“Instituto Tecnológico De Ciudad Madero”

Materia: Ingeniería de Software

Tarea 5

Diseño de un manual de técnico para el sistema.

Nombre:

Rubí E. Moya Hernández

Brandon A. Ramírez Velázquez

Yamary Sánchez Ramos

N° de control:

17071679

17070736

17071310

Índice

[Introducción 3](#_Toc43147782)

[Base de datos 3](#_Toc43147783)

[Conexión de la aplicación con la base de datos 4](#_Toc43147784)

[Procedimientos almacenados 4](#_Toc43147785)

[Añadir equipo 5](#_Toc43147786)

[Ingresar a la aplicación 5](#_Toc43147787)

[Registrar mantenimiento 6](#_Toc43147788)

[Interfaz gráfica 7](#_Toc43147789)

[Login 7](#_Toc43147790)

[Menú 7](#_Toc43147791)

[Agregar computadora 8](#_Toc43147792)

[Computadoras 8](#_Toc43147793)

[Mantenimiento 9](#_Toc43147794)

[Reporte 10](#_Toc43147795)

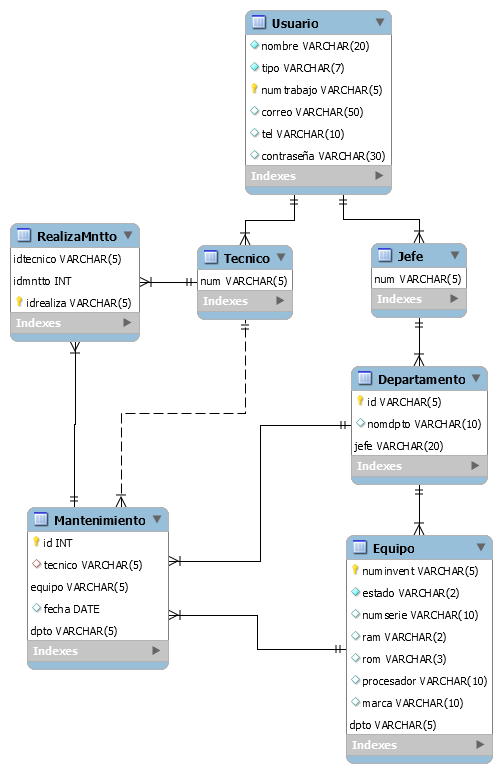
# Introducción

Este manual tiene como objetivo explicar todas las capacidades y limitantes técnicas, mostrar la estructura de datos del sistema y explicar su metodología de programación, explicando cada clase y métodos usados para el desarrollo de este sistema de control de equipos.

# Base de datos

Para el funcionamiento del sistema, se tiene que tener en ejecución el servicio de servidor de base de datos, esto se hace ejecutando MySQL Server con la base de datos montada.

Se incluye un script con las tablas de la base de datos, únicamente se tiene que ejecutar y poner en uso la base de datos para comenzar con el funcionamiento de la aplicación.

En este diagrama entidad-relación se muestra las relaciones que tienen nuestras tablas, donde el usuario es padre de técnico y jefe, último el cual hace referencia a jefe de departamento.

La tabla funciona de tal manera que cuando se registra un Equipo, este pertenezca a un Departamento y este a su vez, tiene que tener asignado un Jefe de Departamento.

Los mantenimientos son ajenos y pueden crearse sin requerir alguna relación, su única relación sería con el Técnico que es el encargado de hacer un mantenimiento a los equipos, el cual tendrá que disponer de los datos informativos que se incluyen en un reporte de mantenimiento ya sea correcto o preventivo.

Aplicación

## Conexión de la aplicación con la base de datos

Este programa está desarrollado en Java, conjunto con SQL, entonces para poder desarrollar todos los requerimientos solicitados, la primer instancia es tener una conexión con nuestra base de datos, para esto es la clase **conexión.**

public class conexion {

Connection conect = null;

protected Connection Conexion() {

try {

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

conect = (Connection) DriverManager.getConnection("jdbc:mysql://localhost:3306/proyecto", "root", "password");

} catch (ClassNotFoundException | SQLException e) {

JOptionPane.showMessageDialog(null, "Conexión fallida" + e.getMessage());

System.out.println(e.getStackTrace());

}

return conect;

}

}

En esta clase hacemos uso de la librería MySQL JDBC Connector 5.1.48. Lo primero que se tiene es un objeto **conect**, usando los paquetes de la librería, es de tipo Connection.

