

UT2.Manejo de Conectores.

Alberto Colmenar Casas

2. **Práctica 2.1 Conexión a MySQL desde Java:** nos conectamos a una BD de MySQL (creada previamente) con el driver de JDBC y vemos su contenido por medio del IDE.
3. **Práctica 2.2 Panel de control de usuario:** login que comprueba si ese usuario y contraseña están en nuestra base de datos.
4. **Práctica 2.2 Actualizar contraseña de usuario:** update de la contraseña de la BD dado un usuario y contraseña, usando PreparedStatement para poder crear la query parametrizada.
5. **Práctica 2.2 Validar contraseña:** validamos una contraseña con los distintos parámetros descritos en la práctica usando principalmente el método matches() y expresiones regulares.
6. **Práctica 2.3 Consulta y descripción de datos. Base de datos:** la clase DatabaseMetaData inicializada con getMetaData() nos aporta los metadatos de la BD conectada anteriormente.
7. **Práctica 2.3 Consulta y descripción de datos. Tablas:** el usuario elige que tabla quiere ver con un JOptionPane y con DatabaseMetaData obtenemos la información de las tablas.
8. **Práctica 2.3 Consulta y descripción de datos. Columnas:** con el DatabaseMetaData pedimos esta vez la información de las columnas de una tabla y con el resultSet las mostramos.
9. **Práctica 2.3 Listado de vuelos:** el método devuelve una DefaultTableModel que la construimos gracias al resultSet con la información de la tabla obtenida ejecutando una query con Statement.
10. **Práctica 2.3 Listado de pasajeros de un vuelo:** obtenemos los códigos de los vuelos con una sentencia a la BD para añadirlos al ComboBox al crear la ventana y que el usuario pueda seleccionar cualquiera.
11. **Práctica 2.4 Conexión a Oracle desde Java:** por consola escogemos a qué BD quieres conectarte. En este caso en la de Oracle cargamos su driver y nos conectamos a la BD que fue creada anteriormente.

Práctica 2.1 Conexión a MySQL desde Java

```
ColmenarAlbertoConectarConMysql.java
Source History
10  /*
11  public class ColmenarAlbertoConectarConMysql {
12
13
14      public static Connection conexion;
15
16      public void getConexion() {
17          try {
18              Class.forName("com.mysql.cj.jdbc.Driver");
19              conexion = (Connection) DriverManager.getConnection("jdbc:mysql://localhost/gestionusuarios2", "root", "");
20              System.out.println("Conexion realizada");
21          } catch (ClassNotFoundException ex) {
22              System.err.println("Error " + ex.getMessage());
23          } catch (SQLException ex) {
24              System.err.println("Error de conexión " + ex.getMessage());
25          }
26      }
27
28      public static void main(String[] args) {
29          ColmenarAlbertoConectarConMysql prueba = new ColmenarAlbertoConectarConMysql();
30          prueba.getConexion();
31      }
32  }
```

Usamos el driver JDBC y la clase DriverManager para inicializar un objeto de la clase Connection. Que es el que usaremos para comunicarnos con la BD, este caso de MySQL.

Se ve la base de datos de gestionusuarios2 que tiene una tabla usuarios con un nombre y una contraseña, la utilizaremos en la siguiente práctica.

The screenshot shows an IDE with several windows. On the left, a 'Projects' pane shows a tree structure with 'MySQL Server at localhost:3306 [root]', 'Java DB', 'Drivers', 'jdbc:derby://localhost:1527/sample [app on APP]', and 'gestionusuarios2'. The 'gestionusuarios2' database is expanded, showing 'Tables' with a sub-entry 'usuarios'. In the center, a 'Services' pane shows a 'Connection' dropdown set to 'jdbc:mysql://localhost:3306/gestionusuarios2?zeroDateTimeBehavior=convertToNull [root on Default schema]'. Below this, a SQL editor shows the query 'SELECT * FROM usuarios LIMIT 100;'. On the right, a 'SQL 1 [jdbc:mysql://localhost:33...]' window displays the query results in a table.

