

## ***Unidad 2***

### ***Manejo de conectores***

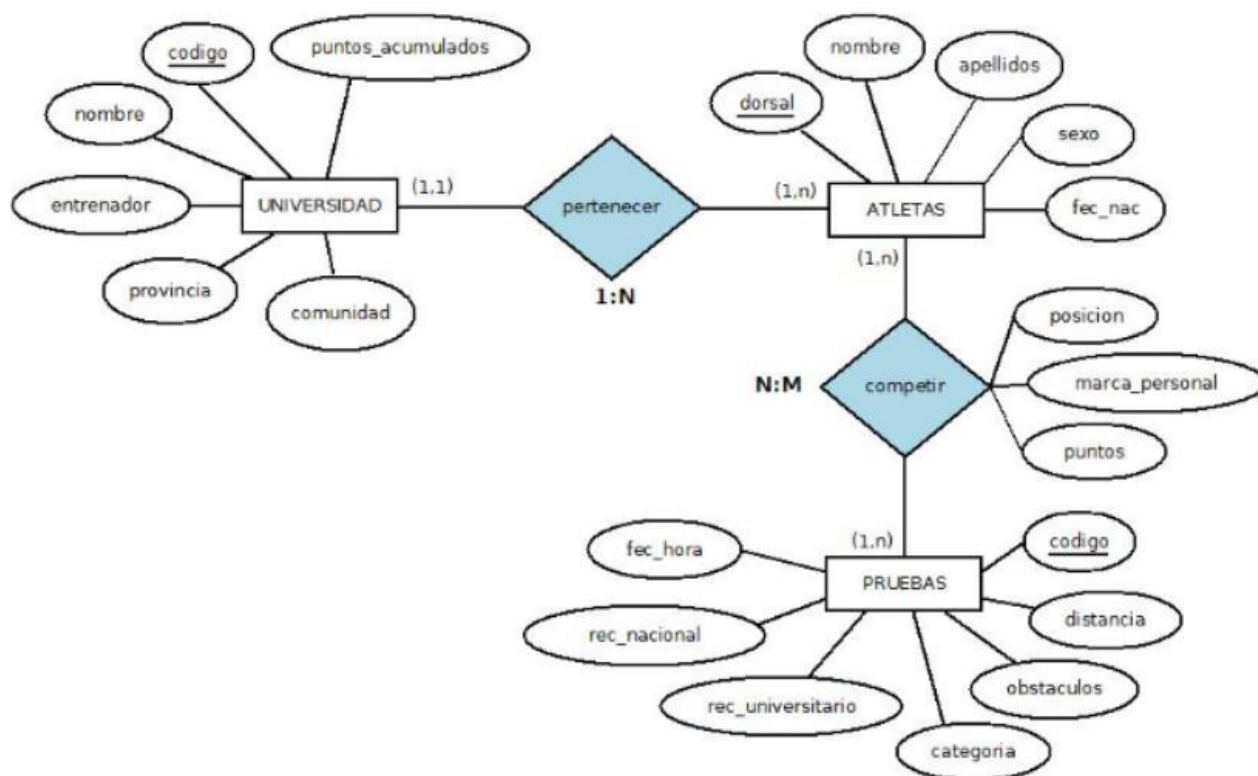
## **Tarea AD02**

### **Manejando conectores**

1.	Enunciado de la tarea .....	2
2.	Preparación del Script, acceso a la consola y comprobación de datos .....	4
3.	Estructura del proyecto Netbeans y establecimiento de la conexión a la base de datos .....	9
4.	Consulta B1: Atletas por comunidad .....	12
5.	Consulta B2. Puntos totales por dorsal.....	16
6.	Actualización C. Modificación de la distancia de una prueba .....	19
7.	Procedimiento D. Ejecución de Procedimiento Almacenado .....	23
8.	Gestión de Excepciones y Cierre de Recursos .....	27
9.	Anexos.....	28
	a) Atributos únicos .....	28
	b) Limpieza de los objetos ComboBox .....	29
	c) Sentencias SQL implementadas en el código java del presente proyecto netbeans ...	30
	d) Código de cada uno de los botones de la interfaz gráfica .....	33
	e) Puesta en servicio de la aplicación .....	36

## 1. Enunciado de la tarea

El diagrama del modelo E/R de la base de datos **Campeonato\_Atlentismo** es el siguiente:



Partiendo de esta base de datos realizar los siguientes ejercicios:

### EJERCICIO 1:

Lanza el [script SQL](#) de creación del esquema de la base de datos **Campeonato\_atletismo** y asegúrate de que se crean todas las tablas y se insertan los registros correspondientes. Puedes usar, bien un cliente gráfico (MySQL Workbench) o bien un cliente en modo texto (Línea de comandos MySQL). Debes documentar los pasos realizados en el documento de entrega de la tarea.

### EJERCICIO 2:

Crea un proyecto en Netbeans con nombre **Nombre\_Apellido1\_AD2\_E2** para crear una aplicación con un interfaz gráfico usando Swing, que permita realizar las siguientes acciones:

- Establecer conexión con la base de datos permitiendo al usuario introducir usuario y contraseña.
- Mostrar en la aplicación, el resultado de implementar a través de sentencias preparadas y parametrizadas las siguientes consultas:
  - Obtener el nombre y apellidos de los atletas junto al nombre de la universidad a la que pertenece de todos aquellos atletas de la comunidad que indique el usuario de la aplicación (la aplicación debe permitir al usuario indicar una comunidad).

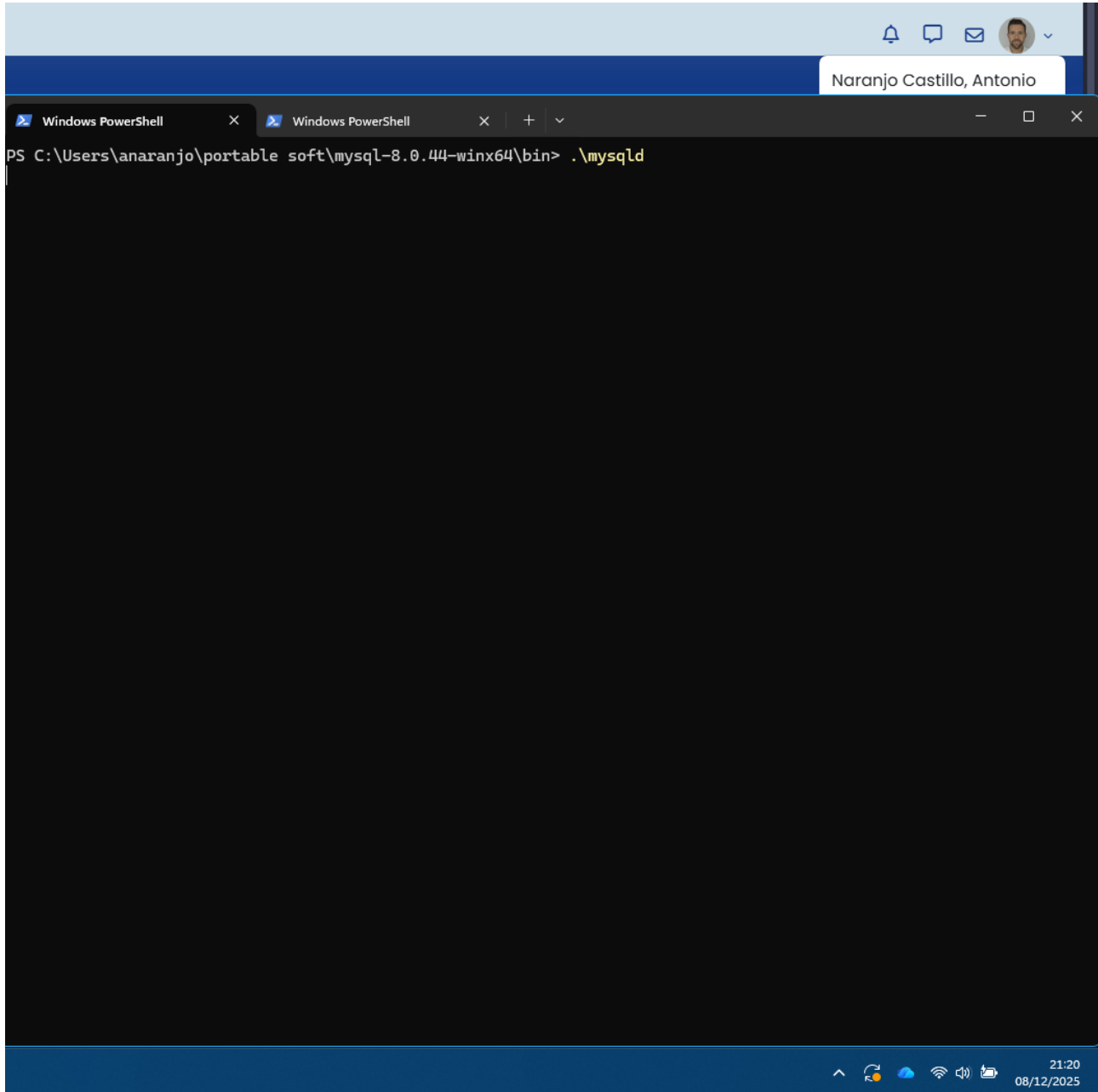
2. Calcular la cantidad total de puntos que ha obtenido un atleta determinado (identificado por su dorsal) en las diferentes pruebas (la aplicación debe permitir al usuario indicar el número de dorsal).
- C. Modificar, a través de una sentencia preparada y parametrizada, la distancia de una prueba determinada indicada por su código (el usuario introducirá el código de la prueba y la nueva distancia).
- D. Ejecutar un procedimiento almacenado llamado **atletas\_posicion** que devuelva en un parámetro de salida (que se mostrará en la aplicación), el número total de atletas que hayan acabado en una determinada posición en alguna prueba. Se debe pasar como parámetro al procedimiento la posición por la que se quiere consultar, que será introducida por el usuario (se debe adjuntar en la entrega de la tarea el script de creación del procedimiento almacenado).

**IMPORTANTE:**

- La aplicación debe gestionar las posibles excepciones y errores que puedan presentarse, así como el cierre de recursos utilizados (usa la sentencia try-catch-finally o equivalente).
- La aplicación deberá informar adecuadamente al usuario de los posibles errores, datos esperados, etc., para guiar al usuario final en el uso de la misma.

## 2. Preparación del Script, acceso a la consola y comprobación de datos

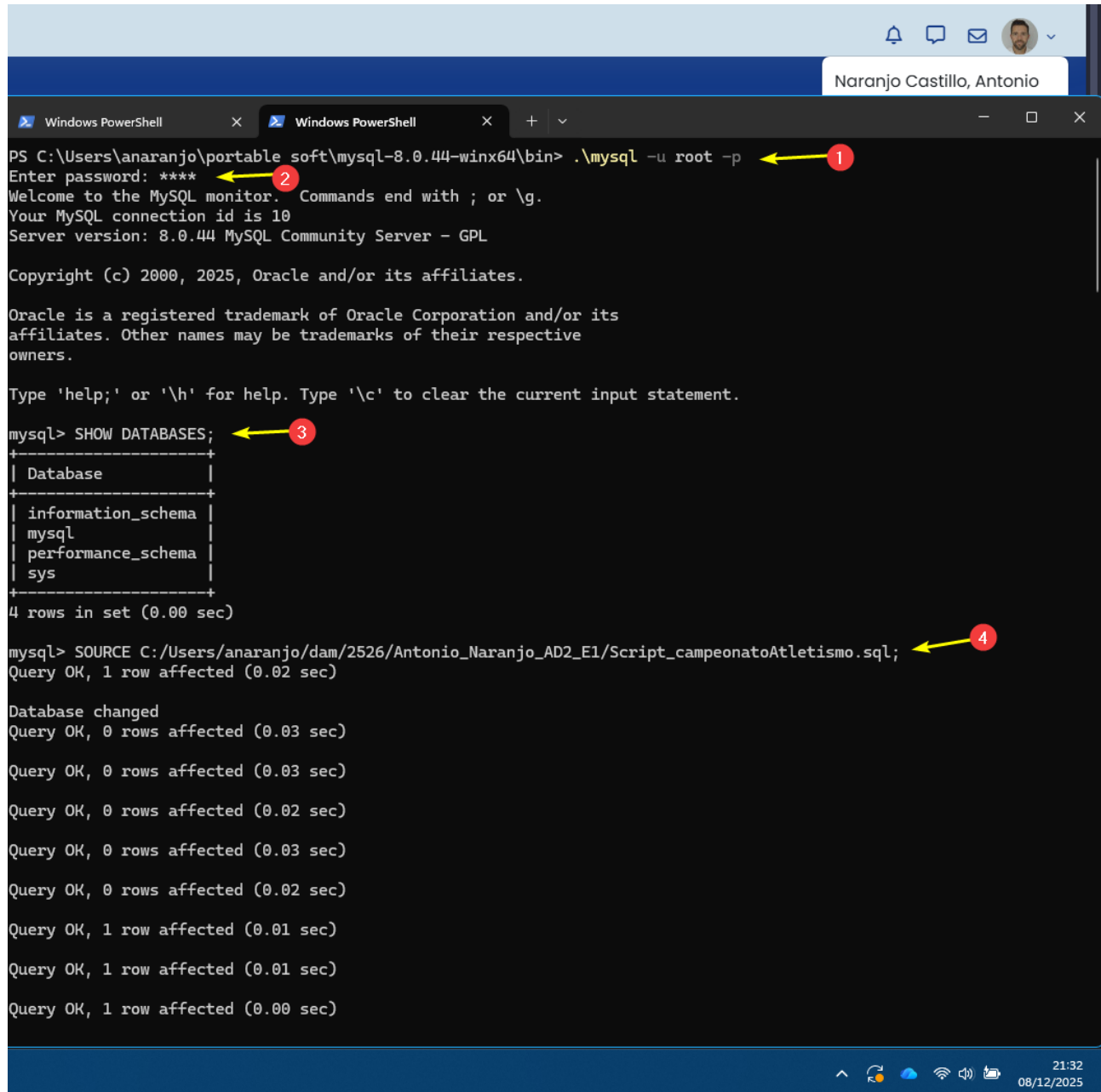
Se ejecuta el cliente en modo texto de MySQL versión 8.0.44 desde línea de comandos.



The screenshot shows a Windows PowerShell terminal window. The title bar at the top indicates the user is 'Naranjo Castillo, Antonio'. The terminal has two tabs, both labeled 'Windows PowerShell'. The command prompt shows the current directory as 'C:\Users\anaranjo\portable soft\mysql-8.0.44-winx64\bin' and the command '.\mysqld' has been entered. The terminal is currently empty, waiting for the command to execute. The Windows taskbar at the bottom shows the time as 21:20 on 08/12/2025.

```
PS C:\Users\anaranjo\portable soft\mysql-8.0.44-winx64\bin> .\mysqld
```

Desde otra pestaña de la línea de comandos, se accede al cliente MySQL con las credenciales de administrador (usuario root y contraseña root) para asegurar los permisos necesarios para la creación de la base de datos y la carga de datos.



```
PS C:\Users\anaranjo\portable soft\mysql-8.0.44-winx64\bin> .\mysql -u root -p
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 10
Server version: 8.0.44 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
4 rows in set (0.00 sec)

mysql> SOURCE C:/Users/anaranjo/dam/2526/Antonio_Naranjo_AD2_E1/Script_campeonatoAtletismo.sql;
Query OK, 1 row affected (0.02 sec)

Database changed
Query OK, 0 rows affected (0.03 sec)

Query OK, 0 rows affected (0.03 sec)

Query OK, 0 rows affected (0.02 sec)

Query OK, 0 rows affected (0.03 sec)

Query OK, 0 rows affected (0.02 sec)

Query OK, 1 row affected (0.01 sec)

Query OK, 1 row affected (0.01 sec)

Query OK, 1 row affected (0.00 sec)
```

Luego, se consultan las bases de datos existentes, y se ejecuta el script facilitado para llevar a cabo la presente tarea.

Se comprueba que la base de datos campeonato\_atletismo se ha cargado con éxito.

Se asegura que la base de datos en uso sea campeonato\_atletismo.

Se muestran las tablas existentes en la base de datos.

Se muestran los datos de la tabla atleta, a modo de comprobación de que los datos han sido incorporados satisfactoriamente.

```
Query OK, 1 row affected (0.00 sec)

mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| campeonato_atletismo |
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0.00 sec)

mysql> USE campeonato_atletismo;
Database changed

mysql> SHOW TABLES;
+-----+
| Tables_in_campeonato_atletismo |
+-----+
| atleta |
| competir |
| podium |
| prueba |
| universidad |
+-----+
5 rows in set (0.00 sec)

mysql> SELECT * FROM atleta;
+-----+
| dorsal | nombre | apellidos | sexo | fec_nac | universidad |
+-----+
| 0151 | Jaime | Pérez López | M | 1998-01-20 | UAL |
| 0152 | Lucía | Gil Martínez | F | 1998-01-15 | UA |
| 0153 | Adrián | Ruiz García | M | 1995-02-11 | UA |
| 0154 | Pedro | Sanz Lorenzo | M | 1996-02-01 | UAM |
| 0155 | Carmen | Aguirre Soria | F | 1997-03-02 | UPV |
| 0156 | Carlos | Beltrán Gómez | M | 1998-03-12 | UAB |
| 0157 | José | Gómez Gil | M | 1998-03-11 | UA |
| 0158 | Manuel | Rodríguez Castilla | M | 1997-04-21 | UPV |
| 0159 | Sara | Castro Ramírez | F | 1996-04-28 | UAB |
| 0160 | María | Valenzuela Garó | F | 1996-05-26 | UA |
| 0161 | Juan | Martínez García | M | 1997-05-12 | UGR |
| 0162 | Luis | Suliman Tez | M | 1996-06-11 | US |
| 0163 | Diego | Arganda Ruiz | M | 1997-06-10 | UIB |
+-----+
```

Se comprueban los datos de algunas tablas más PRUEBA y COMPETIR.