Una vez hecho esto, le asignados al objeto **conect** el retorno de la función *getConnection* provisto por la clase DriverManager, enviando tres parámetros, primero la dirección donde nos vamos a conectar, segundo el usuario con sus permisos de SQL y tercero la contraseña de dicho usuario.

En caso de un error, se enviará a la sentencia de Catch y se imprimirá usando getStackTrace para desplegar cuál fue el error de la conexión.

## Procedimientos almacenados

Para aumentar la seguridad del sistema, se hace uso de transacciones a través de procedimientos almacenados, mismos que serán llamados a través de una instancia de la clase **conexión**.

Estos se encuentran en la clase **callStoredProcedures,** todos ellos con un funcionamiento único para cada función del sistema, mismos que solo envían datos a la base de datos que ya contiene los procesos almacenados.

public static void main(String[] args) throws SQLException {

CallStoredProcedures llamar = new CallStoredProcedures();

}

Esta es la principal estructura de la clase, se crea un objeto de la misma y con esta instancia, se hacen las llamadas a los métodos de la clase.

## Añadir equipo

En este método se reciben los datos desde la interfaz gráfica y se envían al proceso almacenado de MySQL. Se crea una instancia de la clase Connection y se crea una instancia de tipo CallableStatement para llamar el procedimiento **añadir\_equipo** enviando los argumentos recibidos en el método.

protected void añadirEquipo(String numinvent, String estado, String numserie, String ram, String rom, String procesador, String marca, String dpto) throws SQLException {

conexion conex = new conexion();

try (Connection conect = conex.Conexion()) {

CallableStatement sp = conect.prepareCall("{CALL añadir\_equipo(?,?,?,?,?,?,?,?,?)}");

sp.setString("inventNumero", numinvent);

sp.setString("estado", estado);

sp.setString("numeroSerie", numserie);

sp.setString("memRam", ram);

sp.setString("memRom", rom);

sp.setString("procesador", procesador);

sp.setString("marca", marca);

sp.setString("depto", dpto);

sp.registerOutParameter("mensaje", Types.VARCHAR);

sp.execute();

String resul = sp.getString("mensaje");

conect.close();

JOptionPane.showMessageDialog(null, resul);

} catch (Exception e) {

JOptionPane.showMessageDialog(null, "Error en la consulta es : " + e.getMessage());

}

}

## Ingresar a la aplicación

Para ingresar a la aplicación también se tiene un procedimiento, que recibe los datos obtenidos en la primer interfaz gráfica, la de inicio de sesión.

protected String ingresar(String correo, String contraseña) throws SQLException {

conexion conex = new conexion();

String resul = null;

try (Connection conect = conex.Conexion()) {

CallableStatement sp = conect.prepareCall("{CALL ingresar(?,?,?)}");

sp.setString("email", correo);

sp.setString("password", contraseña);

sp.registerOutParameter("mensaje", Types.VARCHAR);

sp.execute();

resul = sp.getString("mensaje");

conect.close();

} catch (Exception e) {

JOptionPane.showMessageDialog(null, "Error en la consulta es : " + e.getMessage());

}

return resul;

}

Al final del método regresa una cadena a través de la variable resul, esta es la encargada de regresar un mensaje de confirmación si la conexión fue exitosa o fracasó.

## Registrar mantenimiento

protected void mostrarequipos(String equipo,String fecha,String depto) throws SQLException{

conexion conex = new conexion();

ResultSet rs = null;

try (Connection conect = conex.Conexion()) {

CallableStatement sp = conect.prepareCall("{CALL mostrarequipos(?,?,?,?)}");

sp.setString("eq", equipo);

sp.setString("fech", fecha);

sp.setString("depto", depto);

sp.registerOutParameter("mensaje", Types.VARCHAR);

sp.execute();

String resul = sp.getString("mensaje");

conect.close();

} catch (Exception e) {

JOptionPane.showMessageDialog(null, "Error en la consulta es : " + e.getMessage());

}

}

# Interfaz gráfica

## Login

Esta ventana cuenta con un Listener para el botón de **Ingresar** una vez colocadas las credenciales en los campos de texto de usuario y contraseña.

private void btnIngresarActionPerformed(java.awt.event.ActionEvent evt) {

CallStoredProcedures llamar = new CallStoredProcedures();