#	nombre	contraseña
1	qwe	qwe
2	qwe	qwer
3	Alberto	AlbertoNueva
4	Roberto	Roberto55

```
Output - ColmenarAlbertoConectarConMysql (run) × Notifications
run:
Conexion realizada
BUILD SUCCESSFUL (total time: 0 seconds)
```

Práctica 2.2 Panel de control de usuario

Los distintos mensajes que puede dar.

```
ColmenarAlbertoMVCgestionUsuarios.java X VentanaVerificar.java X
Source History
33 public boolean buscarUsuario(String nombre, String contrasena) {
34     boolean encontrado = false;
35     if (!nombre.isEmpty() || contrasena.isEmpty()) {
36         try {
37             Statement stmt = (Statement) conexion.createStatement();
38             ResultSet resultado = stmt.executeQuery("SELECT * FROM usuarios WHERE nombre=" +
39             nombre + " and contraseña=" + contrasena + "");
40             if (resultado.next()) {
41                 encontrado = true;
42             }
43             resultado.close();
44             stmt.close();
45         } catch (SQLException ex) {
46             System.err.println(ex.getMessage());
47         }
48     }
49     return encontrado;
50 }
```

```
ColmenarAlbertoMVCgestionUsuarios.java X VentanaVerificar.java X
Source Design History
171 private void botonAceptarActionPerformed(java.awt.event.ActionEvent evt) {
172     if (camposRellenados()) {
173         String nombre = cajaUsuario.getText();
174         String contrasena = String.valueOf(cajaContrasena.getPassword());
175         if (mvcGestionUsuarios.buscarUsuario(nombre, contrasena)) {
176             JOptionPane.showMessageDialog(this, "Conexión correcta, bienvenido");
177         } else {
178             JOptionPane.showMessageDialog(this, "Usuario o contraseña errónea");
179         }
180     } else {
181         JOptionPane.showMessageDialog(this, "Rellena los campos");
182     }
183 }
```

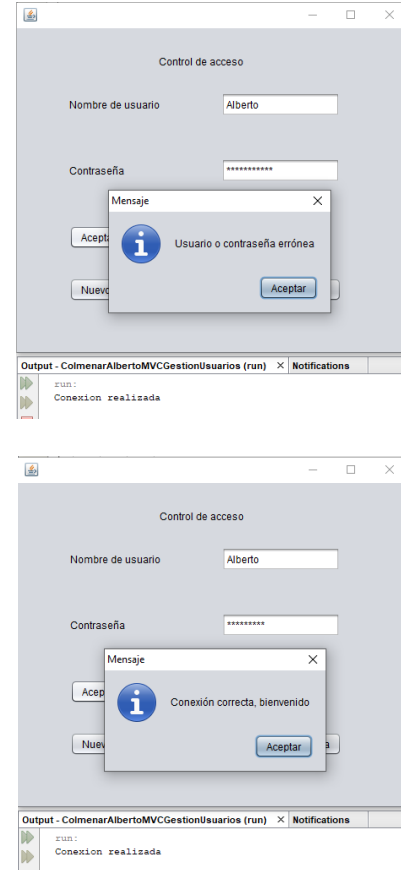
Comprueba si existe el usuario que se quiere logear ejecutando una query.

Usamos la clase Statement para escribir y lanzar la query a la BD previamente conectada.

Capturamos el resultado en un ResultSet y un boolean.

Verifica previamente que los campos nombre y contraseña están rellenos antes de llamar al método anterior.

Al final se muestra un mensaje de bienvenida o errorneo dependiendo de la respuesta de la query del método buscarUsuario().