Windows PowerShell

Naranjo Castillo, Antonio

```
0170 | Alex | Castillo Giménez | M | 1996-11-26 | UGR |
0171 | Mauro | Silva Torres | M | 1996-11-27 | US |
0172 | Silvia | Sanz Barberó | F | 1996-12-11 | US |
0173 | Luisa | Fernández López | F | 1996-03-01 | UA |
0174 | Carolina | Estévez Peláez | F | 1998-11-03 | UIB |
0175 | Alba | Gil Muñoz | F | 1997-10-12 | US |
0176 | Mar | Palao Yuste | F | 1996-10-03 | UAB |
0177 | Candela | Martínez Gómez | F | 1996-08-23 | UAM |
0178 | Carla | Suárez Pineda | F | 1997-07-14 | UAM |
-----
28 rows in set (0.00 sec)
```

mysql> SELECT \* FROM prueba; 1

codigo	distancia	obstaculos	categoria	fec_hora	rec_universitario	rec_nacional
100LF	100	N	F	2017-02-03 09:30:00	00:00:12.020	00:00:11.060
100LM	100	N	M	2017-02-03 09:00:00	00:00:10.110	00:00:10.060
1500F	1500	N	F	2017-02-04 09:00:00	00:04:37.010	00:03:59.410
1500M	1500	N	M	2017-02-04 12:00:00	00:04:02.380	00:03:28.950
3000OF	3000	S	F	2017-02-05 09:00:00	00:11:12.380	00:09:07.320
3000OM	3000	S	M	2017-02-04 11:00:00	00:10:13.830	00:08:07.440
400VF	400	S	F	2017-02-03 10:30:00	00:00:57.198	00:00:55.230
400VM	400	S	M	2017-02-03 10:00:00	00:00:50.080	00:00:48.870

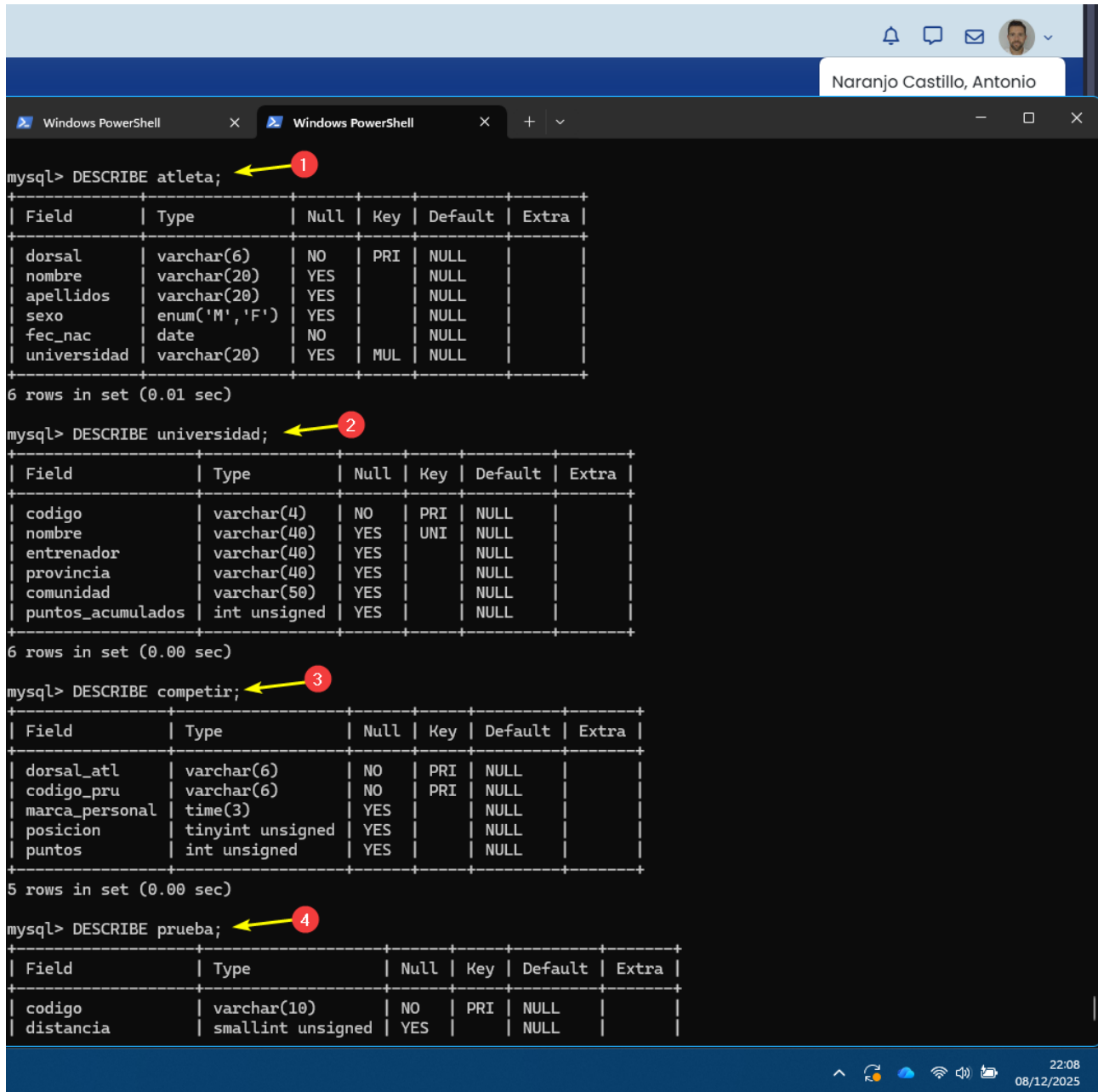
8 rows in set (0.00 sec)

mysql> SELECT \* FROM competir; 2

dorsal_atl	codigo_pru	marca_personal	posicion	puntos
0151	100LM	00:00:10.560	2	20
0151	1500M	00:04:55.380	8	0
0151	3000OM	00:12:40.100	11	0
0151	400VM	00:00:53.890	8	0
0152	100LF	00:00:12.100	1	30
0152	1500F	00:04:39.230	1	30
0152	3000OF	00:11:24.290	2	20
0153	100LM	00:00:10.210	1	30
0153	1500M	00:04:03.380	1	30
0153	3000OM	00:10:43.030	4	10
0153	400VM	00:00:50.420	1	30
0154	1500M	00:05:02.380	9	0
0154	3000OM	00:11:22.540	7	0
0154	400VM	00:00:52.080	6	0

22:06 08/12/2025

Y los tipos de datos almacenados en cada una de ellas.



```
mysql> DESCRIBE atleta;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| dorsal     | varchar(6)    | NO   | PRI | NULL    |       |
| nombre     | varchar(20)   | YES  |     | NULL    |       |
| apellidos  | varchar(20)   | YES  |     | NULL    |       |
| sexo       | enum('M','F') | YES  |     | NULL    |       |
| fec_nac    | date          | NO   |     | NULL    |       |
| universidad| varchar(20)   | YES  | MUL | NULL    |       |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.01 sec)

mysql> DESCRIBE universidad;
+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| codigo         | varchar(4)    | NO   | PRI | NULL    |       |
| nombre        | varchar(40)   | YES  | UNI | NULL    |       |
| entrenador     | varchar(40)   | YES  |     | NULL    |       |
| provincia      | varchar(40)   | YES  |     | NULL    |       |
| comunidad      | varchar(50)   | YES  |     | NULL    |       |
| puntos_acumulados | int unsigned | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)

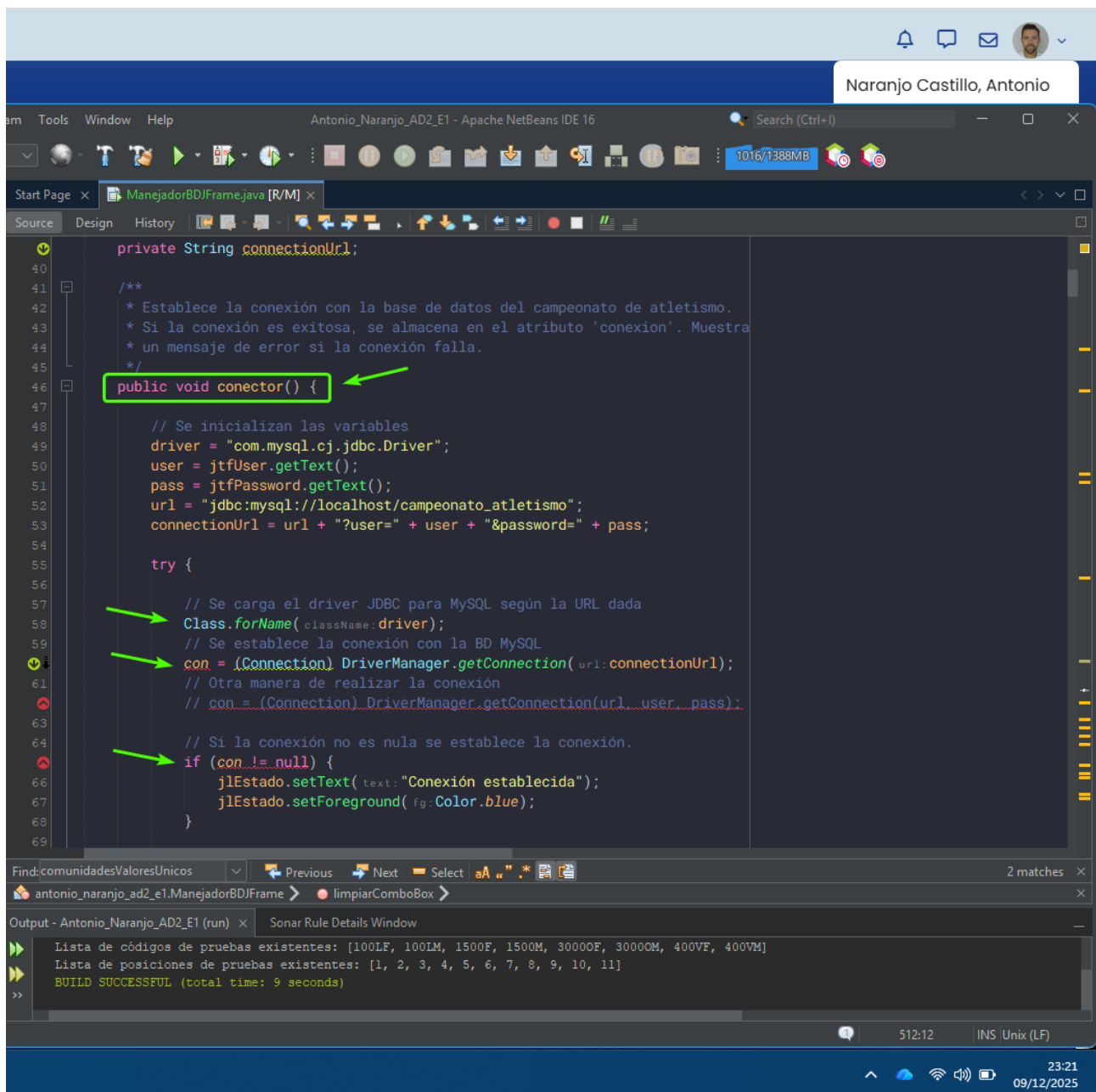
mysql> DESCRIBE competir;
+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| dorsal_atl     | varchar(6)    | NO   | PRI | NULL    |       |
| codigo_pru     | varchar(6)    | NO   | PRI | NULL    |       |
| marca_personal | time(3)       | YES  |     | NULL    |       |
| posicion       | tinyint unsigned | YES  |     | NULL    |       |
| puntos         | int unsigned  | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.00 sec)

mysql> DESCRIBE prueba;
+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| codigo         | varchar(10)   | NO   | PRI | NULL    |       |
| distancia      | smallint unsigned | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
```



### 3. Estructura del proyecto Netbeans y establecimiento de la conexión a la base de datos

Se crea el proyecto en NetBeans Antonio\_Naranjo\_AD2\_E1 y la clase principal ManejadorBDJFrame. Para el manejo de la conexión se implementa un método llamado conector(). Este método establece la conexión JDBC según la siguiente imagen.

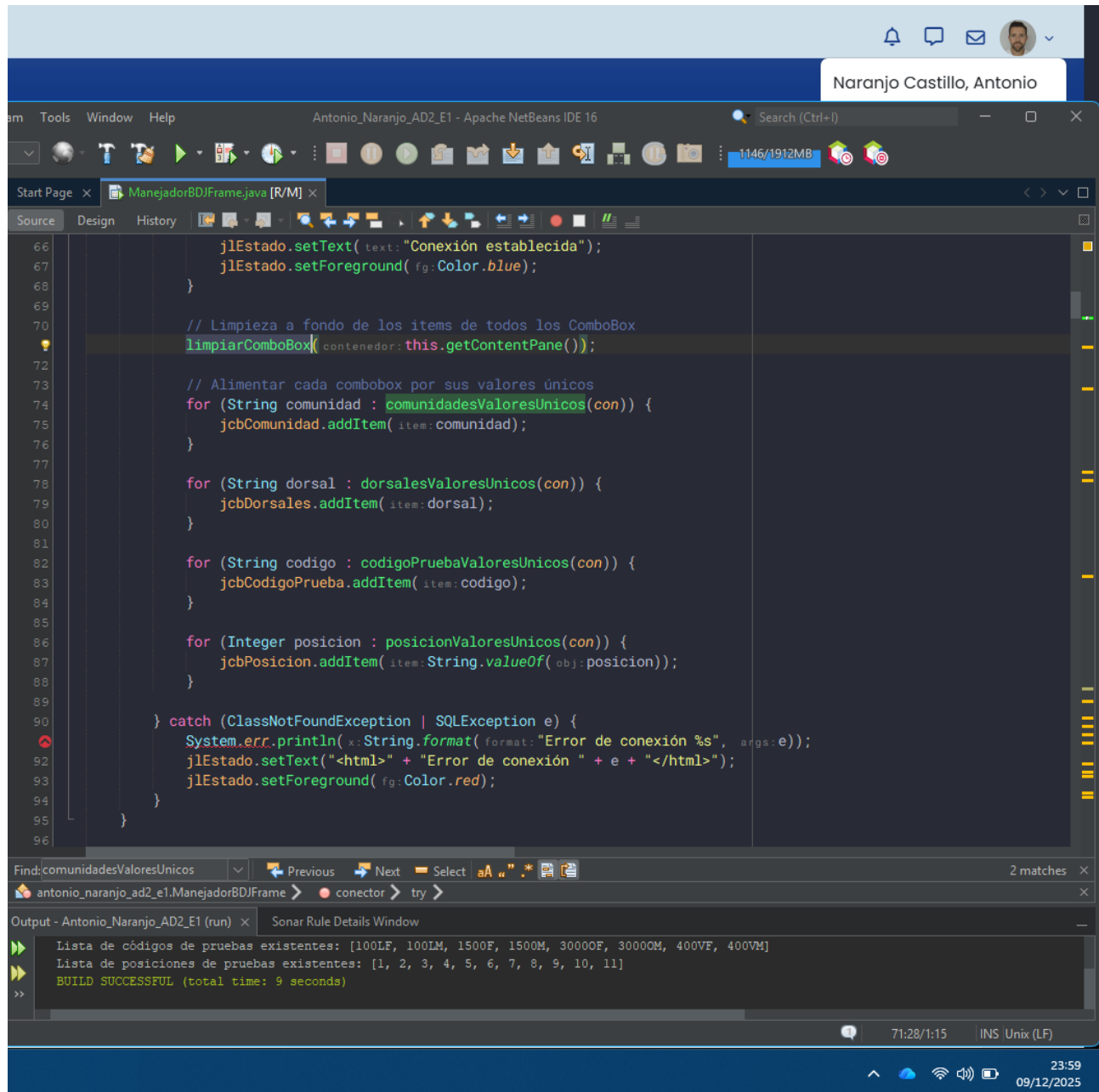


Se carga el driver JDBC empleando el método estático `forName()` de la clase `Class`, pasando como argumento el nombre del conector como un `String`.

Se establece la conexión creando un objeto `Connection` e iniciándolo por medio del método estático `getConnection` de la clase `DriverManager`, pasando como argumento la dirección URL completa donde se define tanto la dirección de la base de datos `campeonato_atletismo` (ubicada en el propio PC o `localhost`), así como el usuario y la contraseña. Se deja comentado un método más moderno

para establecer dicha conexión, llamando directamente al método `getConnetion` anterior y pasándole por argumento directamente la dirección completa de la base de datos, usuario y contraseña.

Se aprovecha este mismo método `conector()` para cargar los `comboBox` de otros paneles del aplicativo Swing diseñado. Previamente se limpian de datos existentes mediante el método `limpiarComboBox()` aportando como argumento el contenedor `JFrame`.



```
66         jlEstado.setText(text:"Conexión establecida");
67         jlEstado.setForeground(fg:Color.blue);
68     }
69
70     // Limpieza a fondo de los items de todos los ComboBox
71     limpiarComboBox(contenedor: this.getContentPane());
72
73     // Alimentar cada comboBox por sus valores únicos
74     for (String comunidad : comunidadesValoresUnicos(con)) {
75         jcbComunidad.addItem(item:comunidad);
76     }
77
78     for (String dorsal : dorsalesValoresUnicos(con)) {
79         jcbDorsales.addItem(item:dorsal);
80     }
81
82     for (String codigo : codigoPruebaValoresUnicos(con)) {
83         jcbCodigoPrueba.addItem(item:codigo);
84     }
85
86     for (Integer posicion : posicionValoresUnicos(con)) {
87         jcbPosicion.addItem(item:String.valueOf(obj:posicion));
88     }
89
90     } catch (ClassNotFoundException | SQLException e) {
91         System.err.println(x:String.format(format:"Error de conexión %s", args:e));
92         jlEstado.setText("<html>" + "Error de conexión " + e + "</html>");
93         jlEstado.setForeground(fg:Color.red);
94     }
95 }
96
```

Find:comunidadesValoresUnicos 2 matches

antonio\_naranjo\_ad2\_e1.ManejadorBDJFrame > conector > try >

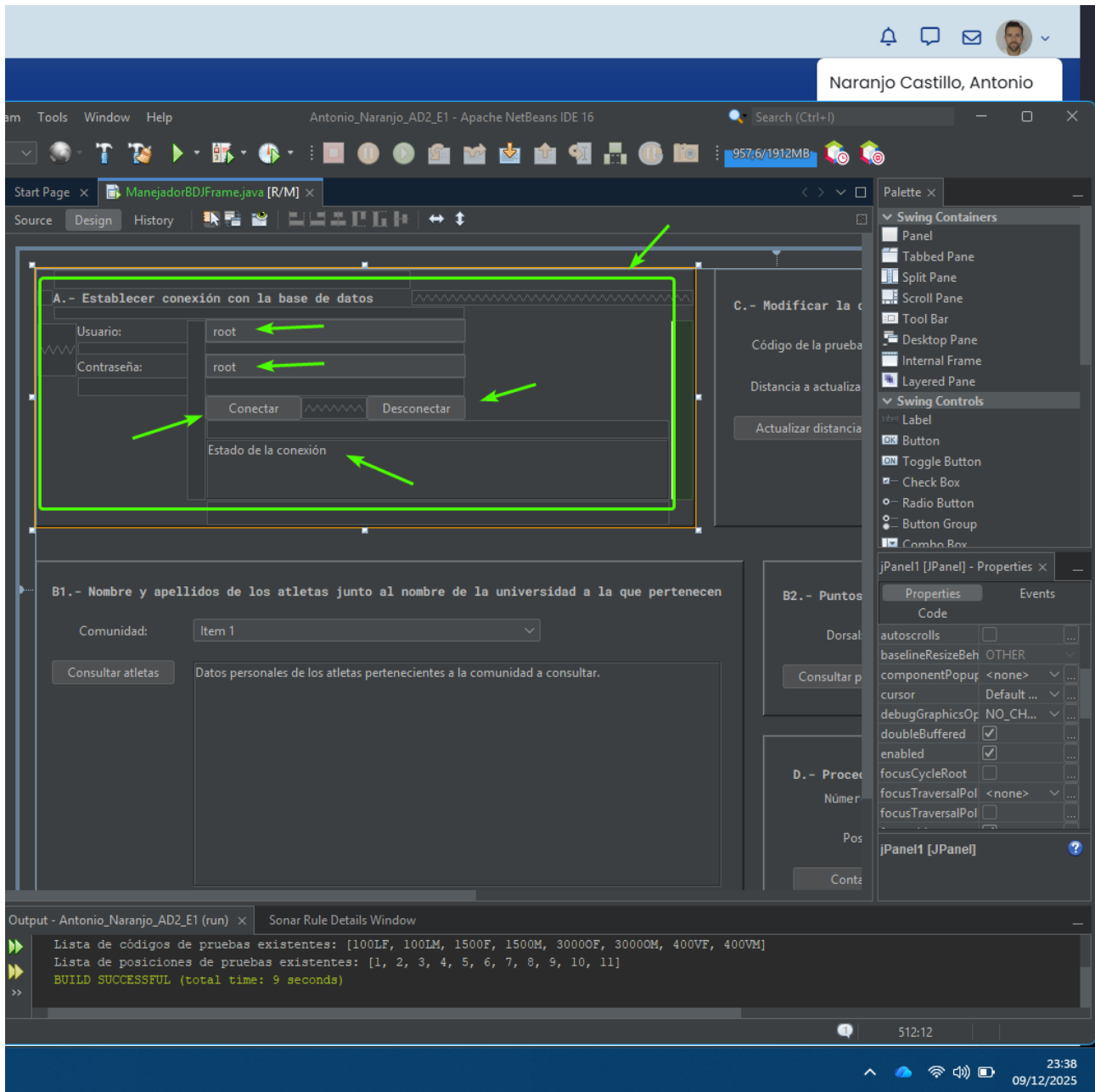
Output - Antonio\_Naranjo\_AD2\_E1 (run) Sonar Rule Details Window

```
>> Lista de códigos de pruebas existentes: [100LF, 100LM, 1500F, 1500M, 3000OF, 3000OM, 400VF, 400VM]
>> Lista de posiciones de pruebas existentes: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
>> BUILD SUCCESSFUL (total time: 9 seconds)
```

71:28/1:15 INS Unix (LF) 23:59 09/12/2025

Tanto si la conexión se establece con éxito o no se lanza un mensaje en una etiqueta establecida en la interfaz gráfica.