String pass = new String(pssContraseña.getPassword());

if((txtUsuario.getText()!=null)&&(pass!=null)){

try {

String var=llamar.ingresar(txtUsuario.getText(), pass);

System.out.println(var);

var.trim();

if(var.equals("true")){

Menu y =new Menu();

y.setVisible(true);

this.dispose();

this.setVisible(false);

}

} catch (SQLException ex) {

Logger.getLogger(AgregarComputadora.class.getName()).log(Level.SEVERE, null, ex);

}

}

txtUsuario.setText("");

pssContraseña.setText("");

}

El método cuenta con una condición que revisa si ambos campos contienen datos, en caso contrario rechaza la solicitud de ingreso y muestra una ventana solicitando que coloque datos para poder ingresar.

Una vez hecho esto, envía las variables a una instancia de callStoredProcedures, llamando al método ingresar para que se revise en la base de datos si se encuentra el usuario y la contraseña coincide. Este método regresa un valor booleano, es decir que si muestra un true pasa a la siguiente ventana, en caso contrario muestra error y se solicita ingresar datos válidos. Al terminar ambos campos se reestablecen a blanco para evitar posibles filtraciones de datos.

## Menú

Aquí se muestra una ventana con 4 botones que corresponden a las 4 funciones del sistema, todos cuentan con un Listener que redirige a la ventana correspondiente de cada botón.

## Agregar computadora

Cuenta con un listener en el botón de **registrar** que permite al sistema detectar si los campos de Área y Estado se encuentran vacíos, esto para evitar que el usuario registre una computadora sin asignarle un área.

if((cbArea.getSelectedIndex()!=0)&&(cbEstado.getSelectedIndex()!=0)){

try {

llamar.añadirEquipo(txtInventario.getText(),String.valueOf(cbEstado.getSelectedItem()), txtSerie.getText(),txtRAM.getText(), txtROM.getText(),txtProcesador.getText(), txtMarca.getText(), String.valueOf(cbArea.getSelectedIndex()) );

} catch (SQLException ex) {

Logger.getLogger(AgregarComputadora.class.getName()).log(Level.SEVERE, null, ex);

}

}

Una vez corroborado que contiene los datos correspondientes, se envían todos los parámetros obtenidos al procedimiento **añadirEquipo** a través de la instancia de la clase callStoredProcedures.

## Computadoras

Ya que este método solo muestra datos, se utiliza una instancia directa de la clase conexión para desplegar las computadoras de un departamento en específico.

public ArrayList<Equipo> equiposList(){

ArrayList<Equipo> equiposList = new ArrayList<>();

try {

conexion conex = new conexion();

Connection conect = conex.Conexion();

String q1 = "SELECT \* FROM equipo WHERE dpto="+ String.valueOf(jComboBox1.getSelectedIndex()).trim();

Statement st = conect.createStatement();

ResultSet rs = st.executeQuery(q1);

Equipo equipos;

while(rs.next()){

equipos = new Equipo(rs.getString("numinvent"), rs.getString("estado"), rs.getString("numserie"), rs.getString("ram"), rs.getString("rom"), rs.getString("procesador"),rs.getString("marca"), rs.getString("dpto"));

equiposList.add(equipos);

}

} catch (SQLException e) {

JOptionPane.showMessageDialog(null, "Error en la consulta es : " + e.getMessage());

}

return equiposList;

}

Este método se encarga de retornar una lista, obteniendo una consulta de SQL que regrese todas las computadoras de cierto departamento, guardando esta consulta en la variables **rs**, entonces se utiliza un ciclo del tamaño de filas que tenga la variable **rs** para que cada fila regrese la columna correspondiente y se guarde en un objeto de tipo Equipo, acabando el ciclo se guarda en **listaEquipos** que es un ArrayList de tipo Equipo, que es la que se regresa de la llamada del método.

El botón Mostrar en la interfaz, tiene un accionador que al ser pulsado manda a llamar al método show\_user.

public void show\_user (){

ArrayList <Equipo> list = equiposList();

DefaultTableModel model = (DefaultTableModel) TbComputadoras.getModel();

model.setRowCount(0);

Object[] row = new Object[7];

for (int i = 0; i < list.size(); i++) {

Object object = row[i];

row[0] = list.get(i).getNuminvent();

row[1] = list.get(i).getEstado();

row[2] = list.get(i).getNumserie();

row[3] = list.get(i).getRam();

row[4] = list.get(i).getRom();

row[5] = list.get(i).getProcesador();

row[6] = list.get(i).getMarca();

model.addRow(row);

}

}

Este es el que se encarga de conseguir la lista de equipos a través del método **equiposList** y guardarla en un modelo de tabla, que servirá como contenedor para la tabla creada en la interfaz y será llenada con un ciclo for en cada fila de la tabla.