Práctica 2.2 Actualizar contraseña de usuario

Flujo normal de la app para cambiar la contraseña del usuario Alberto, previamente tenía Albertoss y después de AlbertoNueva, ya que es una contraseña válida.

```
ColmenarAlbertoMVCGestionUsuarios.java | VentanaVerificar.java | CambiarContraseña.java |
Source | History |
67 | public boolean modificarContraseña(String nombre, String contraseña) {
68 |     boolean modificado = false;
69 |     try {
70 |         PreparedStatement sql = conexion.prepareStatement("UPDATE usuarios SET contraseña = ? WHERE nombre = ?");
71 |         sql.setString(1, contraseña);
72 |         sql.setString(2, nombre);
73 |         sql.executeUpdate();
74 |         modificado = true;
75 |         sql.close();
76 |     } catch (SQLException ex) {
77 |         System.err.println(ex.getMessage());
78 |     }
79 |     return modificado;
80 | }
81 |
82 | public String contrasenaValida(String contraseña) {
83 |     String errores = "";
84 |     if (!contrasena.matches("\\w+")) {
85 |         errores += "O Debe estar formada sólo por letras, números y el carácter \"_\".\n";
86 |     }
87 |     if (contrasena.length() < 8) {
88 |         errores += "O Debe tener 8 caracteres como mínimo.\n";
89 |     }
90 |     if (contrasena.matches("^\\d.*")) {
91 |         errores += "O No debe comenzar con un carácter numérico.\n";
92 |     }
93 |     if (!contrasena.matches("[A-Z].*")) {
94 |         errores += "O Debe contener al menos una letra en mayúscula.\n";
95 |     }
96 |     return errores;
97 | }
```

```
ColmenarAlbertoMVCGestionUsuarios.java | VentanaVerificar.java | CambiarContraseña.java |
Source | Design | History |
102 | private void botonGuardarActionPerformed(java.awt.event.ActionEvent evt) {
103 |     String contrasena1 = String.valueOf(cajaContrasena1.getPassword());
104 |     String contrasenaValida = mvcGestionUsuarios.contrasenaValida(contrasena1);
105 |     boolean contrasenaComprobada = true;
106 |     if (!contrasenaValida.equals("")) {
107 |         JOptionPane.showMessageDialog(this, contrasenaValida);
108 |         cajaContrasena1.setText("");
109 |         cajaContrasena2.setText("");
110 |         contrasenaComprobada = false;
111 |         return;
112 |     }
113 |     String contrasena2 = String.valueOf(cajaContrasena2.getPassword());
114 |     if (!contrasena1.equals(contrasena2)) {
115 |         JOptionPane.showMessageDialog(this, "Las dos contraseñas no son iguales");
116 |         cajaContrasena2.setText("");
117 |         contrasenaComprobada = false;
118 |     }
119 |     if (contrasenaComprobada) {
120 |         boolean guardado = mvcGestionUsuarios.modificarContraseña(nombre, contrasena1);
121 |         if (guardado) {
122 |             JOptionPane.showMessageDialog(this, "Se ha guardado correctamente");
123 |             this.dispose();
124 |         } else {
125 |             JOptionPane.showMessageDialog(this, "Error al guardar la contraseña");
126 |         }
127 |     }
128 | }
```

```
ColmenarAlbertoMVCGestionUsuarios.java | VentanaVerificar.java | CambiarContraseña.java | SQL 3 [jdbc:mysql://localhost:33...]
Connection: jdbc:mysql://localhost:3306/gestionusuarios2?zeroDateTimeBehavior=convertToNull [root on Default schema]
1 | SELECT * FROM usuarios LIMIT 100;
2 |
SELECT * FROM usuarios LI... X
# | nombre | contraseña
1 | qwe | qwe
2 | qwe | qwer
3 | Alberto | Albertoss
4 | Roberto | Robertoss
```

Nueva contraseña: [password field]

Repita contraseña: [password field]

Cancelar | Guardar

Mensaje: Se ha guardado correctamente. Aceptar

```
ColmenarAlbertoMVCGestionUsuarios.java | VentanaVerificar.java | CambiarContraseña.java | SQL 3 [jdbc:mysql://localhost:33...]
Connection: jdbc:mysql://localhost:3306/gestionusuarios2?zeroDateTimeBehavior=convertToNull [root on Default schema]
1 | SELECT * FROM usuarios LIMIT 100;
2 |
SELECT * FROM usuarios LI... X
# | nombre | contraseña
1 | qwe | qwe
2 | qwe | qwer
3 | Alberto | AlbertoNueva
4 | Roberto | Robertoss
```

Con una instancia de la clase PreparedStatement construimos el update parametrizado. Añadimos los parámetros y ejecutamos la query con executeQuery(). Devolvemos un boolean para mostrar un mensaje después.

La contraseña nueva se valida (siguiente diapositiva) y se comprueba que se escribe las dos veces iguales. Después se ejecuta el método anterior y se muestra el mensaje afirmativo o negativo.

Práctica 2.2 Validar contraseña

```
ColmenarAlbertoMVCGestionUsuarios.java x VentanaVerificar.java x
Source History
82 public String contrasenaValida(String contrasena) {
83     String errores = "";
84     if (!contrasena.matches("\\w+")) {
85         errores += "⊗ Debe estar formada sólo por letras, números y el carácter \"_\".\\n";
86     }
87     if (contrasena.length() < 8) {
88         errores += "⊗ Debe tener 8 caracteres como mínimo.\\n";
89     }
90     if (contrasena.matches("^\\d.*")) {
91         errores += "⊗ No debe comenzar con un carácter numérico.\\n";
92     }
93     if (!contrasena.matches(".*[A-Z]+.*")) {
94         errores += "⊗ Debe contener al menos una letra en mayúscula.\\n";
95     }
96     return errores;
97 }
98 }
99 }
```