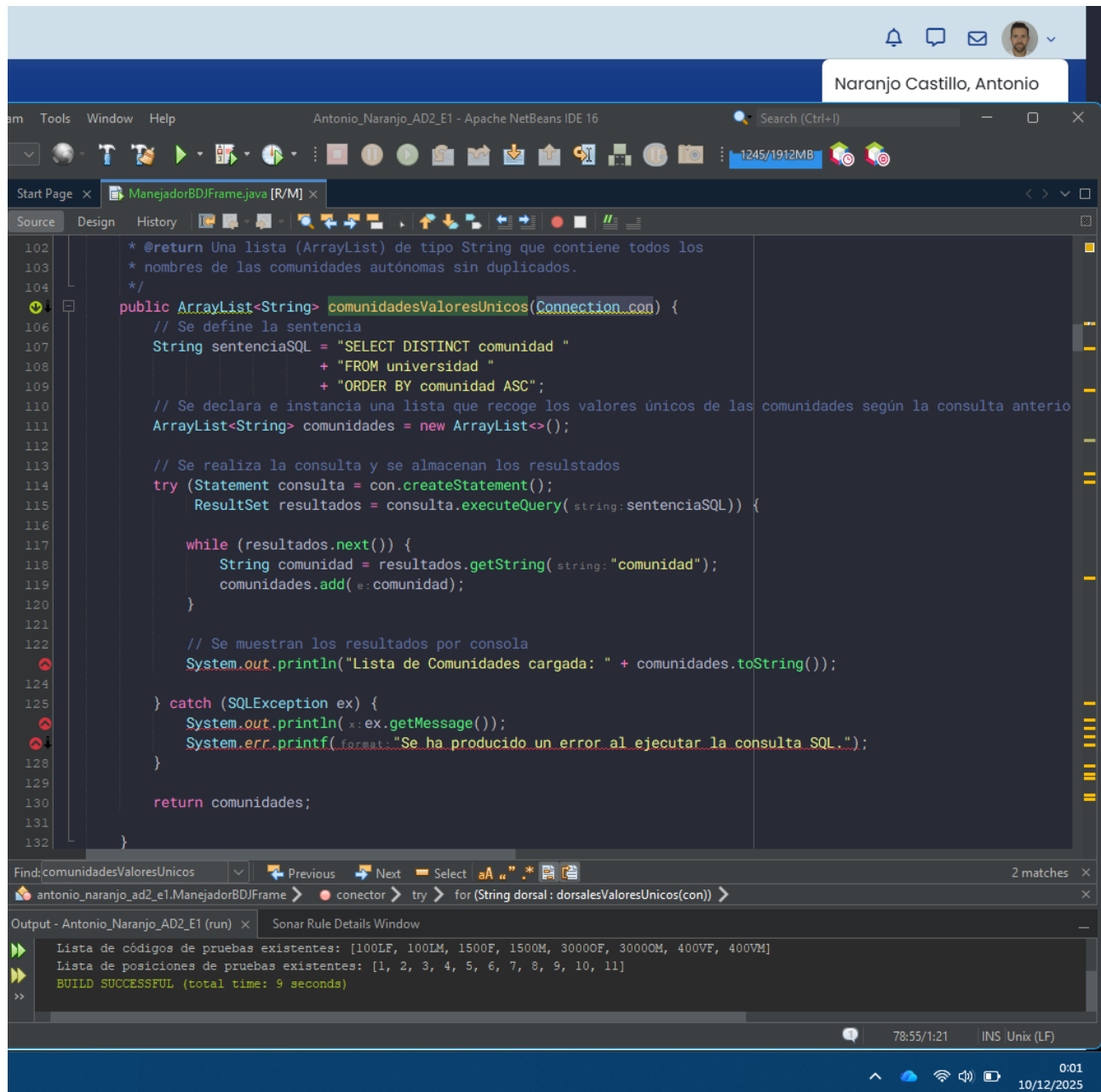
Se diseña una interfaz usando Swing con un panel reservado para establecer la conexión, incluyendo campos de texto para introducir usuario y contraseña, botones para activar/desactivar la conexión y una etiqueta para mostrar el resultado obtenido.



#### 4. Consulta B1: Atletas por comunidad

Se obtienen el nombre y apellidos de los atletas junto al nombre de la universidad a la que pertenece de todos aquellos atletas de la comunidad que indique el usuario de la aplicación.

Se preparan los ítems del ComboBox que podrá emplear el usuario de la aplicación para seleccionar la comunidad. Para ello se implementa un método llamado `comunidadesValoresUnicos()` al cual se le aporta como argumento el objeto `Connection` `con` creado mediante el método `conector()`.



```
102  * @return Una lista (ArrayList) de tipo String que contiene todos los
103  * nombres de las comunidades autónomas sin duplicados.
104  */
105  public ArrayList<String> comunidadesValoresUnicos(Connection con) {
106      // Se define la sentencia
107      String sentenciaSQL = "SELECT DISTINCT comunidad "
108                          + "FROM universidad "
109                          + "ORDER BY comunidad ASC";
110      // Se declara e instancia una lista que recoge los valores únicos de las comunidades según la consulta anterior
111      ArrayList<String> comunidades = new ArrayList<>();
112
113      // Se realiza la consulta y se almacenan los resultados
114      try (Statement consulta = con.createStatement();
115           ResultSet resultados = consulta.executeQuery(sentenciaSQL)) {
116
117          while (resultados.next()) {
118              String comunidad = resultados.getString("comunidad");
119              comunidades.add(comunidad);
120          }
121
122          // Se muestran los resultados por consola
123          System.out.println("Lista de Comunidades cargada: " + comunidades.toString());
124
125      } catch (SQLException ex) {
126          System.out.println(ex.getMessage());
127          System.err.printf("Se ha producido un error al ejecutar la consulta SQL.");
128      }
129
130      return comunidades;
131  }
132  }
```

Find:comunidadesValoresUnicos 2 matches

antonio\_naranjo\_ad2\_e1.ManejadorBDJFrame > conector > try > for (String dorsal : dorsalesValoresUnicos(con)) >

Output - Antonio\_Naranjo\_AD2\_E1 (run) x Sonar Rule Details Window

```
>> Lista de códigos de pruebas existentes: [100LF, 100LM, 1500F, 1500M, 3000OF, 3000OM, 400VF, 400VM]
>> Lista de posiciones de pruebas existentes: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
>> BUILD SUCCESSFUL (total time: 9 seconds)
```

78:55/1:21 INS Unix (LF) 0:01 10/12/2025

Se define la sentencia que selecciona los valores únicos de las comunidades que existen en la entidad universidad.

Se emplea la clase `Statement` y la clase `ResultSet` para crear la sentencia con la primera y recoger los datos de la consulta con la segunda. Finalmente, los datos obtenidos se almacenan en una lista de tipo `ArrayList`.

Una vez alimentado el comboBox se procede definir el método que, de la misma manera, define la sentencia, ejecuta y presenta los resultados para obtener los nombres y apellidos de los atletas de una universidad que pertenece a la comunidad seleccionada por el usuario.

Este método se ha denominado `consultaB1()` y recibe como argumentos la conexión establecida y la comunidad seleccionada por el usuario.

```
142 public void consultaB1(Connection con, String comunidad) {
143     // Se define la sentencia
144     String sentenciaSQL = "SELECT a.nombre, a.apellidos, u.nombre as nomUniv "
145         + "FROM atleta a, universidad u "
146         + "WHERE a.universidad = u.codigo "
147         + "AND u.comunidad = ?";
148     // Se declara e instancia una lista que recoge una lista de resultados
149     ArrayList<String> listaResultados = new ArrayList<>();
150
151     // Se realiza la consulta y se almacenan los resultados
152     try (PreparedStatement consulta = con.prepareStatement(sentenciaSQL);) {
153         consulta.setString(1, comunidad);
154         ResultSet resultados = consulta.executeQuery();
155
156         while (resultados.next()) {
157             String nombreA = resultados.getString("nombre");
158             String apellidosA = resultados.getString("apellidos");
159             String nombreU = resultados.getString("nomUniv");
160             String resultado = String.format("El atleta %s %s pertenece a la %s.", nombreA, apellidosA, nombreU);
161             System.out.println(x: resultado);
162             listaResultados.add(e: resultado);
163         }
164
165         // Se muestran los resultados en la aplicación swing
166         mostrarListaJLabel(listaResultados, jLabelJlConsultaB1);
167         jlConsultaB1.setForeground(Color.BLUE);
168         // Se cierra el objeto ResultSet
169         resultados.close();
170
171     } catch (SQLException ex) {
172         System.out.println(x: ex.getMessage());
173         System.err.printf(format: "Se ha producido un error al ejecutar la consulta SQL.");
174     }
175 }
```

Find: comunidadesValoresUnicos 2 matches

antonio\_naranjo\_ad2\_e1.ManejadorBDJFrame > consultaB1

Output - Antonio\_Naranjo\_AD2\_E1 (run)

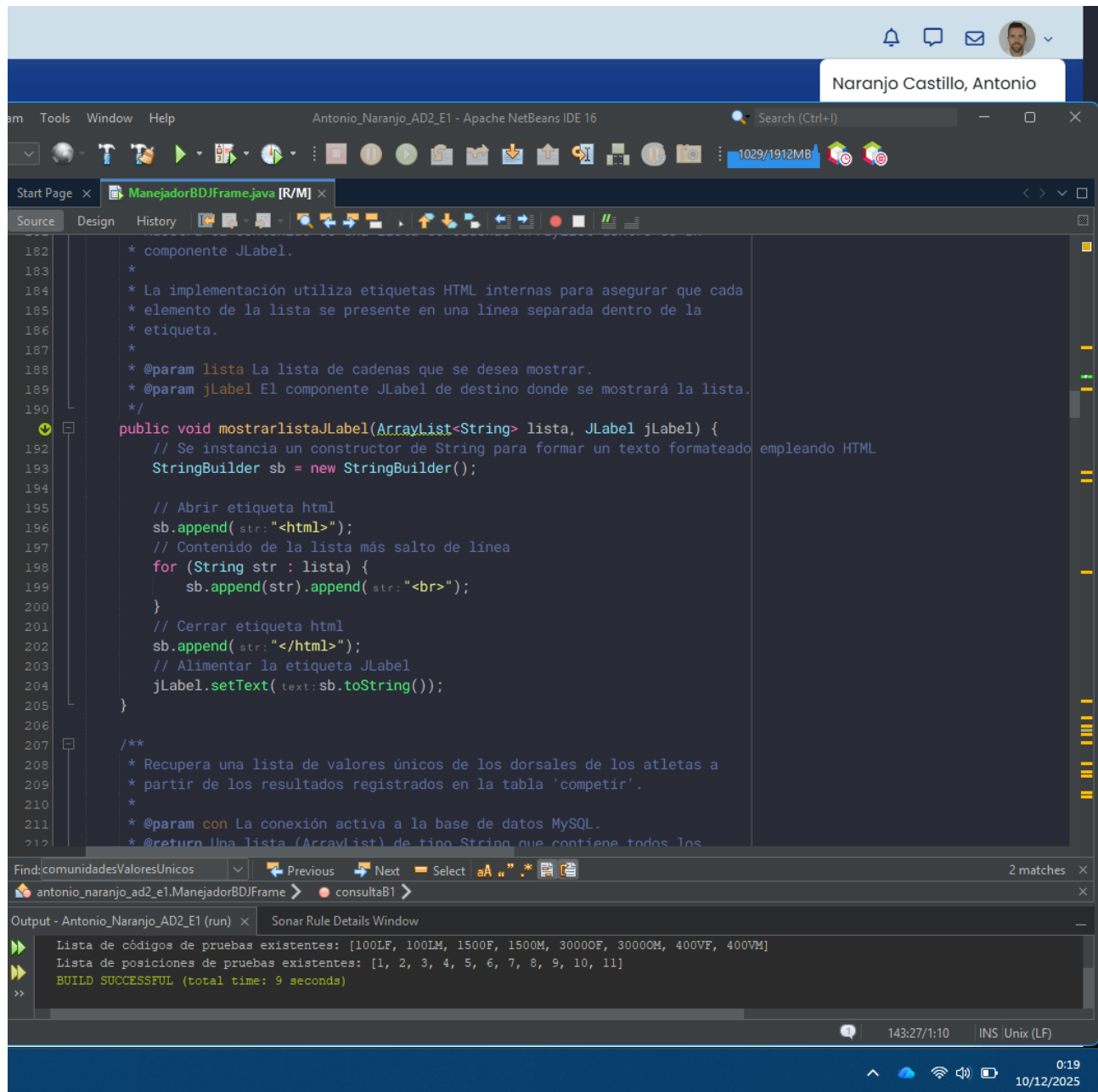
```
Lista de códigos de pruebas existentes: [100LF, 100LM, 1500F, 1500M, 3000OF, 3000OM, 400VF, 400VM]
Lista de posiciones de pruebas existentes: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
BUILD SUCCESSFUL (total time: 9 seconds)
```

143:27/1:10 INS Unix (LF) 0:12 10/12/2025

En este método se utiliza la clase `PreparedStatement` en lugar de la clase `Statement`, porque la sentencia SQL requiere de un parámetro de entrada para su definición, representado en la imagen anterior como un símbolo de interrogación.

Si la consulta se ejecuta con éxito se recogen los resultados mediante un objeto de la clase `ResultSet`, y mediante un bucle tipo `while`, mientras existan resultados que mostrar éstos se almacenarán en una lista que posteriormente se formateará para mostrarla en una etiqueta de la interfaz gráfica. Por último, se cierra el objeto resultados de la clase `ResultSet` puesto que se encuentra fuera de la condición del bloque `try-catch`.

Para el formateo de los resultados se define un método denominado `mostrarListaJLabel()` que recibe como argumentos la lista donde se han almacenado los resultados de la consulta SQL anterior y la etiqueta donde se requiere que se muestren los datos formateados.



```
182  * componente JLabel.
183  *
184  * La implementación utiliza etiquetas HTML internas para asegurar que cada
185  * elemento de la lista se presente en una línea separada dentro de la
186  * etiqueta.
187  *
188  * @param lista La lista de cadenas que se desea mostrar.
189  * @param jLabel El componente JLabel de destino donde se mostrará la lista.
190  */
191  public void mostrarListaJLabel(ArrayList<String> lista, JLabel jLabel) {
192      // Se instancia un constructor de String para formar un texto formateado empleando HTML
193      StringBuilder sb = new StringBuilder();
194
195      // Abrir etiqueta html
196      sb.append(str:"<html>");
197      // Contenido de la lista más salto de línea
198      for (String str : lista) {
199          sb.append(str).append(str:"<br>");
200      }
201      // Cerrar etiqueta html
202      sb.append(str:"</html>");
203      // Alimentar la etiqueta JLabel
204      jLabel.setText(text: sb.toString());
205  }
206
207  /**
208  * Recupera una lista de valores únicos de los dorsales de los atletas a
209  * partir de los resultados registrados en la tabla 'competir'.
210  *
211  * @param con La conexión activa a la base de datos MySQL.
212  * @return Una lista (ArrayList) de tipo String que contiene todos los
```

Find:comunidadesValoresUnicos 2 matches

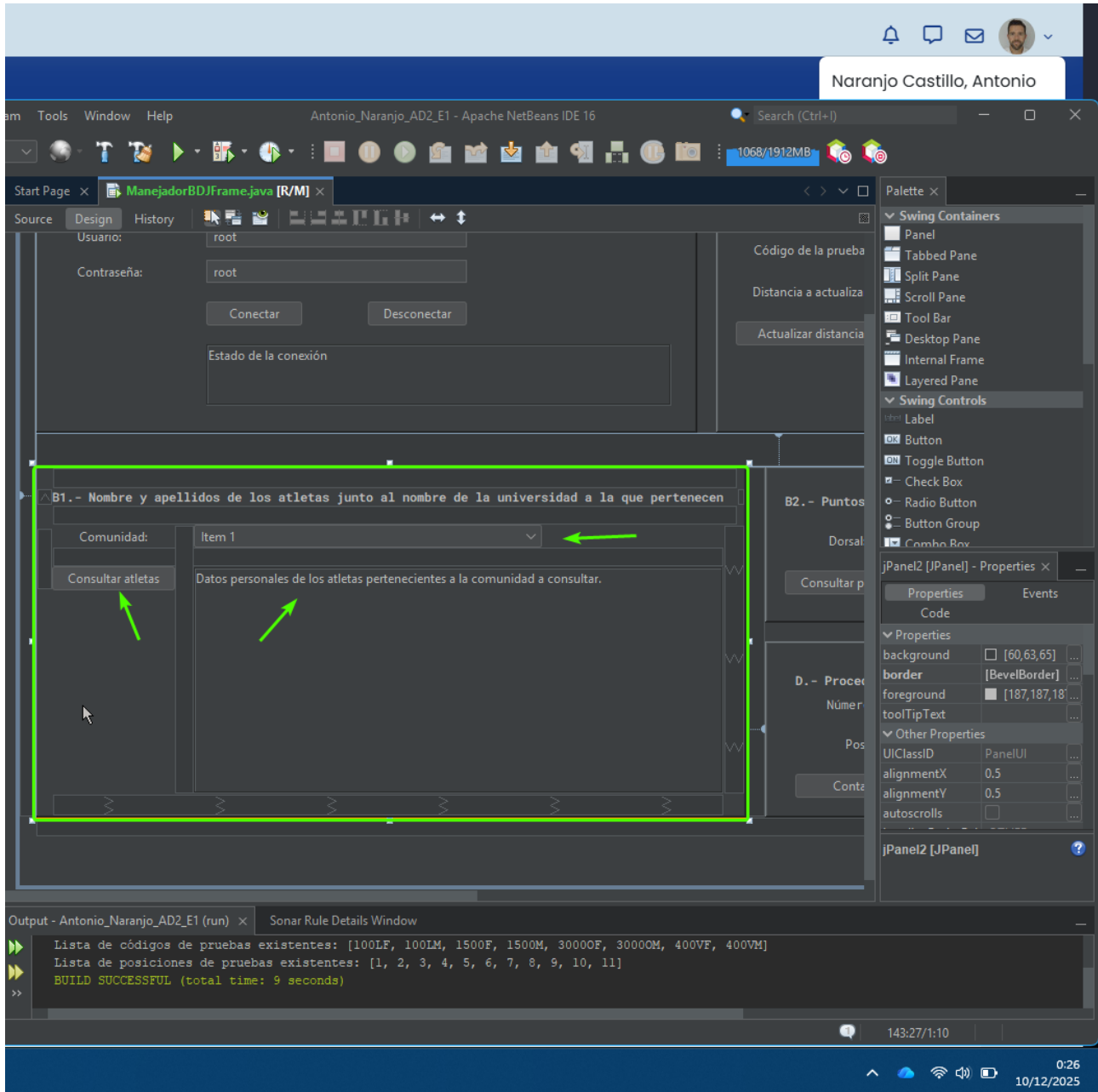
Output - Antonio\_Naranjo\_AD2\_E1 (run) x Sonar Rule Details Window

```
>> Lista de códigos de pruebas existentes: [100LF, 100LM, 1500F, 1500M, 30000F, 30000M, 400VF, 400VM]
>> Lista de posiciones de pruebas existentes: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
>> BUILD SUCCESSFUL (total time: 9 seconds)
```

143:27/1:10 INS Unix (LF) 0:19 10/12/2025

Se hace uso de un constructor de cadenas `StringBuilder` para formatear los datos de la lista ayudado de código HTML.