## Mantenimiento

En esta interfaz se utilizó la librería jcalendar y jgoodie-looks para desplegar un calendario sin tener que utilizar el integrado en la clase AWT de Swing, para tener mayor control sobre este.

El calendario sirve para elegir una fecha y el componente solo regresa el String en formato de la fecha colocada.

El botón desplegable del departamento, muestra los departamentos disponibles y hace una consulta que regresa una lista de las computadoras de esa área, esto para que el botón desplegable de computadores muestre las computadores disponibles en el área seleccionada.

private void cbAreaActionPerformed(java.awt.event.ActionEvent evt) {

try {

jComboBox1.removeAllItems();

conexion conex = new conexion();

Connection conect = conex.Conexion();

String q1 = "select numinvent from equipo where "+String.valueOf(cbArea.getSelectedIndex()).trim()+"=dpto";

Statement st = conect.createStatement();

ResultSet rs = st.executeQuery(q1);

while(rs.next()){

jComboBox1.addItem(rs.getString("numinvent"));

}

} catch (SQLException e) {

//JOptionPane.showMessageDialog(null, "Error en la consulta es : " + e.getMessage());

}

}

El botón **Guardar** tiene un listener que reúne los datos y cambia el formato obtenido del calendario, para que sea compatible con el utilizado en la base de datos propia, haciendo el envío de estos a través del procedimiento *mostrarEquipos* por la instancia de llamar en callStoredProcedures.

private void btnGuardarActionPerformed(java.awt.event.ActionEvent evt) {

CallStoredProcedures llamar = new CallStoredProcedures();

if((cbArea.getSelectedIndex()!=0)&&(jdFecha.getDate()!=null)){

try {

SimpleDateFormat dcn = new SimpleDateFormat("yyyy-MM-dd");

String date = dcn.format(jdFecha.getDate());

llamar.mostrarequipos(jComboBox1.getSelectedItem().toString().trim(), date, String.valueOf(cbArea.getSelectedIndex()).trim());

} catch (SQLException ex) {

Logger.getLogger(AgregarComputadora.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

## Reporte

En este interfaz se genera un reporte a partir de la información generada según el número de equipos con los que contemos, colocamos dicha información en la caja de texto más grande de la interfaz y buscamos una dirección en nuestro ordenador para guardar nuestro reporte, esto es posible gracias a un botón que se llama Dirección que al pulsarlo nos abre el explorador de arhivos de nuestra PC, después de elgir la ubicación, se escribe automáticamente la ruta sobre la caja de texto situada a la izquierda del botón antes mencionado.

Finalmente damos click en generar reporte y automáticamente se generara un PDF Con la información requerida.

El reporte se genera gracias al siguiente código:

private void btnReporteActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_btnReporteActionPerformed

// TODO add your handling code here:

String d= txtdir.getText();

String tex=txtacont.getText();

try{

FileOutputStream archivo= new FileOutputStream(d+ ".pdf");

Document doc ;

doc = new Document();

PdfWriter.getInstance(doc, archivo);

doc.open();

doc.add(new Paragraph(tex));

doc.close();

JOptionPane.showMessageDialog(null, "Documento creado con exito.");

}catch (Exception e){

JOptionPane.showMessageDialog(null, "error"+ e);

}

}//GEN-LAST:event\_btnReporteActionPerformed

Primero creamos una variable de tipo cadena llamada **d** que manda a llamar el texto ubicado en la caja de texto superior, ahí encontraremos la dirección que hemos seleccionado previamente, después crearemos otra variable de tipo cadena llamada **tex** que copia lo que llevará por dentro el reporte.

Creamos un objeto de la clase FileOutputStream, la cual tiene métodos los cuales ayudan a manejar el flujo de salida, y uno de la clase Document. Con el objeto doc mandamos a abrir el documento y agregar el texto con los metpdps open() y add(), y al final usaremos el método close() para cerrarlo. Si se creó exitosamente nos avisara con un mensaje en un cuadro de texto y así se almacenara el pdf creado en la dirección seleccionada.