMouseClicked(java.awt.event.MouseEvent evt) {
228     /* Obtiene el número de fila que ha sido seleccionada */
229     int fila = this.jTableProfesor.rowAtPoint(evt.getPoint());
230     int columna = this.jTableProfesor.columnAtPoint(evt.getPoint());
231     /* Muestra los datos en la ventana de edición si corresponde */
232     if ((fila > -1) && (columna > -1)){
233         this.txt_id.setText(String.valueOf((Integer) this.dtm.getValueAt(fila,0)));
234         this.txt_nombre.setText((String) this.dtm.getValueAt(fila,1));
235         this.txt_horas.setText(String.valueOf((Integer) this.dtm.getValueAt(fila,2)));
236         this.txt_valor.setText(String.valueOf((Integer) this.dtm.getValueAt(fila,3)));
237
238         /* Activamos el botón para guardar los datos */
239         this.btGuardar.setEnabled(true);
240     }
241 }

```

Figure 16: Programación del evento para hacer clic sobre un registro de la tabla

```

243 private void btGuardarActionPerformed(java.awt.event.ActionEvent evt) {
244
245     int id, horas, valorHora;
246     String nombre;
247
248     /* Obtiene los datos desde la ventana de edición */
249     id = Integer.parseInt(this.txt_id.getText());
250     nombre = this.txt_nombre.getText();
251     horas = Integer.parseInt(this.txt_horas.getText());
252     valorHora = Integer.parseInt(this.txt_valor.getText());
253
254     /* Crea el objeto con los datos obtenidos */
255     p = new Profesor(id, nombre, horas, valorHora);
256     try{
257         p.grabar();
258         JOptionPane.showMessageDialog(null, "Datos han sido actualizados", "Registro profesor",
259                                     JOptionPane.INFORMATION_MESSAGE);
260
261         cargarTabla();
262         /* Setea los valores del sector de edición */
263         this.txt_id.setText("VALOR ID");
264         this.txt_nombre.setText("");
265         this.txt_horas.setText("0");
266         this.txt_valor.setText("0");
267         /* Desactivamos el botón para guardar los datos */
268         this.btGuardar.setEnabled(false);
269     }
270     catch(java.lang.ClassNotFoundException e){ JOptionPane.showMessageDialog(null, "Descripción : " + e.getMessage()); }
271     catch(java.lang.IllegalAccessException e){ JOptionPane.showMessageDialog(null, "Descripción : " + e.getMessage()); }
272     catch(java.langInstantiationException e){ JOptionPane.showMessageDialog(null, "Descripción : " + e.getMessage()); }
273     catch(java.sql.SQLException e){ JOptionPane.showMessageDialog(null, "Descripción : " + e.getMessage()); }
274     catch(MyError e){ JOptionPane.showMessageDialog(null, "Descripción : " + e.getMessage()); }
275 }

```

Figure 17: Programación del evento para guardar los datos

```

234 private void viewMenuItemActionPerformed(java.awt.event.ActionEvent evt) {
235     /* Muestra la ventana que visualiza los registros */
236     if (verRegistros == null) verRegistros = new FVerTabla();
237     verRegistros.setLocationRelativeTo(null);
238     verRegistros.setVisible(true);
239
240 }

```

Figure 18: Vinculación de la ventana en el evento del menú