En la interfaz gráfica Swing, se asigna un panel para la consulta en cuestión, completado por un comboBox que recoge los valores únicos de las comunidades, un botón que ejecuta el método anterior consultaB1() y una etiqueta que presenta los resultados obtenidos de dicha consulta.

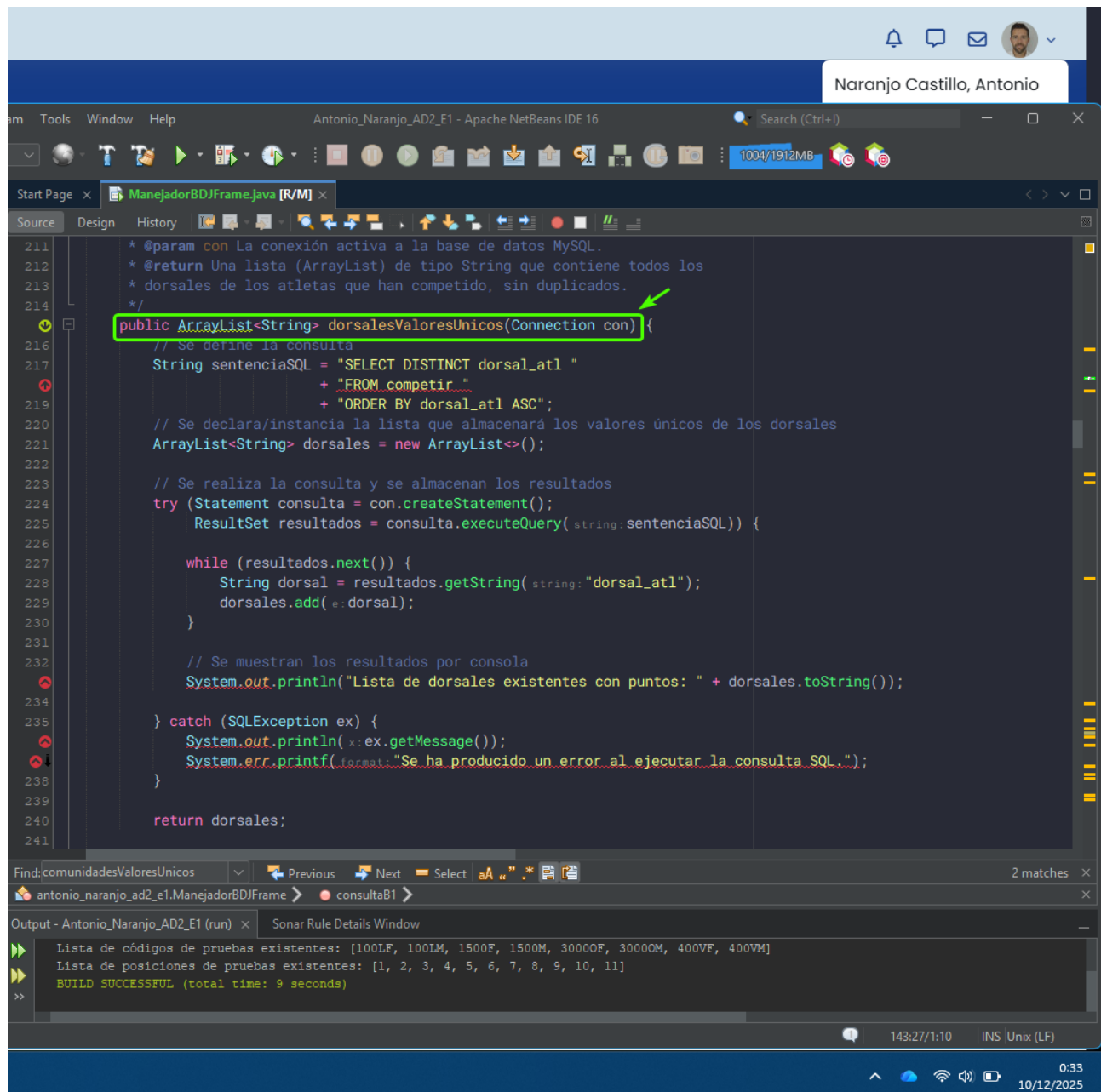




## 5. Consulta B2. Puntos totales por dorsal

De manera similar a la anterior, se prepara los datos a listar por un comboBox, valores únicos de los números de dorsales ordenados de manera ascendente.

Para ello se ejecuta la sentencia SQL necesaria y se almacenan los datos obtenidos en una nueva lista.



```
211  * @param con La conexión activa a la base de datos MySQL.
212  * @return Una lista (ArrayList) de tipo String que contiene todos los
213  * dorsales de los atletas que han competido, sin duplicados.
214  */
215  public ArrayList<String> dorsalesValoresUnicos(Connection con) {
216      // Se define la consulta
217      String sentenciaSQL = "SELECT DISTINCT dorsal_atl "
218                          + "FROM competir_"
219                          + "ORDER BY dorsal_atl ASC";
220      // Se declara/instancia la lista que almacenará los valores únicos de los dorsales
221      ArrayList<String> dorsales = new ArrayList<>();
222
223      // Se realiza la consulta y se almacenan los resultados
224      try (Statement consulta = con.createStatement();
225           ResultSet resultados = consulta.executeQuery(string: sentenciaSQL)) {
226
227          while (resultados.next()) {
228              String dorsal = resultados.getString(string: "dorsal_atl");
229              dorsales.add(dorsal);
230          }
231
232          // Se muestran los resultados por consola
233          System.out.println("Lista de dorsales existentes con puntos: " + dorsales.toString());
234
235      } catch (SQLException ex) {
236          System.out.println("Error: " + ex.getMessage());
237          System.err.printf(format: "Se ha producido un error al ejecutar la consulta SQL.");
238      }
239
240      return dorsales;
241  }
```

Find: comunidadesValoresUnicos 2 matches

Output - Antonio\_Naranjo\_AD2\_E1 (run) x Sonar Rule Details Window

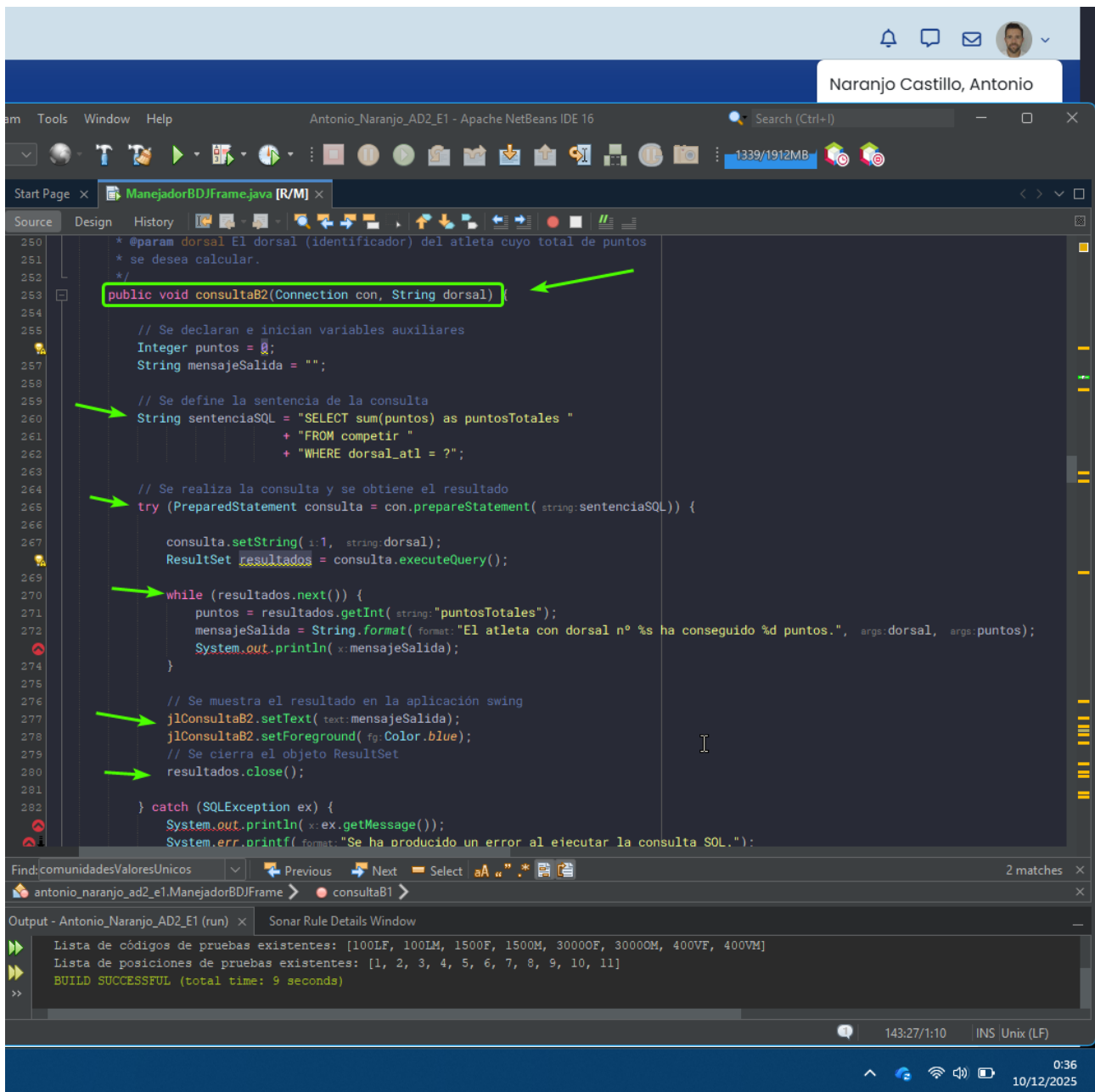
```
>> Lista de códigos de pruebas existentes: [100LF, 100LM, 1500F, 1500M, 3000OF, 3000OM, 400VF, 400VM]
>> Lista de posiciones de pruebas existentes: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
>> BUILD SUCCESSFUL (total time: 9 seconds)
```

143:27/1:10 INS Unix (LF) 0:33 10/12/2025

Al igual que anteriormente, se hacen uso de las clases Statement para ejecutar la sentencia y ResultSet para almacenar los datos obtenidos. En este caso no es necesario cerrar el objeto ResultSet porque se encuentra dentro de la condición del bloque try-catch.



Se implementa el método `consultaB2()` que recibe como argumento el objeto `Connection` y un `String` dorsal seleccionado por el usuario de la aplicación por medio del `comboBox`.



```
250  * @param dorsal El dorsal (identificador) del atleta cuyo total de puntos
251  * se desea calcular.
252  */
253  public void consultaB2(Connection con, String dorsal) {
254
255      // Se declaran e inician variables auxiliares
256      Integer puntos = 0;
257      String mensajeSalida = "";
258
259      // Se define la sentencia de la consulta
260      String sentenciaSQL = "SELECT sum(puntos) as puntosTotales "
261          + "FROM competir "
262          + "WHERE dorsal_atl = ?";
263
264      // Se realiza la consulta y se obtiene el resultado
265      try (PreparedStatement consulta = con.prepareStatement(sentenciaSQL)) {
266
267          consulta.setString(1, dorsal);
268          ResultSet resultados = consulta.executeQuery();
269
270          while (resultados.next()) {
271              puntos = resultados.getInt("puntosTotales");
272              mensajeSalida = String.format("El atleta con dorsal nº %s ha conseguido %d puntos.", dorsal, puntos);
273              System.out.println(mensajeSalida);
274          }
275
276          // Se muestra el resultado en la aplicación swing
277          jlConsultaB2.setText(mensajeSalida);
278          jlConsultaB2.setForeground(Color.BLUE);
279          // Se cierra el objeto ResultSet
280          resultados.close();
281
282      } catch (SQLException ex) {
283          System.out.println(ex.getMessage());
284          System.err.printf("Se ha producido un error al ejecutar la consulta SQL.");
285      }
286  }
```

Find: comunidadesValoresUnicos 2 matches

antonio\_naranjo\_ad2\_e1.ManejadorBDJFrame > consultaB1 >

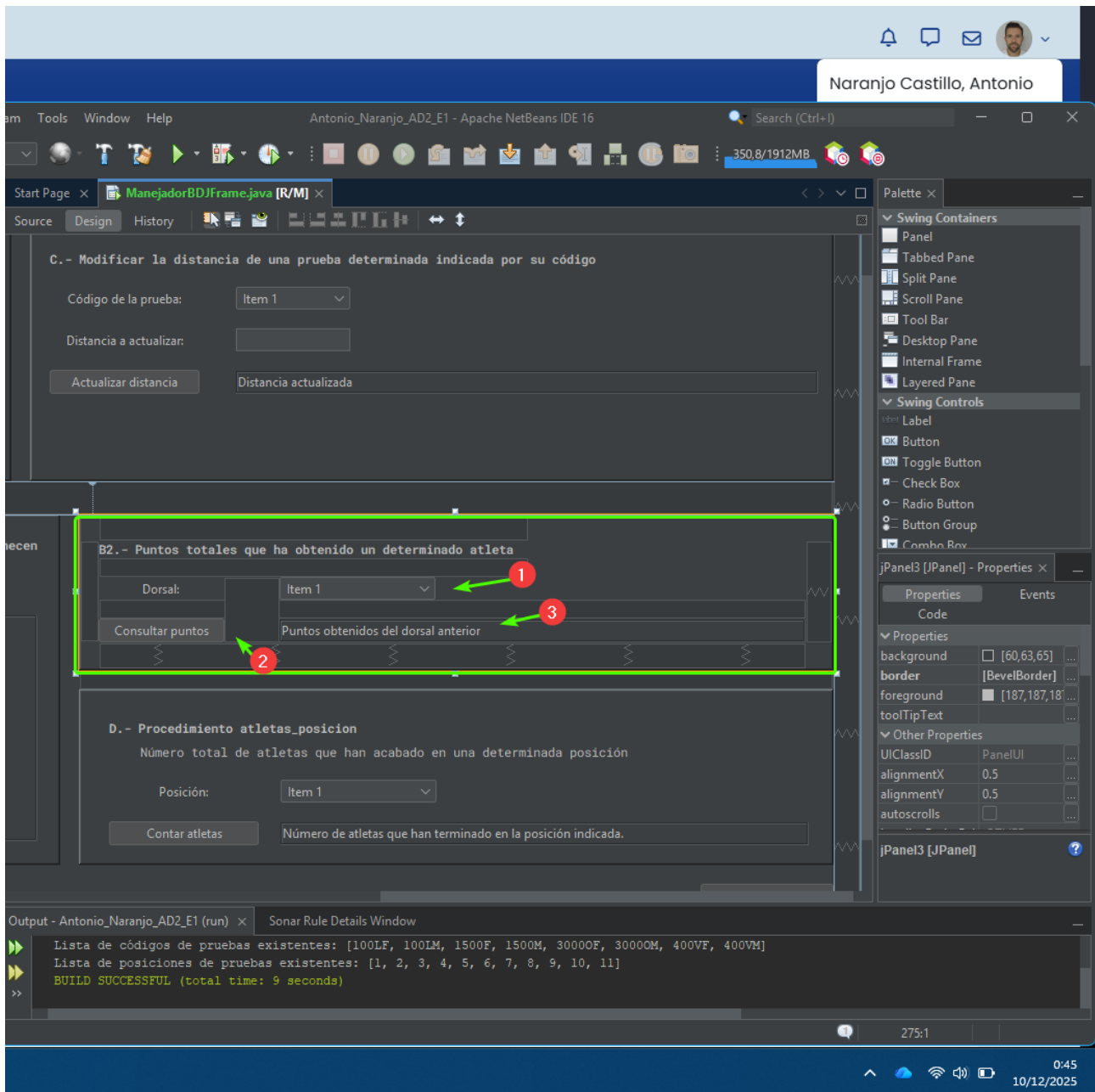
Output - Antonio\_Naranjo\_AD2\_E1 (run) x Sonar Rule Details Window

```
>> Lista de códigos de pruebas existentes: [100LF, 100LM, 1500F, 1500M, 3000OF, 3000OM, 400VF, 400VM]
>> Lista de posiciones de pruebas existentes: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
>> BUILD SUCCESSFUL (total time: 9 seconds)
```