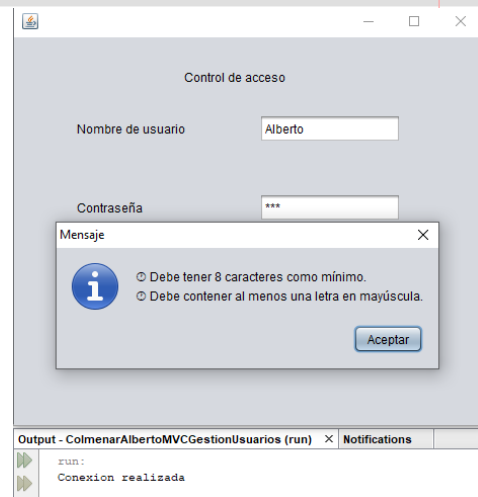
Método que comprueba que una contraseña es válida usando matches() y expresiones regulares.

Si es válida devuelve un String vacío, sino se van concatenando los diferentes errores en la contraseña para luego mostrarlos en un mensaje para el usuario.

Este método de validar se usa tanto en la siguiente imagen que se ejecuta al darle a la tecla Intro cuando escribes tu contraseña al logearte como al cambiar la contraseña (diapositiva anterior).

```
ColmenarAlbertoMVCGestionUsuarios.java x VentanaVerificar.java x
Source Design History
209 private void cajaContrasenaActionPerformed(java.awt.event.ActionEvent evt) {
210     if (camposRellenados()) {
211         String contrasena = String.valueOf(cajaContrasena.getPassword());
212         String contrasenaValida = mvcGestionUsuarios.contrasenaValida(contrasena);
213         if (contrasenaValida.equals("")) {
214             habilitarBotones();
215         } else {
216             JOptionPane.showMessageDialog(this, contrasenaValida);
217         }
218     }
219 }
220 }
```

Posible ejecución donde vemos que nuestra contraseña no tiene 8 caracteres ni ninguna mayúscula.

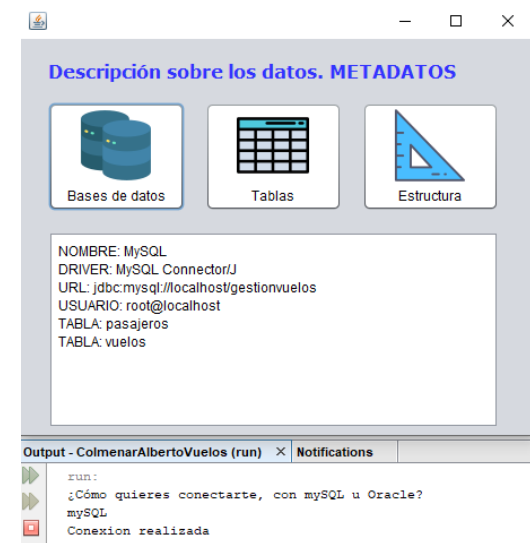


Práctica 2.3 Consulta y descripción de datos. Base de datos

```
ColmenarAlbertoMVCVuelos.java x ColmenarAlbertoVentanaDescripcion.java x
Source History
59 public String informaBD() {
60     String datos = "";
61     try {
62         DatabaseMetaData dbmd = conexion.getMetaData();
63         datos = String.format("NOMBRE: %s\n"
64             + "DRIVER: %s\n"
65             + "URL: %s\n"
66             + "USUARIO: %s\n",
67             dbmd.getDatabaseProductName(), dbmd.getDriverName(), dbmd.getURL(), dbmd.getUserName());
68         ResultSet tablas = dbmd.getTables(nombreBD, nombreBD, null, null);
69         while (tablas.next()) {
70             datos += "TABLA: " + tablas.getString("TABLE_NAME") + "\n";
71         }
72     } catch (SQLException ex) {
73         System.err.println("Error " + ex.getMessage());
74     }
75     return datos;
76 }
```

```
ColmenarAlbertoMVCVuelos.java x ColmenarAlbertoVentanaDescripcion.java x
Source Design History
117 private void botonBDActionPerformed(java.awt.event.ActionEvent evt) {
118     String datos = MVCVuelos.informaBD();
119     areaDatos.setText(datos);
120 }
```

La información pedida se vuelca directamente en el TextArea de debajo de los botones.



Con el método `getMetaData()` de `Connection` inicializamos una instancia de `DatabaseMetaData` que contiene la información de la BD, previamente conectada, como su nombre, driver, tablas. Retorna un `String` con toda la información pedida.