143:27/1:10 INS Unix (LF) 0:36 10/12/2025

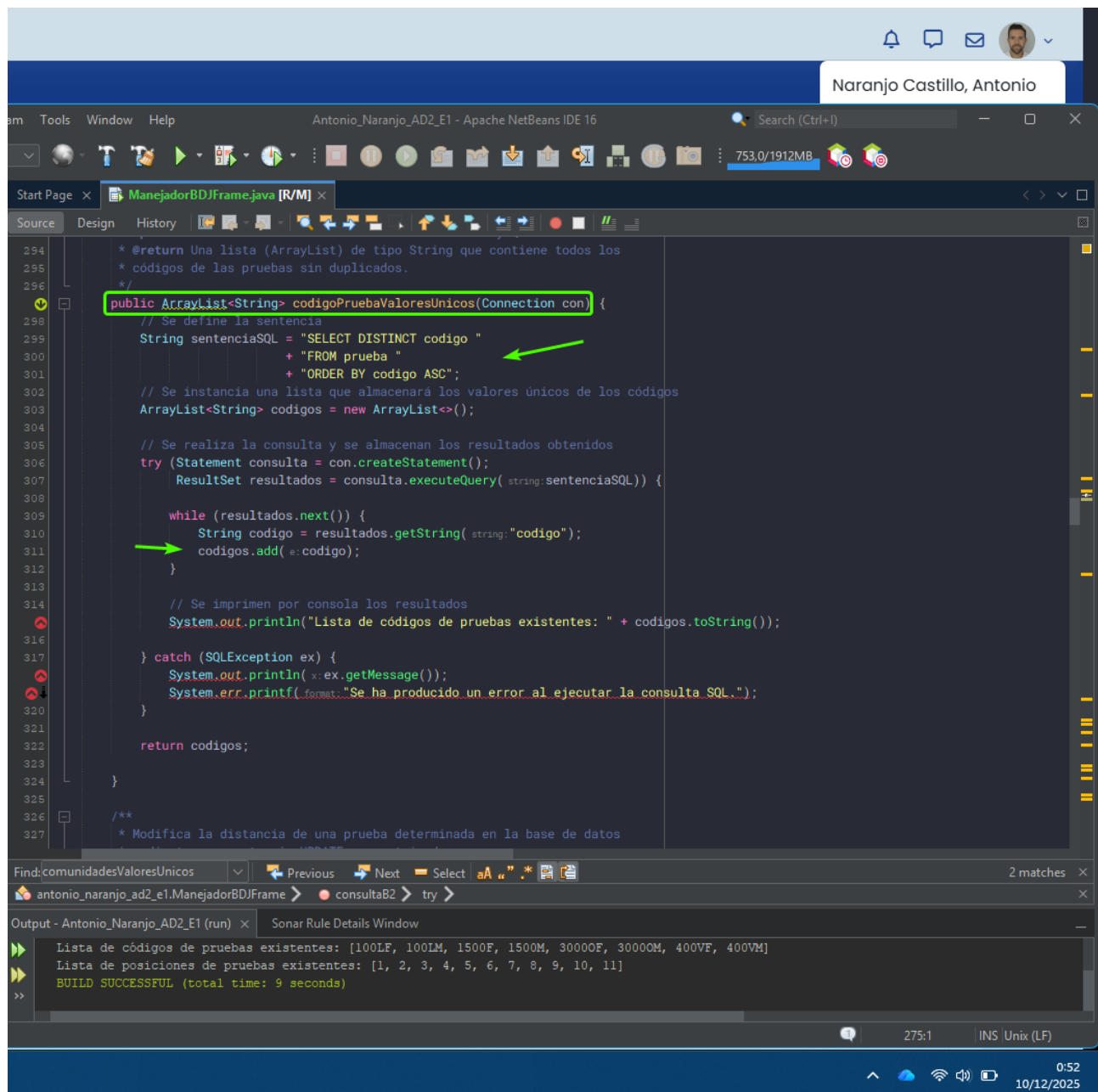
Se ejecuta la sentencia pasando como argumento el `String` dorsal anterior. Todo de manera similar al método `consulta` anterior, pero en esta ocasión solo se obtiene un resultado por motivo de implementarse en la sentencia la función `SUM` que solo devolverá una fila. El resultado obtenido se formatea y se muestra en una etiqueta de la interfaz gráfica.

En la interfaz gráfica se incorpora un nuevo panel para alojar en su interior el comboBox que recibe los valores únicos de los dorsales, un botón que ejecutará el método consultaB2() y una etiqueta que mostrará el resultado de la consulta, es decir, los puntos totales del dorsal seleccionado.



## 6. Actualización C. Modificación de la distancia de una prueba

De manera similar a casos anteriores, se alimenta un comboBox con una lista de valores únicos de códigos de prueba. Cabe decir que, el único atributo “unique” es el nombre de las universidades, por ello, las listas que alimentan los comboBox siempre se realiza una consulta de valores únicos y ordenados de manera ascendente. Más adelante se mostrará un anexo de imágenes aclaratorias y consultas realizadas en el cliente mysql en línea de comandos. También hay que destacar la importancia de fijar los datos a seleccionar por el usuario de la aplicación en un comboBox, descontando gran cantidad de posibles excepciones que deberían implementarse en caso de recibir los datos del usuario de la aplicación por medio de un campo de texto.



```
294  * @return Una lista (ArrayList) de tipo String que contiene todos los
295  * códigos de las pruebas sin duplicados.
296  */
297  public ArrayList<String> codigoPruebaValoresUnicos(Connection con) {
298      // Se define la sentencia
299      String sentenciaSQL = "SELECT DISTINCT codigo "
300                          + "FROM prueba "
301                          + "ORDER BY codigo ASC";
302      // Se instancia una lista que almacenará los valores únicos de los códigos
303      ArrayList<String> codigos = new ArrayList<>();
304
305      // Se realiza la consulta y se almacenan los resultados obtenidos
306      try (Statement consulta = con.createStatement();
307           ResultSet resultados = consulta.executeQuery( String.valueOf(sentenciaSQL))) {
308
309          while (resultados.next()) {
310              String codigo = resultados.getString( String.valueOf("codigo"));
311              codigos.add( String.valueOf(codigo));
312          }
313
314          // Se imprimen por consola los resultados
315          System.out.println("Lista de códigos de pruebas existentes: " + codigos.toString());
316
317      } catch (SQLException ex) {
318          System.out.println( String.valueOf(ex.getMessage()));
319          System.err.printf( String.valueOf("Se ha producido un error al ejecutar la consulta SQL."));
320      }
321
322      return codigos;
323  }
324
325  /**
326   * Modifica la distancia de una prueba determinada en la base de datos
327   */
```

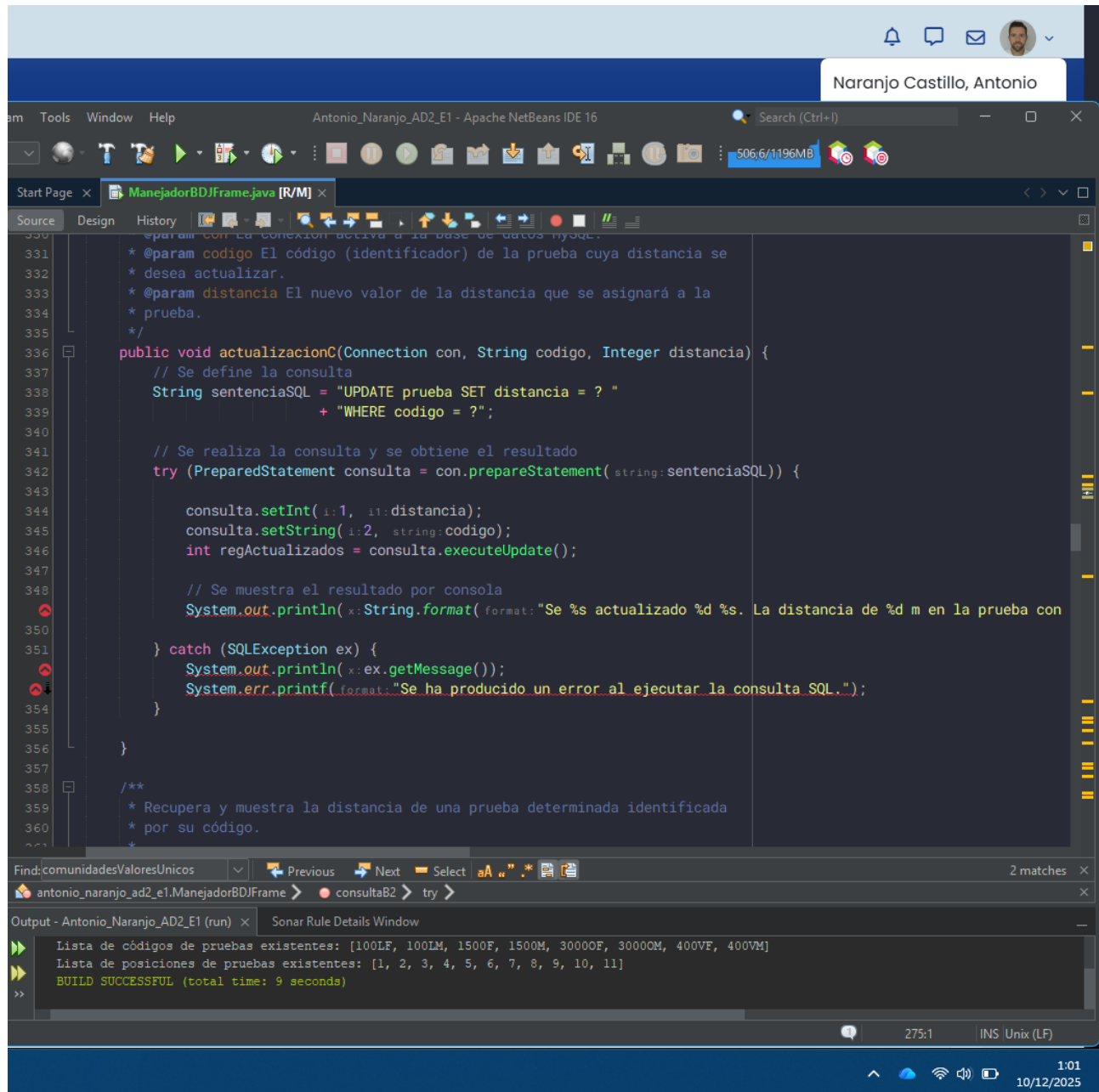
Find:comunidadesValoresUnicos 2 matches

Output - Antonio\_Naranjo\_AD2\_E1 (run)

```
Lista de códigos de pruebas existentes: [100LF, 100LM, 1500F, 1500M, 30000F, 30000M, 400VF, 400VM]
Lista de posiciones de pruebas existentes: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
BUILD SUCCESSFUL (total time: 9 seconds)
```

Como en casos anterior, finalmente los datos obtenidos se almacenan en la lista.

Se implementa el método `actualizacionC` que recibe como argumento la conexión, objeto con de la clase `Connection`, el código seleccionado por el usuario por medio del `comboBox` y la distancia a modificar o actualizar a través de un campo de texto o `TextField`.



```
331  * @param codigo El código (identificador) de la prueba cuya distancia se
332  * desea actualizar.
333  * @param distancia El nuevo valor de la distancia que se asignará a la
334  * prueba.
335  */
336  public void actualizacionC(Connection con, String codigo, Integer distancia) {
337      // Se define la consulta
338      String sentenciaSQL = "UPDATE prueba SET distancia = ? "
339                          + "WHERE codigo = ?";
340
341      // Se realiza la consulta y se obtiene el resultado
342      try (PreparedStatement consulta = con.prepareStatement( string: sentenciaSQL)) {
343
344          consulta.setInt( 1:1, 11:distancia);
345          consulta.setString( 1:2, string:codigo);
346          int regActualizados = consulta.executeUpdate();
347
348          // Se muestra el resultado por consola
349          System.out.println( x:String.format( format:"Se %s actualizado %d %s. La distancia de %d m en la prueba con
350
351      } catch (SQLException ex) {
352          System.out.println( x:ex.getMessage());
353          System.err.printf( format:"Se ha producido un error al ejecutar la consulta SQL.");
354      }
355
356  }
357
358  /**
359   * Recupera y muestra la distancia de una prueba determinada identificada
360   * por su código.
361   */
```

Find:comunidadesValoresUnicos 2 matches

antonio\_naranjo\_ad2\_e1.ManejadorBDJFrame > consultaB2 > try >

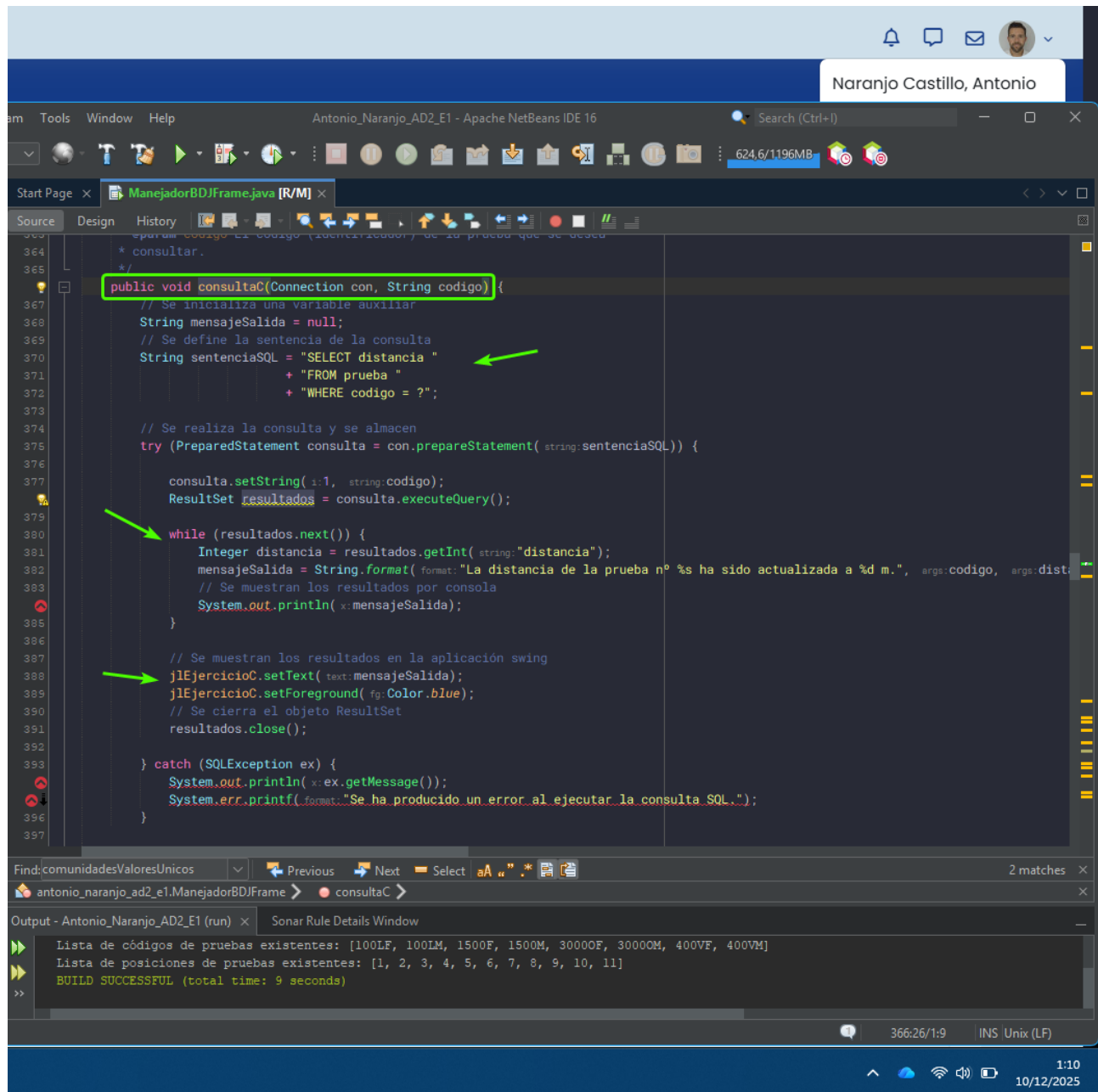
Output - Antonio\_Naranjo\_AD2\_E1 (run) > Sonar Rule Details Window

```
>> Lista de códigos de pruebas existentes: [100LF, 100LM, 1500F, 1500M, 30000F, 30000M, 400VF, 400VM]
>> Lista de posiciones de pruebas existentes: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
>> BUILD SUCCESSFUL (total time: 9 seconds)
```

275:1 INS Unix (LF) 1:01 10/12/2025

En esta ocasión el método recoge los registros actualizados y los muestra en consola.

Una vez ejecutado el código del método anterior, se procede a realizar una consulta SQL implementando el método `consultaC()` que, de manera similar a anteriores casos, recibe como parámetros el objeto `Connection`, y un `String` código.



```
364 * consultar.
365 */
366 public void consultaC(Connection con, String codigo) {
367     // Se inicializa una variable auxiliar
368     String mensajeSalida = null;
369     // Se define la sentencia de la consulta
370     String sentenciaSQL = "SELECT distancia "
371         + "FROM prueba "
372         + "WHERE codigo = ?";
373
374     // Se realiza la consulta y se almacena
375     try (PreparedStatement consulta = con.prepareStatement(sentenciaSQL)) {
376
377         consulta.setString(1, codigo);
378         ResultSet resultados = consulta.executeQuery();
379
380         while (resultados.next()) {
381             Integer distancia = resultados.getInt("distancia");
382             mensajeSalida = String.format("La distancia de la prueba nº %s ha sido actualizada a %d m.", args.codigo, args.distancia);
383             // Se muestran los resultados por consola
384             System.out.println(x:mensajeSalida);
385         }
386
387         // Se muestran los resultados en la aplicación swing
388         jlEjercicioC.setText(text:mensajeSalida);
389         jlEjercicioC.setForeground(fg:Color.blue);
390         // Se cierra el objeto ResultSet
391         resultados.close();
392
393     } catch (SQLException ex) {
394         System.out.println(x:ex.getMessage());
395         System.err.printf(format:"Se ha producido un error al ejecutar la consulta SQL.");
396     }
397 }
```

Find:comunidadesValoresUnicos 2 matches

antonio\_naranjo\_ad2\_e1.ManejadorBDJFrame > consultaC >

Output - Antonio\_Naranjo\_AD2\_E1 (run) x Sonar Rule Details Window

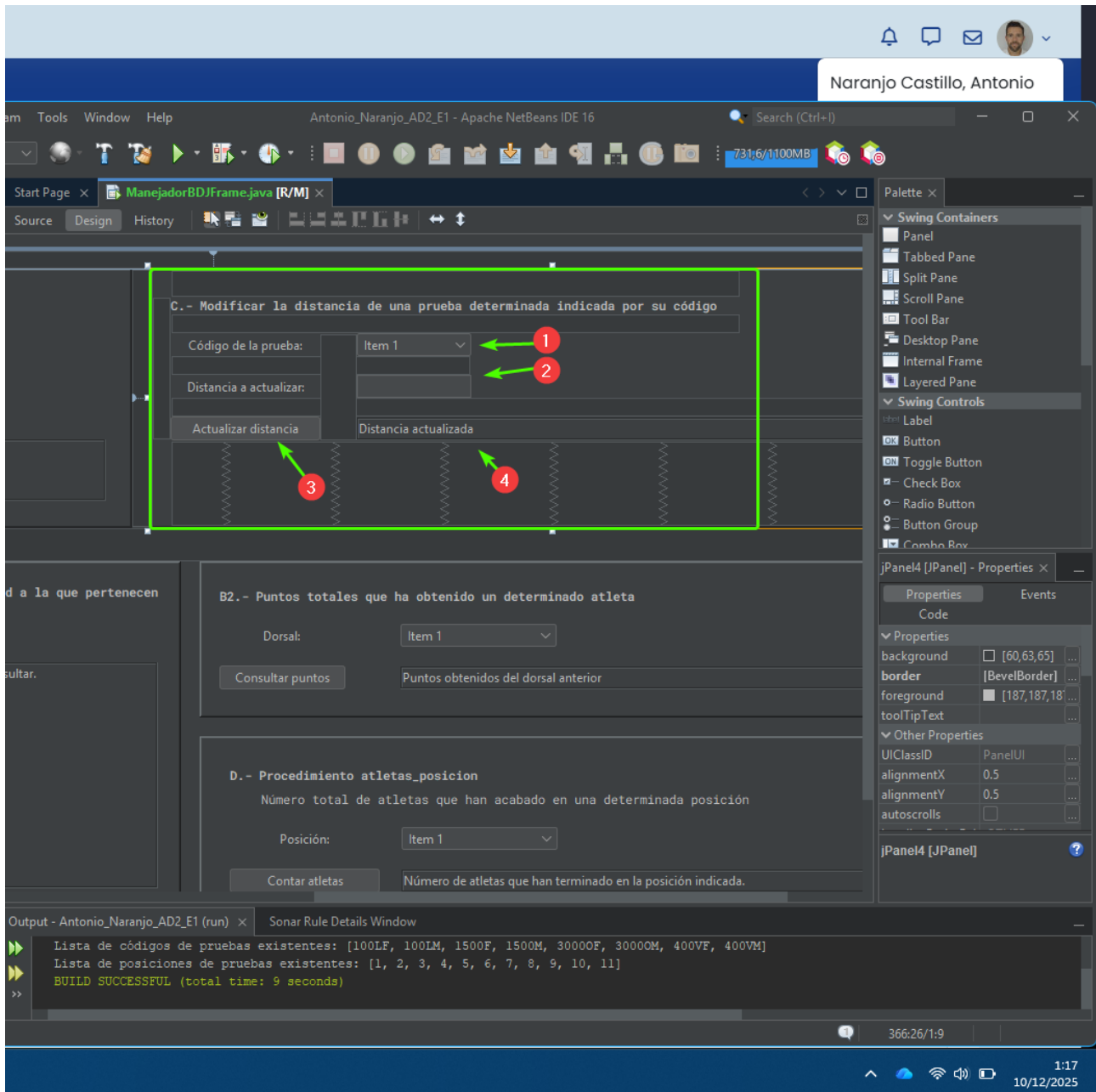
```
>> Lista de códigos de pruebas existentes: [100LF, 100LM, 1500F, 1500M, 30000F, 30000M, 400VF, 400VM]
>> Lista de posiciones de pruebas existentes: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
>> BUILD SUCCESSFUL (total time: 9 seconds)
```

366:26/1:9 INS Unix (LF) 1:10 10/12/2025

Se ejecuta la consulta que determina la nueva distancia que ha sido actualizada para una determinada prueba seleccionada a través de su clave primaria código.