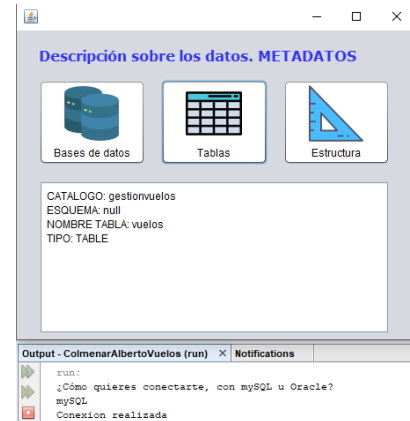
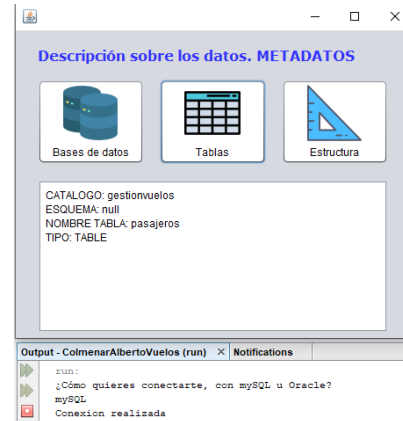
Práctica 2.3 Consulta y descripción de datos. Tablas

```
ColmenarAlbertoMVCVuelos.java x
Source History
78 public String informaTabla(String nombreTabla) {
79     String datos = "";
80     try {
81         DatabaseMetaData dbmd = conexion.getMetaData();
82         ResultSet tabla = dbmd.getTables(nombreBD, nombreBD, nombreTabla, null);
83         while (tabla.next()) {
84
85             datos += String.format("CATALOGO: %s\n"
86                                     + "ESQUEMA: %s\n"
87                                     + "NOMBRE TABLA: %s\n"
88                                     + "TIPO: %s\n",
89                                     tabla.getString("TABLE_CAT"),
90                                     tabla.getString("TABLE_SCHEM"),
91                                     tabla.getString("TABLE_NAME"),
92                                     tabla.getString("TABLE_TYPE"));
93         }
94     } catch (SQLException ex) {
95         System.err.println("Error " + ex.getMessage());
96     }
97     return datos;
98 }
```

```
ColmenarAlbertoMVCVuelos.java x ColmenarAlbertoVentanaDescripcion.java x
Source Design History
122 private void botonTablaActionPerformed(java.awt.event.ActionEvent evt) {
123     // muestra un Dialog con los nombres de las tablas para que el usuario escoja
124     String[] opciones = {"PASAJEROS", "VUELOS"};
125     int seleccionado = JOptionPane.showOptionDialog(this, "¿Qué tabla quieres ver?", "Información tabla",
126     JOptionPane.DEFAULT_OPTION, INFORMATION_MESSAGE, null, opciones, opciones[0]);
127     if (seleccionado > -1) {
128         String datos = MVCVuelos.informaTabla(opciones[seleccionado]);
129         areaDatos.setText(datos);
130     }
}
```

Creamos un JOptionPane con las opciones de los nombres de las tablas pasajeros y vuelos para que el usuario pueda elegir cual ver, llamar al método anterior y volcar su retorno en el TextArea.

Con una instancia de la clase DatabaseMetaData y el método getMetaData de Connection obtenemos la información de la tabla pasada por parámetro. Gracias al resultset podemos mostrar los distintos datos que pedía el enunciado.