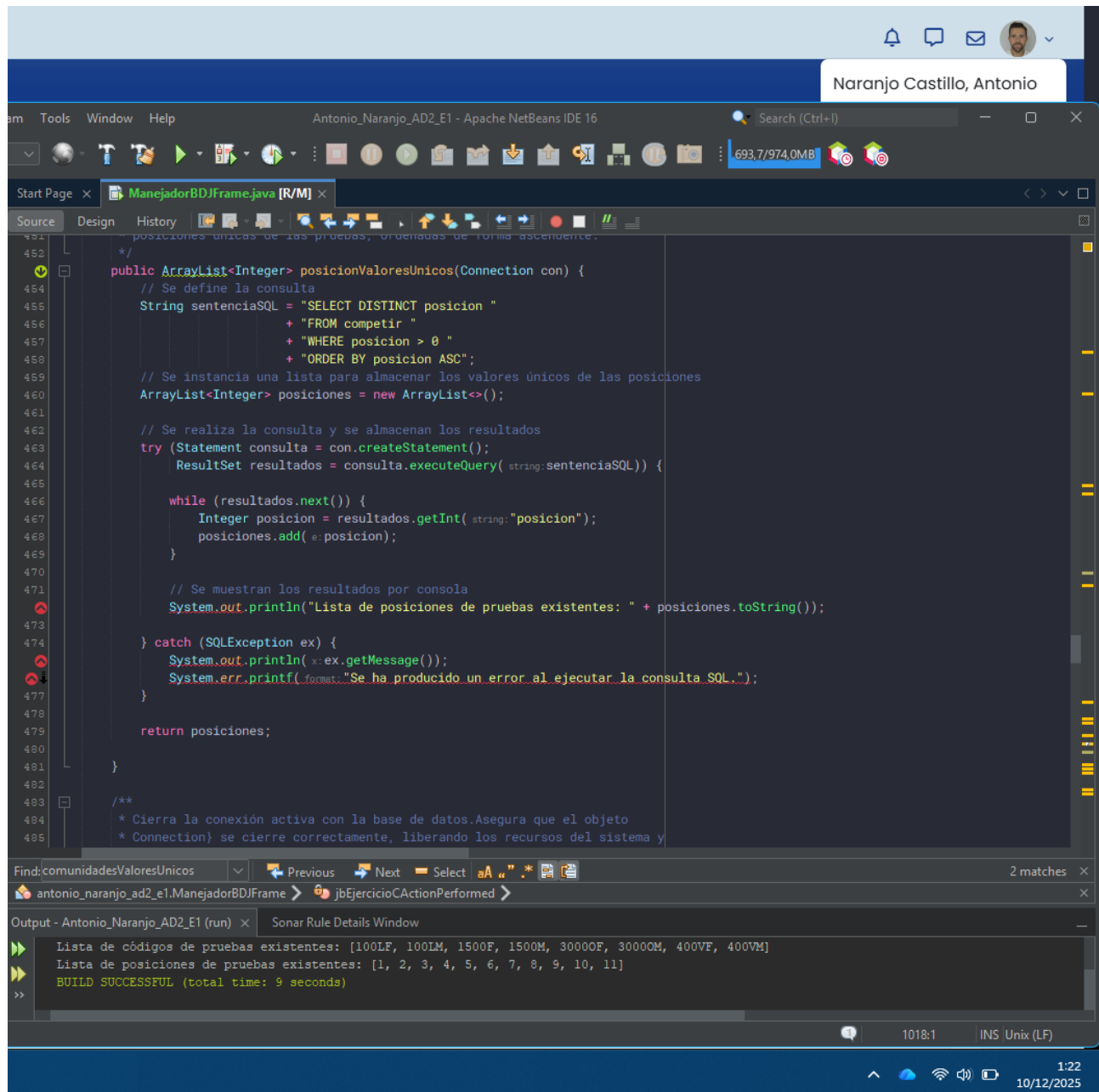
Se recogen los datos obtenidos, en este caso una fila, y se muestra en la aplicación Swing, por medio de una etiqueta. Finalmente se cierra el objeto `ResultSet`.

En cuanto a la interfaz gráfica, se introduce un nuevo panel que contiene un comboBox para recibir los valores de los códigos de las distintas pruebas, un TextField para recibir la distancia a modificar requerida por el usuario de la aplicación, un botón para ejecutar tanto el método actualizaciónC() como el método consultaC(), y por último, una etiqueta que mostrará el resultado de la actualización.



## 7. Procedimiento D. Ejecución de Procedimiento Almacenado

Como en anteriores ocasiones, se emplea un comboBox para recoger los datos de una lista que contiene los valores únicos ordenada ascendente de las posiciones de todas las pruebas.



```
451  // posiciones únicas de las pruebas, ordenadas de forma ascendente.
452
453  public ArrayList<Integer> posicionValoresUnicos(Connection con) {
454      // Se define la consulta
455      String sentenciaSQL = "SELECT DISTINCT posicion "
456                          + "FROM competir "
457                          + "WHERE posicion > 0 "
458                          + "ORDER BY posicion ASC";
459      // Se instancia una lista para almacenar los valores únicos de las posiciones
460      ArrayList<Integer> posiciones = new ArrayList<>();
461
462      // Se realiza la consulta y se almacenan los resultados
463      try (Statement consulta = con.createStatement()) {
464          ResultSet resultados = consulta.executeQuery( string: sentenciaSQL) {
465
466              while (resultados.next()) {
467                  Integer posicion = resultados.getInt( string:"posicion");
468                  posiciones.add( e:posicion);
469              }
470
471              // Se muestran los resultados por consola
472              System.out.println("Lista de posiciones de pruebas existentes: " + posiciones.toString());
473
474          } catch (SQLException ex) {
475              System.out.println( x:ex.getMessage());
476              System.err.printf( format:"Se ha producido un error al ejecutar la consulta SQL.");
477          }
478
479          return posiciones;
480      }
481  }
482
483  /**
484   * Cierra la conexión activa con la base de datos. Asegura que el objeto
485   * Connection se cierre correctamente, liberando los recursos del sistema y
```

Find: comunidadesValoresUnicos 2 matches

antonio\_naranjo\_ad2\_e1.ManejadorBDJFrame > jbEjercicioCActionPerformed >

Output - Antonio\_Naranjo\_AD2\_E1 (run) x Sonar Rule Details Window

```
>> Lista de códigos de pruebas existentes: [100LF, 100IM, 1500F, 1500M, 30000F, 30000M, 400VF, 400VM]
>> Lista de posiciones de pruebas existentes: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
>> BUILD SUCCESSFUL (total time: 9 seconds)
```

1018:1 INS Unix (LF) 1:22 10/12/2025

La posición será seleccionada por el usuario de la aplicación, accediendo a las posiciones por medio del mencionado comboBox.



Se implementa el método `procedimientoD()` que recibe como argumentos el objeto `Connection` y un objeto `String` `posicion`.

En esta ocasión se almacena en un `String` la llamada al procedimiento `atletas_posicion` disponiendo de dos argumentos, uno de entrada `IN` y otro de salida `OUT`.

Se realiza la llamada al procedimiento mediante la clase `CallableStatement`, haciendo uso del método estático `prepareCall` de la clase `Connection` que recibe como parámetro el `String` llamada al procedimiento `atletas_posicion`.

Se establece el parámetro de entrada `IN` con el método `setInt()` de la clase `CallableStatement` y se recibe el parámetro de salida `OUT` con el método `getInt()` de la misma clase. Previamente, se definió el parámetro de salida mediante el método `registerOutParameter()` y se ejecutó el procedimiento con el método `execute()`.

El resultado obtenido se formatea y se muestra tanto en consola como en una etiqueta de la interfaz grafica.

```
408      *  
409      * @param con La conexión activa a la base de datos MySQL.  
410      * @param posicion El valor de la posición utilizada como criterio de  
411      * búsqueda (parámetro IN).  
412      */  
413      public void procedimientoD(Connection con, String posicion) {  
414  
415          // Se declaran variables auxiliares  
416          Integer numAtletas;  
417          String mensajeSalida;  
418          Integer posicionAtleta=Integer.parseInt( posicion);  
419          // Se define la llamada al procedimiento  
420          String llamadaProcedimiento = "{ call atletas_posicion(?, ?) }";  
421  
422          // Se realiza la llamada al procedimiento y se recogen los resultados  
423          try (CallableStatement prcProcedimientoNumAtletas = con.prepareCall( llamadaProcedimiento)) {  
424              // Se ejecuta el procedimiento  
425              prcProcedimientoNumAtletas.setInt( 1, 1, posicionAtleta);  
426              prcProcedimientoNumAtletas.registerOutParameter( 1, 2, java.sql.Types.INTEGER);  
427              prcProcedimientoNumAtletas.execute();  
428              // Se obtiene el resultado  
429              numAtletas=prcProcedimientoNumAtletas.getInt( 1, 2);  
430              // Se muestra el resultado por pantalla  
431              mensajeSalida=String.format( format:"El número de atletas que han acabado en la posición nº %d = %d.", args:posicionAtleta, args  
432              System.out.println( x:mensajeSalida);  
433              // Se muestra el resultado en la aplicación swing  
434              jlProcedimientoD.setText( text:mensajeSalida);  
435              jlProcedimientoD.setForeground( fg:Color.blue);  
436  
437          } catch (SQLException ex) {  
438              System.out.println( x:ex.getMessage());  
439              System.err.printf( format:"Se ha producido un error al ejecutar la consulta SQL.");  
440          }  
441      }
```

Find:comunidadesValoresUnicos 2 matches

antonio\_naranjo\_ad2\_e1.ManejadorBDJFrame jlbEjercicioCActionPerformed

Output - Antonio\_Naranjo\_AD2\_E1 (run) Sonar Rule Details Window

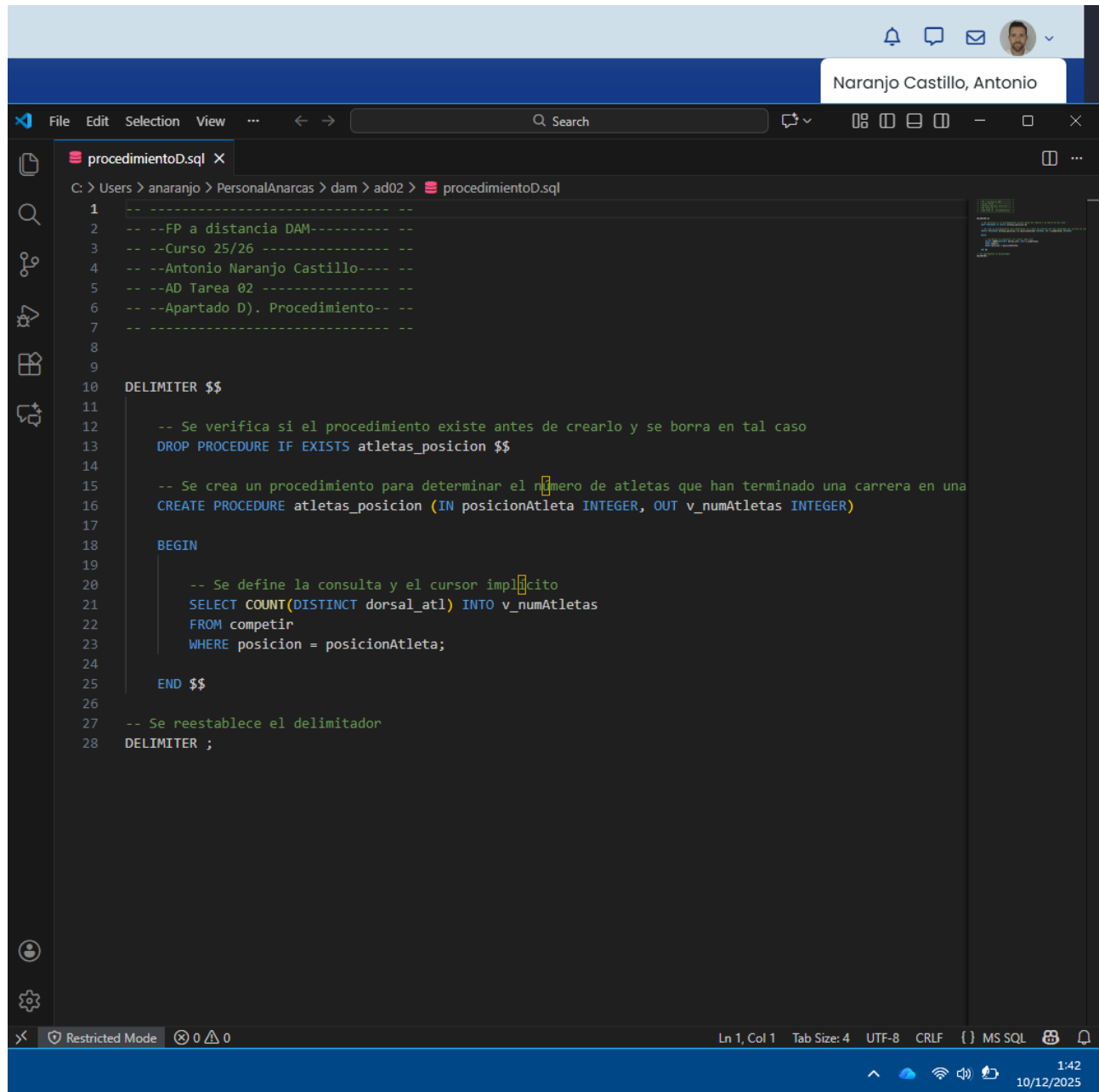
```
>> Lista de códigos de pruebas existentes: [100LF, 100LM, 1500F, 1500M, 3000OF, 3000OM, 400VF, 400VM]  
>> Lista de posiciones de pruebas existentes: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]  
>> BUILD SUCCESSFUL (total time: 9 seconds)
```

1018:1 INS Unix (LF) 1:28 10/12/2025



Finalmente, se recogen las posibles excepciones en caso de producirse un error al ejecutar la sentencia del procedimiento.

Previamente a la ejecución del método anterior, se tuvo que preparar la sentencia SQL del procedimiento atletas\_posicion en un script e incorporarlo a la base de datos campeonato\_atletismo. El código de dicho procedimiento se muestra a continuación.



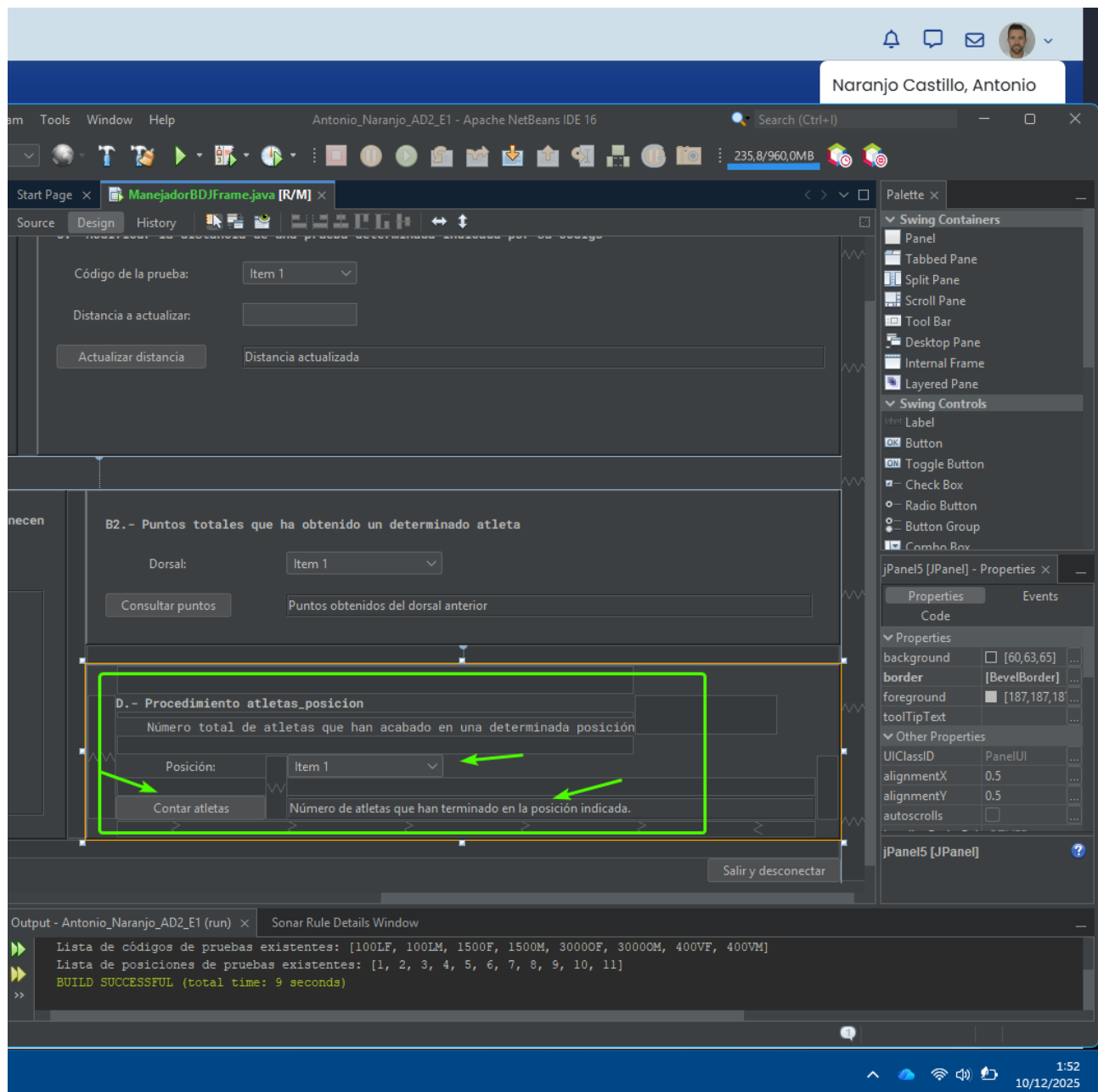
```
1  -----
2  -- FP a distancia DAM-----
3  -- Curso 25/26 -----
4  -- Antonio Naranjo Castillo---
5  -- AD Tarea 02 -----
6  -- Apartado D). Procedimiento--
7  -----
8
9
10 DELIMITER $$
11
12 -- Se verifica si el procedimiento existe antes de crearlo y se borra en tal caso
13 DROP PROCEDURE IF EXISTS atletas_posicion $$
14
15 -- Se crea un procedimiento para determinar el número de atletas que han terminado una carrera en una
16 CREATE PROCEDURE atletas_posicion (IN posicionAtleta INTEGER, OUT v_numAtletas INTEGER)
17
18 BEGIN
19
20     -- Se define la consulta y el cursor implícito
21     SELECT COUNT(DISTINCT dorsal_atl) INTO v_numAtletas
22     FROM competir
23     WHERE posicion = posicionAtleta;
24
25 END $$
26
27 -- Se reestablece el delimitador
28 DELIMITER ;
```

Se incluyen las directivas DELIMITER y la cláusula DROP IF EXISTS para su correcta carga en el cliente mysql.

Se define una consulta que almacena su resultado único en un cursor implícito v\_numAtletas, el cual será el parámetro de salida que recogerá nuestra aplicación tras ejecutar el método procedimiento D()).

Por último, una vez más se establece una interfaz gráfica disponiendo de un panel contenedor para recibir componentes que simplifiquen la aplicación del procedimiento anterior por parte el usuario de la aplicación.

Se dispone de un comboBox que recibe los valores únicos de las posiciones obtenidas en cada una de las pruebas, el usuario seleccionará la posición que corresponda, un botón para ejecutar el procedimiento `atletas_posicion` y una etiqueta que presenta el valor del resultado del tal procedimiento.

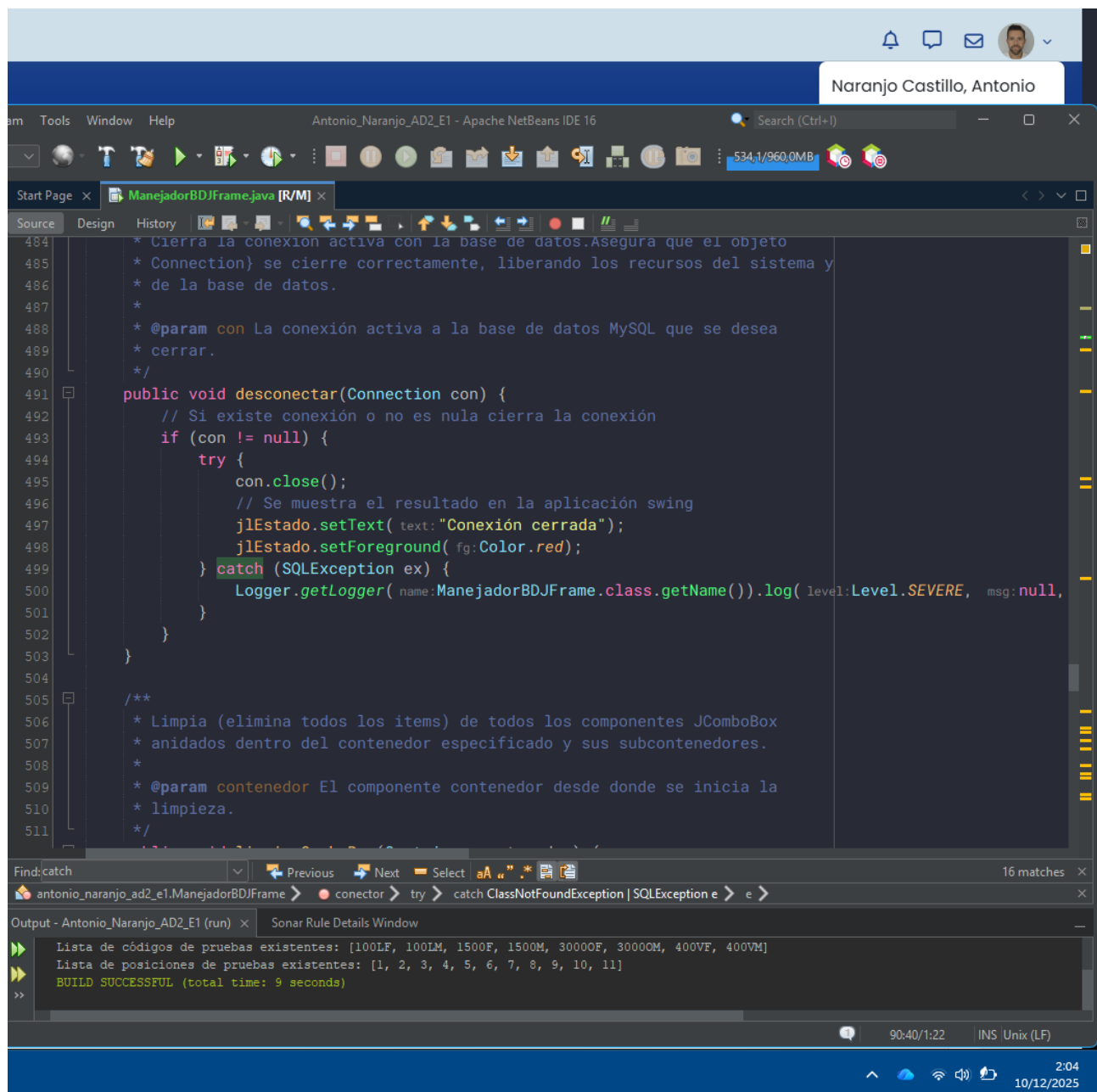


## 8. Gestión de Excepciones y Cierre de Recursos

En cada una de las capturas anteriores se puede visualizar el uso de excepciones para recoger los posibles problemas que se pudieran presentar empleando bloque try-catch para su manejo, del tipo SQLException en caso de errores en la ejecución de las sentencias SQL y ClassNotFoundException para el caso de no encontrar el driver correspondiente en el método conector().

Por otra parte, la correcta liberación de recursos ResultSet siempre y cuando no se encuentren en la condición del bloque try-catch y otros como Statement, PreparedStatement o CallableStatement siempre han estado dentro de dicha condición try-catch por lo que, se cierran de manera automática tras la ejecución del bloque try-catch.

También es necesario cerrar la conexión con la base de datos para liberar recursos, por ello se implementa el siguiente método desconectar().



```
484 * Cierra la conexión activa con la base de datos. Asegura que el objeto
485 * Connection se cierre correctamente, liberando los recursos del sistema y
486 * de la base de datos.
487 *
488 * @param con La conexión activa a la base de datos MySQL que se desea
489 * cerrar.
490 */
491 public void desconectar(Connection con) {
492     // Si existe conexión o no es nula cierra la conexión
493     if (con != null) {
494         try {
495             con.close();
496             // Se muestra el resultado en la aplicación swing
497             jlEstado.setText(text: "Conexión cerrada");
498             jlEstado.setForeground(fg: Color.red);
499         } catch (SQLException ex) {
500             Logger.getLogger(ManejadorBDJFrame.class.getName()).log(Level.SEVERE, null, ex);
501         }
502     }
503 }
504
505 /**
506 * Limpia (elimina todos los items) de todos los componentes JComboBox
507 * anidados dentro del contenedor especificado y sus subcontenedores.
508 *
509 * @param contenedor El componente contenedor desde donde se inicia la
510 * limpieza.
511 */
```

Find: catch 16 matches

antonio\_naranjo\_ad2\_e1.ManejadorBDJFrame > conector > try > catch ClassNotFoundException | SQLException e > e >

Output - Antonio\_Naranjo\_AD2\_E1 (run) x Sonar Rule Details Window

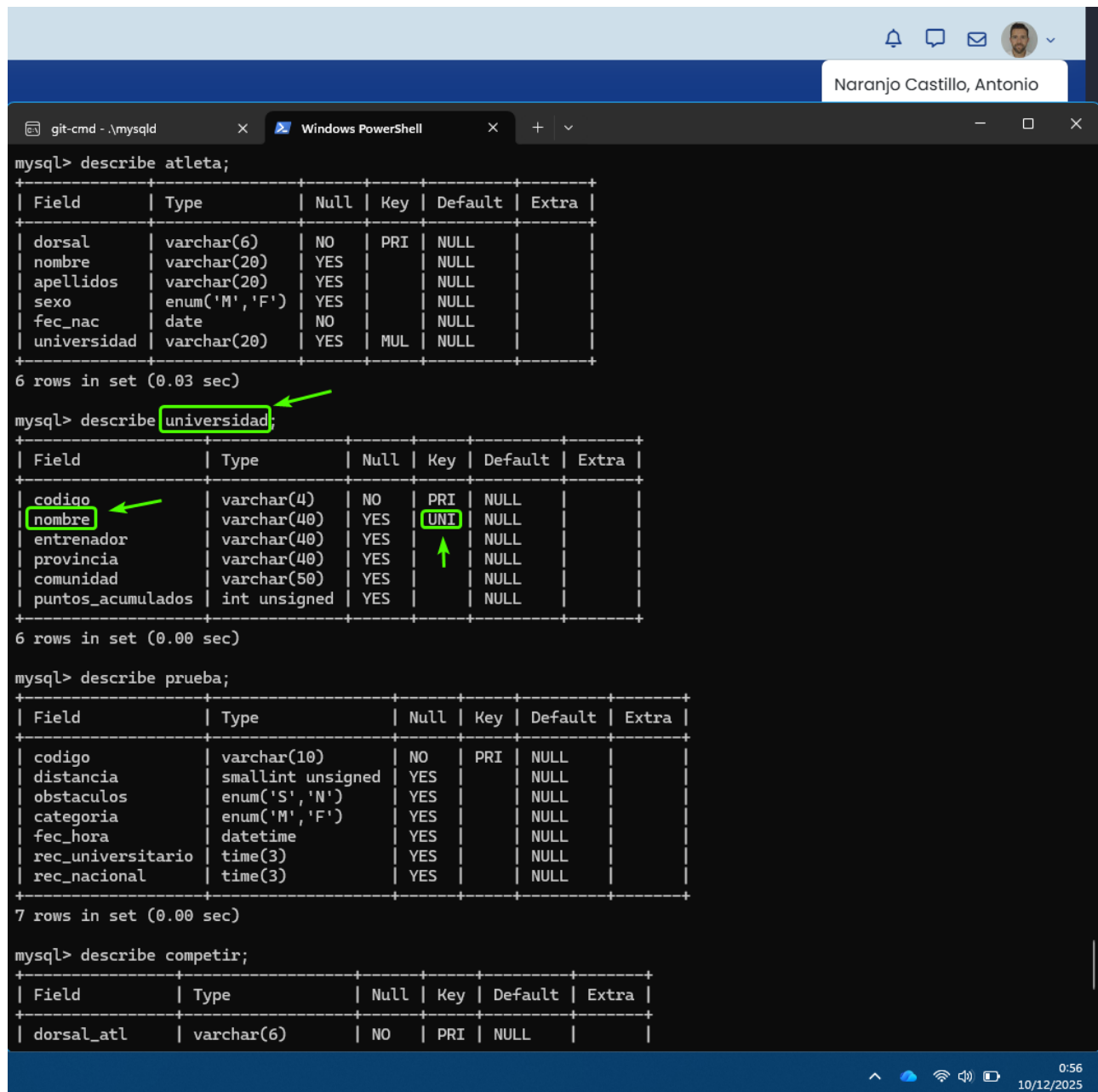
```
>> Lista de códigos de pruebas existentes: [100LF, 100LM, 1500F, 1500M, 30000F, 30000M, 400VF, 400VM]
>> Lista de posiciones de pruebas existentes: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
>> BUILD SUCCESSFUL (total time: 9 seconds)
```

90:40/1:22 INS Unix (LF) 2:04 10/12/2025

## 9. Anexos

### a) Atributos únicos

El único atributo definido como 'unique' es el nombre de las universidades. Importante conocer este dato a la hora de diseñar la sentencia correcta de cara a conseguir los valores únicos y ordenados para alimentar cada uno de los comboBox empleados en la interfaz gráfica Swing presentada en la tarea.



```
mysql> describe atleta;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| dorsal | varchar(6) | NO | PRI | NULL | |
| nombre | varchar(20) | YES | | NULL | |
| apellidos | varchar(20) | YES | | NULL | |
| sexo | enum('M','F') | YES | | NULL | |
| fec_nac | date | NO | | NULL | |
| universidad | varchar(20) | YES | MUL | NULL | |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.03 sec)

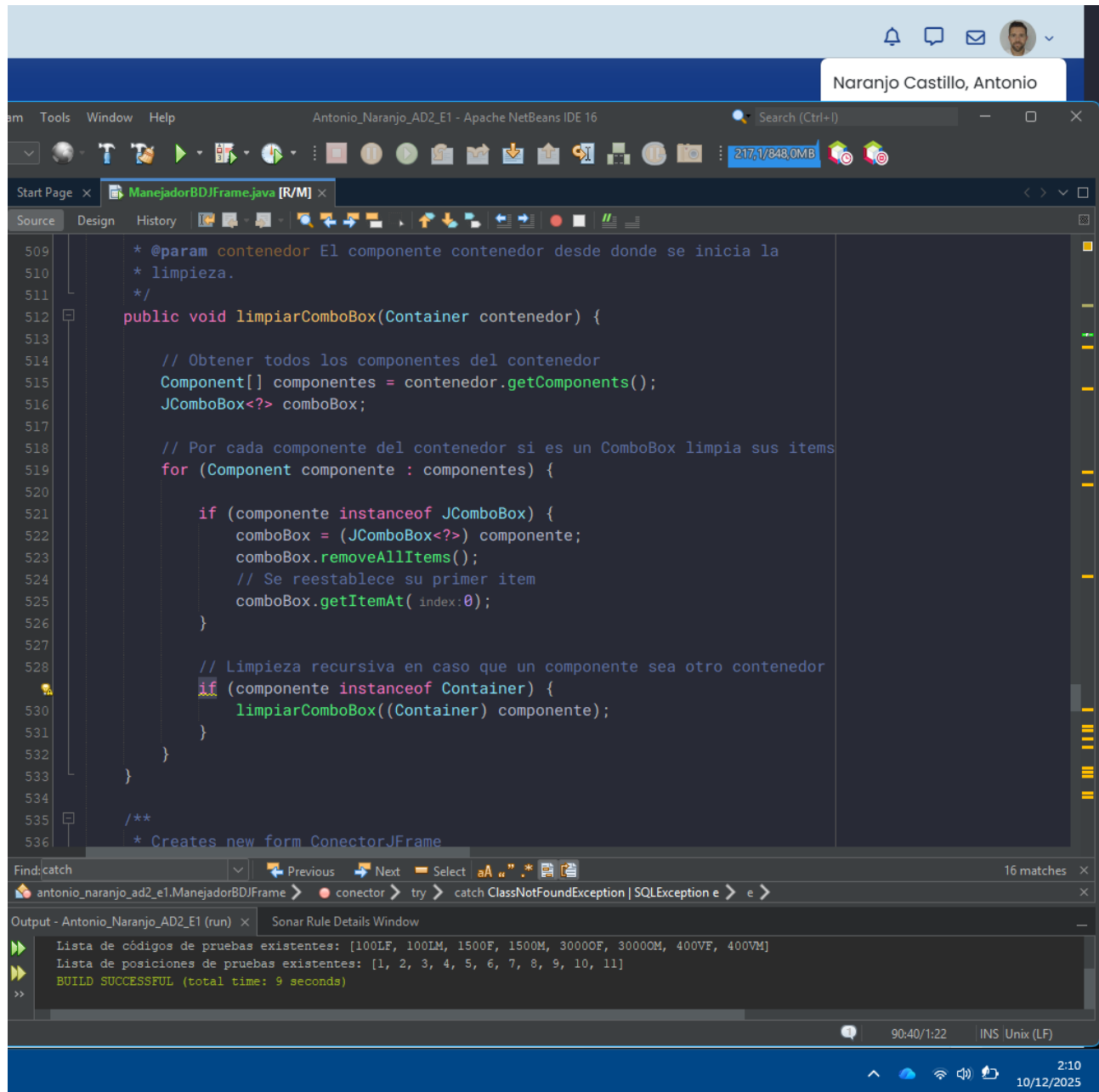
mysql> describe universidad;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| codigo | varchar(4) | NO | PRI | NULL | |
| nombre | varchar(40) | YES | UNI | NULL | |
| entrenador | varchar(40) | YES | | NULL | |
| provincia | varchar(40) | YES | | NULL | |
| comunidad | varchar(50) | YES | | NULL | |
| puntos_acumulados | int unsigned | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)

mysql> describe prueba;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| codigo | varchar(10) | NO | PRI | NULL | |
| distancia | smallint unsigned | YES | | NULL | |
| obstaculos | enum('S','N') | YES | | NULL | |
| categoria | enum('M','F') | YES | | NULL | |
| fec_hora | datetime | YES | | NULL | |
| rec_universitario | time(3) | YES | | NULL | |
| rec_nacional | time(3) | YES | | NULL | |
+-----+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)

mysql> describe competir;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| dorsal_atl | varchar(6) | NO | PRI | NULL | |
+-----+-----+-----+-----+-----+-----+
```