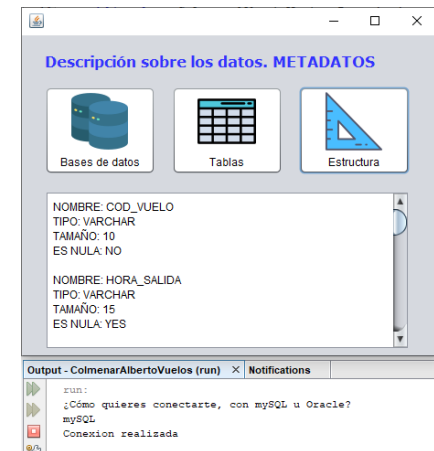
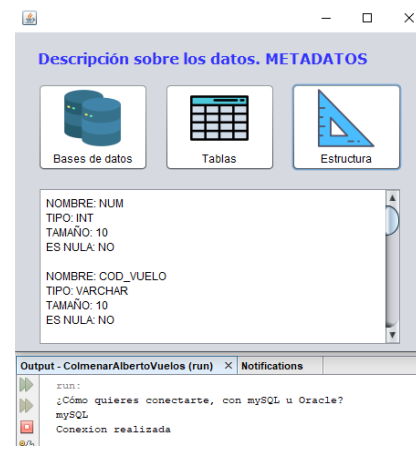
Práctica 2.3 Consulta y descripción de datos. Columnas

```
ColmenarAlbertoMVCVuelos.java x ColmenarAlbertoVentanaDescripcion.java x
Source History
101 public String informaColumnas(String nombreTabla) {
102     String datos = "";
103     try {
104         DatabaseMetaData dbmd = conexion.getMetaData();
105         ResultSet columnas = dbmd.getColumns(null, nombreBD, nombreTabla, null);
106
107         while (columnas.next()) {
108             datos += String.format("NOMBRE: %s\n"
109                                     + "TIPO: %s\n"
110                                     + "TAMAÑO: %s\n"
111                                     + "ES NULA: %s\n",
112                                     columnas.getString("COLUMN_NAME"),
113                                     columnas.getString("TYPE_NAME"),
114                                     columnas.getString("COLUMN_SIZE"),
115                                     columnas.getString("IS_NULLABLE"));
116         }
117     } catch (SQLException ex) {
118         System.err.println("Error " + ex.getMessage());
119     }
120     return datos;
121 }
```

```
ColmenarAlbertoMVCVuelos.java x ColmenarAlbertoVentanaDescripcion.java x
Source Design History
133 private void botonEstructuraActionPerformed(java.awt.event.ActionEvent evt) {
134     // muestra un Dialog con los nombres de las tablas para que el usuario escoja
135     String[] opciones = {"PASAJEROS", "VUELOS"};
136     int seleccionado = JOptionPane.showOptionDialog(this, "¿Qué tabla quieres ver?", "Información tabla",
137     JOptionPane.DEFAULT_OPTION, INFORMATION_MESSAGE, null, opciones, opciones[0]);
138     if (seleccionado > -1) {
139         String datos = MVCVuelos.informaColumnas(opciones[selectedorado]);
140         areaDatos.setText(datos);
141     }
142 }
```

Misma estructura para pedir al usuario que elija la tabla y una vez ejecutado el método se vuelca la información de en TextArea.

Obtenemos el DatabaseMetaData como anteriormente, esta vez en el resulset nos quedamos con las columnas de una tabla y concatenamos la información pedida.



Práctica 2.3 Listado de vuelos

```
ColmenarAlbertoMVCVuelos.java x ColmenarAlbertoVentanaVuelos.java x
Source History
123 public DefaultTableModel generarTablaVuelos() {
124     DefaultTableModel tabla = new DefaultTableModel();
125     try {
126         Statement stmt = conexion.createStatement();
127         ResultSet consulta = stmt.executeQuery("SELECT cod_vuelo,"
128             + " procedencia, destino FROM vuelos");
129         tabla.addColumn("COD_VUELO");
130         tabla.addColumn("PROCEDENCIA");
131         tabla.addColumn("DESTINO");
132         Object datos[] = new Object[3];
133
134         while (consulta.next()) {
135             for (int i = 0; i < 3; i++) {
136                 datos[i] = consulta.getObject(i + 1);
137             }
138             tabla.addRow(datos);
139         }
140         if (stmt != null) {
141             stmt.close();
142         }
143     } catch (SQLException ex) {
144         System.err.println(ex.getMessage());
145     }
146     return tabla;
147 }
```

Creamos un Statement con la conexión previa y ejecutamos una query para obtener todos los vuelos.

Como queremos mostrarlo en formato tabla nos hacemos una DefaultTableModel con las columnas que queremos y recorremos el resultSet añadiéndole la información.

```
ColmenarAlbertoMVCVuelos.java x ColmenarAlbertoVentanaVuelos.java x
Source Design History
98
99 private void botonListadoActionPerformed(java.awt.event.ActionEvent evt) {
100     DefaultTableModel tabla = MVCVuelos.generarTablaVuelos();
101     tablaVuelos.setModel(tabla);
102 }
```

Como el método nos devuelve ya la información en formato tabla la volcamos a la interfaz que es una tabla de Swing.



COD_VUELO	PROCEDENCIA	DESTINO
AI-1289-9	BONN	BARCELONA
AI-D7-347	MOSCÚ	BILBAO
AV-DC-347	ROMA	VALENCIA
AV-DC2-269	LA HAYA	MADRID
AV-DC9-233	SOFÍA	VALENCIA
BG-WW-PPX	FUENLABRADA	ALASKA
FR-DC-4667	SEVILLA	BRUSELAS
FR-DC2-269	MANILA	CÓRDOBA
FR-DC7-247	EL CAIRO	CÓRDOBA
IB-98779	LIMA	MADRID
IB-BA-46DC	MADRID	ROMA
IB-D5-347	PARIS	ZARAGOZA
IB-SP-4567	MADRID	PARIS
SP-DC-438	SEVILLA	MOSCÚ

Output - ColmenarAlbertoVuelos (run) x Notifications

run:
¿Cómo quieres conectarte, con MySQL u Oracle?
MySQL
Conexion realizada

Práctica 2.3 Listado de pasajeros de un vuelo

```
ColmenarAlbertoMVCVuelos.java ColmenarAlbertoVentanaPasajeros.java
Source History
149 public ArrayList obtenerListaCodigo() {
150     ArrayList<String> lista = new ArrayList();
151     String cod;
152     try {
153         Statement stmt = conexion.createStatement();
154         ResultSet resultado = stmt.executeQuery("SELECT COD_VUELO FROM vuelos");
155         while (resultado.next()) {
156             cod = resultado.getString("COD_VUELO");
157             lista.add(cod);
158         }
159     } catch (SQLException ex) {
160         System.err.println(ex.getMessage());
161     }
162     return lista;
163 }
```

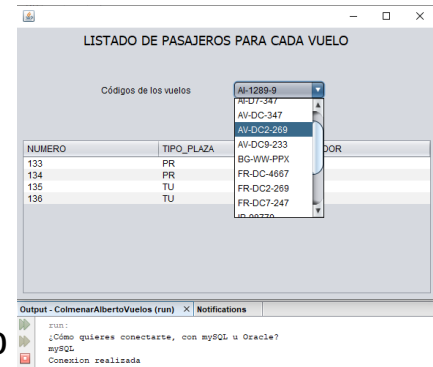
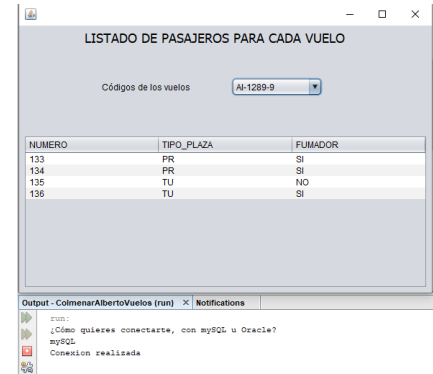
Creamos un arrayList para poder añadirlo al ComboBox con los códigos de los vuelos obtenidas con un Statement ejecutando la query y recorriéndolo añadiéndolo a la lista.

```
ColmenarAlbertoMVCVuelos.java ColmenarAlbertoVentanaPasajeros.java
Source Design History
14 public ColmenarAlbertoVentanaPasajeros() {
15     initComponents();
16     MVCVuelos.getConnection();
17     this.cargarListaCodigos();
18 }
19
20 public void cargarListaCodigos() {
21     ArrayList<String> lista = new ArrayList();
22     lista = MVCVuelos.obtenerListaCodigo();
23     for (String cod : lista) {
24         this.comboCodigos.addItem(cod);
25     }
26 }
27 }
```

Al crear la ventana hacemos la conexión a la BD y cargamos los códigos al ComboBox.

El código que esté seleccionado mostramos sus pasajeros

```
ColmenarAlbertoMVCVuelos.java ColmenarAlbertoVentanaPasajeros.java
Source Design History
119 private void comboCodigosActionPerformed(java.awt.event.ActionEvent evt) {
120     DefaultTableModel tabla = MVCVuelos.generarTablaPasajeros(String.valueOf(comboCodigos.getSelectedItem()));
121     tablaPasajeros.setModel(tabla);
122 }
123 }
```



Práctica 2.4 Conexión a Oracle desde Java

```
ColmenarAlbertoMVCVuelos.java | SQL 2 [jdbc:oracle:thin:@localhost:...] |
Source | History |
25 | public void getConnection() {
26 |     Scanner in = new Scanner(System.in);
27 |     String bd = "";
28 |     // pregunta como quiere conectarse
29 |     while ((!bd.equals("mySQL")) && !bd.equals("Oracle")) {
30 |         System.out.println("¿Cómo quieres conectarte, con mySQL u Oracle?");
31 |         bd = in.nextLine();
32 |     }
33 |
34 |     if (bd.equals("mySQL")) {
35 |         try {
36 |             nombreBD = "gestionvuelos";
37 |             Class.forName("com.mysql.cj.jdbc.Driver");
38 |             conexion = (Connection) DriverManager.getConnection("jdbc:mysql://localhost/gestionvuelos", "root", "");
39 |             System.out.println("Conexion realizada");
40 |         } catch (ClassNotFoundException ex) {
41 |             System.err.println("Error " + ex.getMessage());
42 |         } catch (SQLException ex) {
43 |             System.err.println("Error de conexión " + ex.getMessage());
44 |         }
45 |     } else if (bd.equals("Oracle")) {
46 |         nombreBD = "SYSTEM";
47 |         try {
48 |             Class.forName("oracle.jdbc.driver.OracleDriver");
49 |             conexion = (Connection) DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE", "system", "Dam-2022");
50 |             System.out.println("Conexion realizada");
51 |         } catch (ClassNotFoundException ex) {
52 |             System.err.println("Error " + ex.getMessage());
53 |         } catch (SQLException ex) {
54 |             System.err.println("Error de conexión " + ex.getMessage());
55 |         }
56 |     }
57 | }
```

Connection: jdbc:oracle:thin:@localhost:1521:XE [system on SYSTEM]

```
1 SELECT * FROM "SYSTEM".PASAJEROS WHERE ROWNUM <= 100;
2 SELECT * FROM "SYSTEM".VUELOS WHERE ROWNUM <= 100;
```

#	NUM	COD_VUELO	TIPO_PLAZA	FUMADOR
1	123	IB-SP-4567	TU	SI
2	124	IB-SP-4567	PR	SI
3	125	IB-SP-4567	PR	NO
4	126	IB-BA-46DC	TU	SI
5	127	IB-BA-46DC	PR	SI
6	128	FR-DC-4667	TU	NO
7	129	FR-DC-4667	TU	SI
8	130	AV-DC9-233	TU	SI
9	131	AV-DC9-233	TU	NO
10	132	AV-DC9-233	PR	SI
11	133	IB-D5-347	PR	SI
12	134	IB-D5-347	PR	SI
13	135	IB-D5-347	TU	NO
14	136	IB-D5-347	TU	SI
15	137	FR-DC-4667	TU	SI
16	138	FR-DC-4667	TU	NO
17	139	FR-DC-4667	PR	SI

Se muestra el contenido de las dos tablas y de la conexión de Oracle.

Descripción sobre los datos. METADATOS

Base de datos | Tablas | Estructura

Output X Notifications | ColmenarAlbertoVuelos (run) X

```
not:
¿Cómo quieres conectarte, con mySQL u Oracle?
Oracle
Conexion realizada
```

Connection: jdbc:oracle:thin:@localhost:1521:XE [system on SYSTEM]

```
1 SELECT * FROM "SYSTEM".PASAJEROS WHERE ROWNUM <= 100;
2 SELECT * FROM "SYSTEM".VUELOS WHERE ROWNUM <= 100;
```

#	COD_VUELO	HORA_SALIDA	DESTINO	PROCEDENCIA	PLAZAS_FUMADOR	PLAZAS_NO_FUMADOR	PLAZAS_TURISTA	PLAZAS_PRIMERA
1	IB-SP-4567	27/03/99-10:30	PARIS	MADRID	100	100	160	40
2	IB-BA-46DC	28/03/99-12:30	ROMA	MADRID	90	100	160	30
3	FR-DC-4667	28/03/99-13:30	BRUSLAS	SEVILLA	100	100	160	30
4	AV-OC-947	28/03/99-12:35	VALENCIA	ROMA	200	100	210	90
5	SP-OC-438	30/03/99-09:20	MOSCÚ	SEVILLA	100	100	160	30
6	AL-07-347	30/03/99-13:35	BILBAO	MOSCÚ	100	200	210	90
7	IB-D5-347	01/04/99-12:35	ZARAGOZA	PARIS	100	200	210	90
8	FR-OC-247	01/04/99-15:35	CORDOBA	EL CARO	100	100	100	100
9	AV-OC3-233	01/04/99-17:35	VALENCIA	SOFIA	100	100	100	100
10	FR-OC2-289	01/04/99-19:00	CORDOBA	MALEA	100	100	100	100
11	IB-98779	02/04/99-08:00	MADRID	LIMA	100	100	100	100
12	AV-OC2-289	02/04/99-12:00	MADRID	LA HAYA	100	100	100	100
13	AL-1289-9	02/04/99-14:30	BARCELONA	BONN	150	100	100	100

Se muestran la conexión de Oracle (debajo)
El usuario escribe por consola 'Oracle' para conectarse a ella.
Se carga el driver de Oracle y se conecta a la BD previamente creada.