## b) Limpieza de los objetos ComboBox

Se implementa un método complementario para limpiar cada uno de los objetos ComboBox que se disponen en la interfaz gráfica, actuando de manera recursiva sobre componentes que pertenecen a otros componentes, a modo de ejemplo, un comboBox que pertenece a un panel que a su vez pertenece a un frame.



```
509  * @param contenedor El componente contenedor desde donde se inicia la
510  * limpieza.
511  */
512  public void limpiarComboBox(Container contenedor) {
513
514      // Obtener todos los componentes del contenedor
515      Component[] componentes = contenedor.getComponents();
516      JComboBox<?> comboBox;
517
518      // Por cada componente del contenedor si es un ComboBox limpia sus items
519      for (Component componente : componentes) {
520
521          if (componente instanceof JComboBox) {
522              comboBox = (JComboBox<?>) componente;
523              comboBox.removeAllItems();
524              // Se reestablece su primer item
525              comboBox.getItemAt( index:0);
526          }
527
528          // Limpieza recursiva en caso que un componente sea otro contenedor
529          if (componente instanceof Container) {
530              limpiarComboBox((Container) componente);
531          }
532      }
533  }
534
535  /**
536  * Creates new form ConectorJFrame
```

Find: catch 16 matches

antonio\_naranjo\_ad2\_e1.ManejadorBDJFrame > conector > try > catch ClassNotFoundException | SQLException e > e >

Output - Antonio\_Naranjo\_AD2\_E1 (run) x Sonar Rule Details Window

```
>> Lista de códigos de pruebas existentes: [100LF, 100LM, 1500F, 1500M, 3000OF, 3000OM, 400VF, 400VM]
>> Lista de posiciones de pruebas existentes: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
>> BUILD SUCCESSFUL (total time: 9 seconds)
```

90:40/1:22 INS Unix (LF) 2:10 10/12/2025

## c) Sentencias SQL implementadas en el código java del presente proyecto netbeans

- Valores únicos de las comunidades por orden ascendente.
- Nombre y apellidos de los atletas de una determinada universidad que pertenece a la comunidad de Andalucía.
- Valores únicos de los dorsales por orden ascendente.

```
mysql> select distinct comunidad from universidad order by comunidad asc;
+-----+
| comunidad |
+-----+
| Andalucía |
| Cataluña |
| Comunidad de Madrid |
| Comunidad Valenciana |
| Islas Baleares |
+-----+
5 rows in set (0.00 sec)

mysql> select a.nombre, a.apellidos, u.nombre as nomUniv from atleta a, universidad u where a.universidad=u.codigo and u.comunidad='Andalucía';
+-----+-----+-----+
| nombre | apellidos | nomUniv |
+-----+-----+-----+
| Jaime | Pérez López | Universidad de Almería |
| Carlota | Campillo Pérez | Universidad de Almería |
| Juan | Martínez García | Universidad de Granada |
| Alex | Castillo Giménez | Universidad de Granada |
| Luis | Suliman Tez | Universidad de Sevilla |
| Vanesa | Pérez Soriano | Universidad de Sevilla |
| Mauro | Silva Torres | Universidad de Sevilla |
| Silvia | Sanz Barberó | Universidad de Sevilla |
| Alba | Gil Muñoz | Universidad de Sevilla |
+-----+-----+-----+
9 rows in set (0.00 sec)

mysql> select distinct dorsal_atl from competir order by dorsal_atl asc;
+-----+
| dorsal_atl |
+-----+
| 0151 |
| 0152 |
| 0153 |
| 0154 |
| 0155 |
| 0156 |
| 0157 |
| 0158 |
| 0159 |
| 0160 |
| 0161 |
| 0162 |
+-----+
```

2:35  
10/12/2025

- Puntos totales del dorsal 0151.
- Valores únicos de los códigos de las distintas pruebas ordenados de manera ascendente.
- Distancia de la prueba con código 100LF.
- Modificación de la distancia de la prueba con código 100LF a 350.
- Comprobación de la distancia actualmente modificada.

```
git-cmd - .\mysql x Windows PowerShell x + v
Naranjo Castillo, Antonio

28 rows in set (0.00 sec)

mysql> select sum(puntos) as puntosTotales from competir where dorsal_atl='0151';
+-----+
| puntosTotales |
+-----+
|          20 |
+-----+
1 row in set (0.00 sec)

mysql> select distinct codigo from prueba order by codigo asc;
+-----+
| codigo |
+-----+
| 100LF |
| 100LM |
| 1500F |
| 1500M |
| 30000F |
| 30000M |
| 400VF |
| 400VM |
+-----+
8 rows in set (0.00 sec)

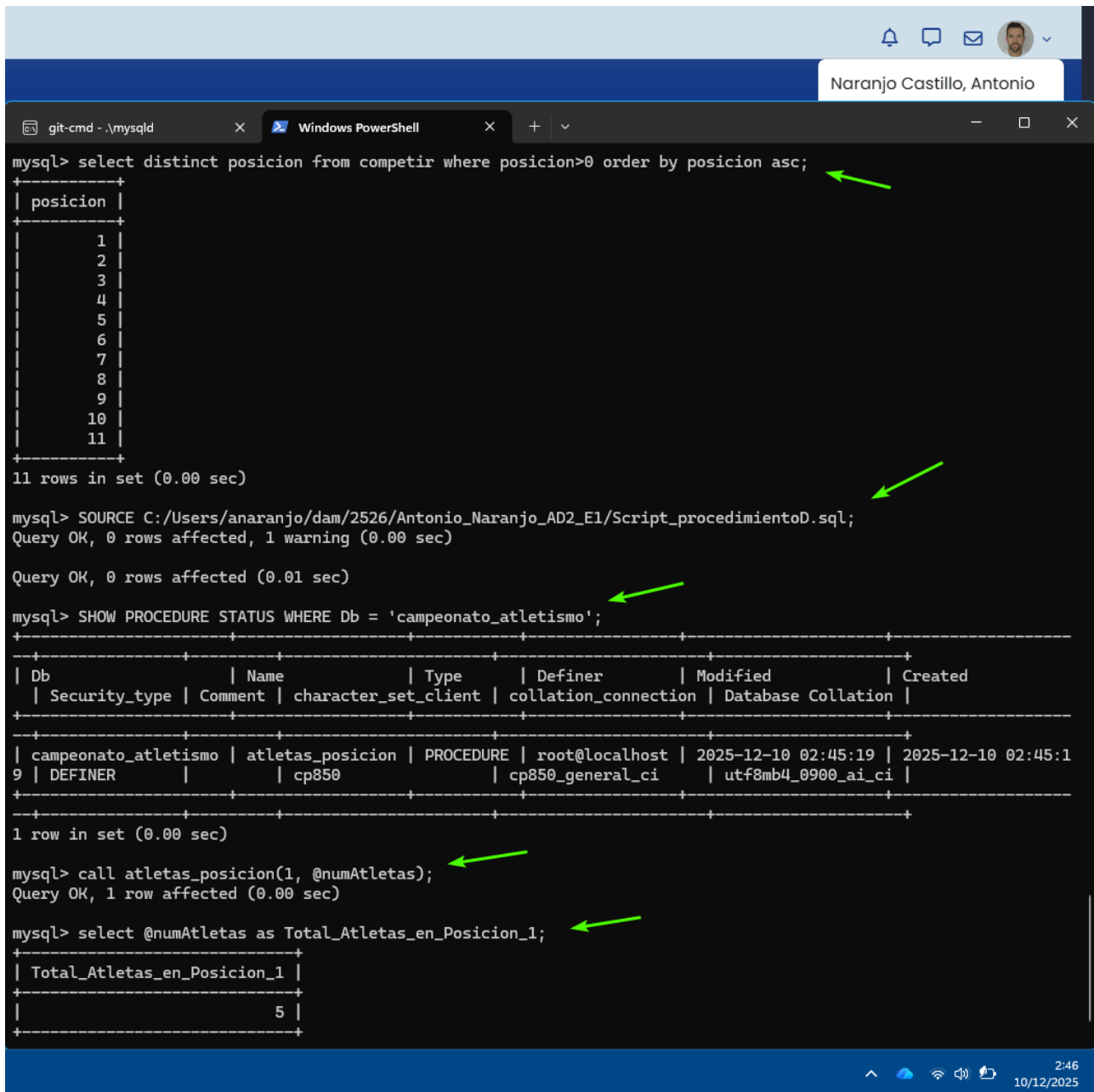
mysql> select distancia from prueba where codigo='100LF';
+-----+
| distancia |
+-----+
|        100 |
+-----+
1 row in set (0.00 sec)

mysql> update prueba set distancia=350 where codigo='100LF';
Query OK, 1 row affected (0.02 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> select distancia from prueba where codigo='100LF';
+-----+
| distancia |
+-----+
|        350 |
+-----+
1 row in set (0.00 sec)
```

2:41  
10/12/2025

- Valores únicos de las posiciones mayores que cero y ordenadas ascendentemente.
- Se carga el script que contiene el procedimiento del apartado D.
- Se comprueba que está correctamente cargado en la base de datos.
- Se llama a tal procedimiento definiendo como parámetro de entrada la posición 1 y como parámetro de salida la variable @numAtletas
- Se consulta el valor adquirido por la variable @numAtletas una vez ejecutado el procedimiento.



```
mysql> select distinct posicion from competir where posicion>0 order by posicion asc;
+-----+
| posicion |
+-----+
|         1 |
|         2 |
|         3 |
|         4 |
|         5 |
|         6 |
|         7 |
|         8 |
|         9 |
|        10 |
|        11 |
+-----+
11 rows in set (0.00 sec)

mysql> SOURCE C:/Users/anaranjo/dam/2526/Antonio_Naranjo_AD2_E1/Script_procedimientoD.sql;
Query OK, 0 rows affected, 1 warning (0.00 sec)

Query OK, 0 rows affected (0.01 sec)

mysql> SHOW PROCEDURE STATUS WHERE Db = 'campeonato_atletismo';
+-----+-----+-----+-----+-----+-----+-----+
| Db      | Name      | Type      | Definer      | Modified      | Created      |
| Security_type | Comment | character_set_client | collation_connection | Database Collation |
+-----+-----+-----+-----+-----+-----+-----+
| campeonato_atletismo | atletas_posicion | PROCEDURE | root@localhost | 2025-12-10 02:45:19 | 2025-12-10 02:45:19 |
| 9 | DEFINER | | cp850 | cp850_general_ci | utf8mb4_0900_ai_ci |
+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

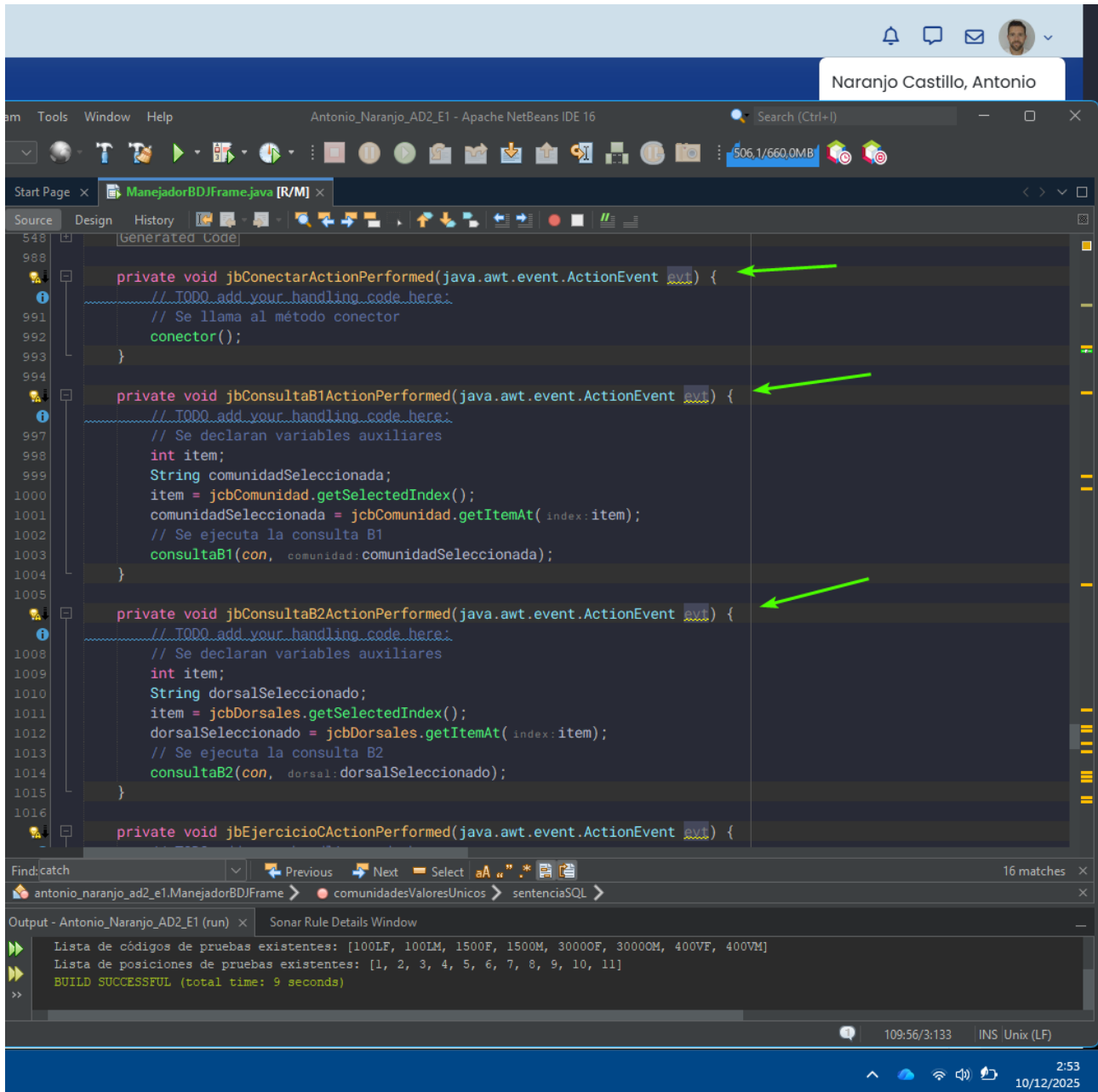
mysql> call atletas_posicion(1, @numAtletas);
Query OK, 1 row affected (0.00 sec)

mysql> select @numAtletas as Total_Atletas_en_Posicion_1;
+-----+
| Total_Atletas_en_Posicion_1 |
+-----+
| 5 |
+-----+
```



## d) Código de cada uno de los botones de la interfaz gráfica

- Botón conectar Ejercicio A.
- Botón consulta Ejercicio B1.
- Botón consulta Ejercicio B2.



```
548 [Generated Code]
988 private void jbConectarActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    // Se llama al método conector
    conector();
}
994
997 private void jbConsultaB1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    // Se declaran variables auxiliares
    int item;
    String comunidadSeleccionada;
    item = jcbComunidad.getSelectedIndex();
    comunidadSeleccionada = jcbComunidad.getItemAt(index:item);
    // Se ejecuta la consulta B1
    consultaB1(con, comunidad:comunidadSeleccionada);
}
1005
1008 private void jbConsultaB2ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    // Se declaran variables auxiliares
    int item;
    String dorsalSeleccionado;
    item = jcbDorsales.getSelectedIndex();
    dorsalSeleccionado = jcbDorsales.getItemAt(index:item);
    // Se ejecuta la consulta B2
    consultaB2(con, dorsal:dorsalSeleccionado);
}
1016
1019 private void jbEjercicioCAActionPerformed(java.awt.event.ActionEvent evt) {
```

Find: catch Previous Next Select aA " . \* [16 matches]

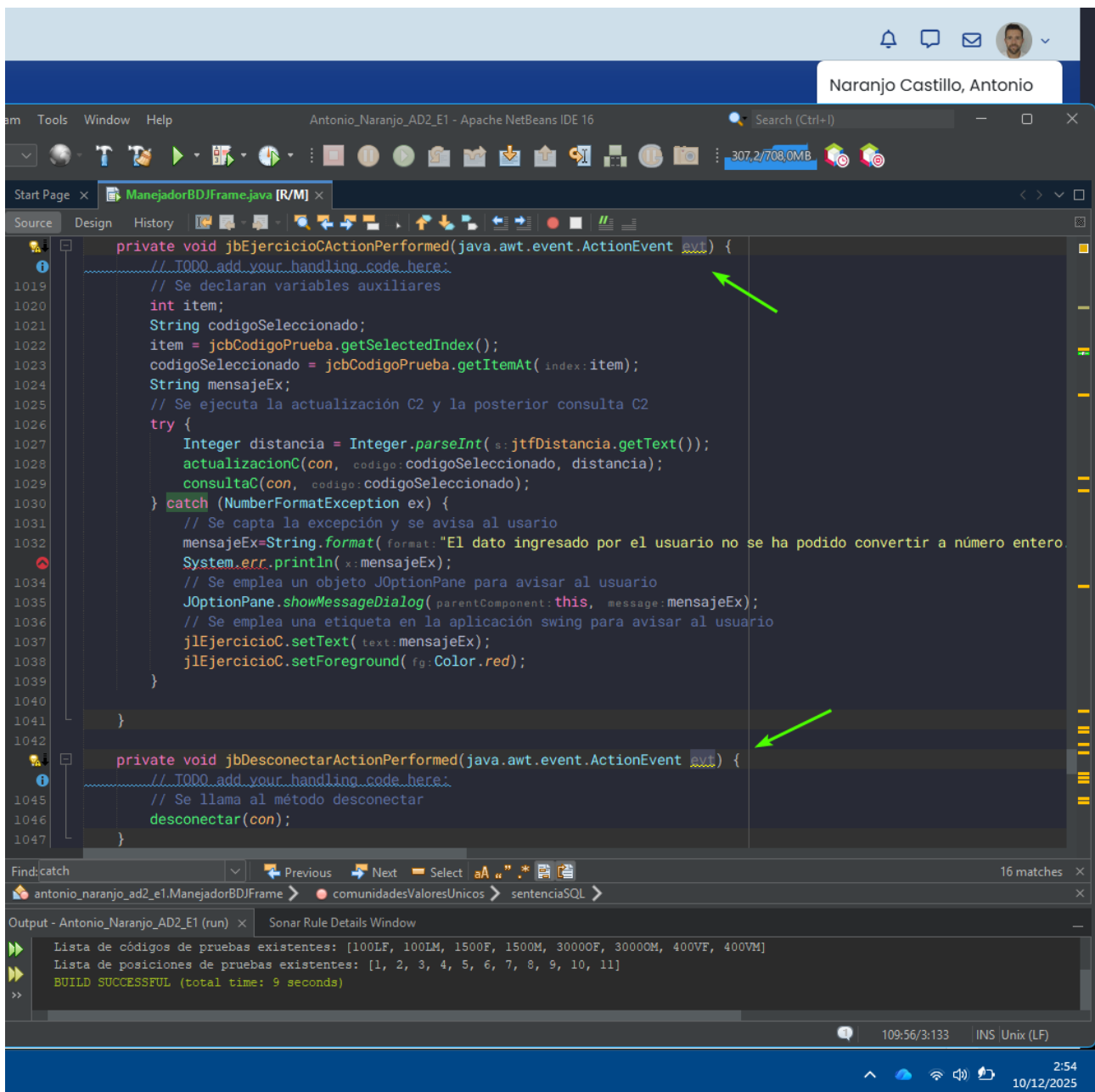
antonio\_naranjo\_ad2\_e1.ManejadorBDJFrame > comunidadesValoresUnicos > sentenciaSQL >

Output - Antonio\_Naranjo\_AD2\_E1 (run) > Sonar Rule Details Window

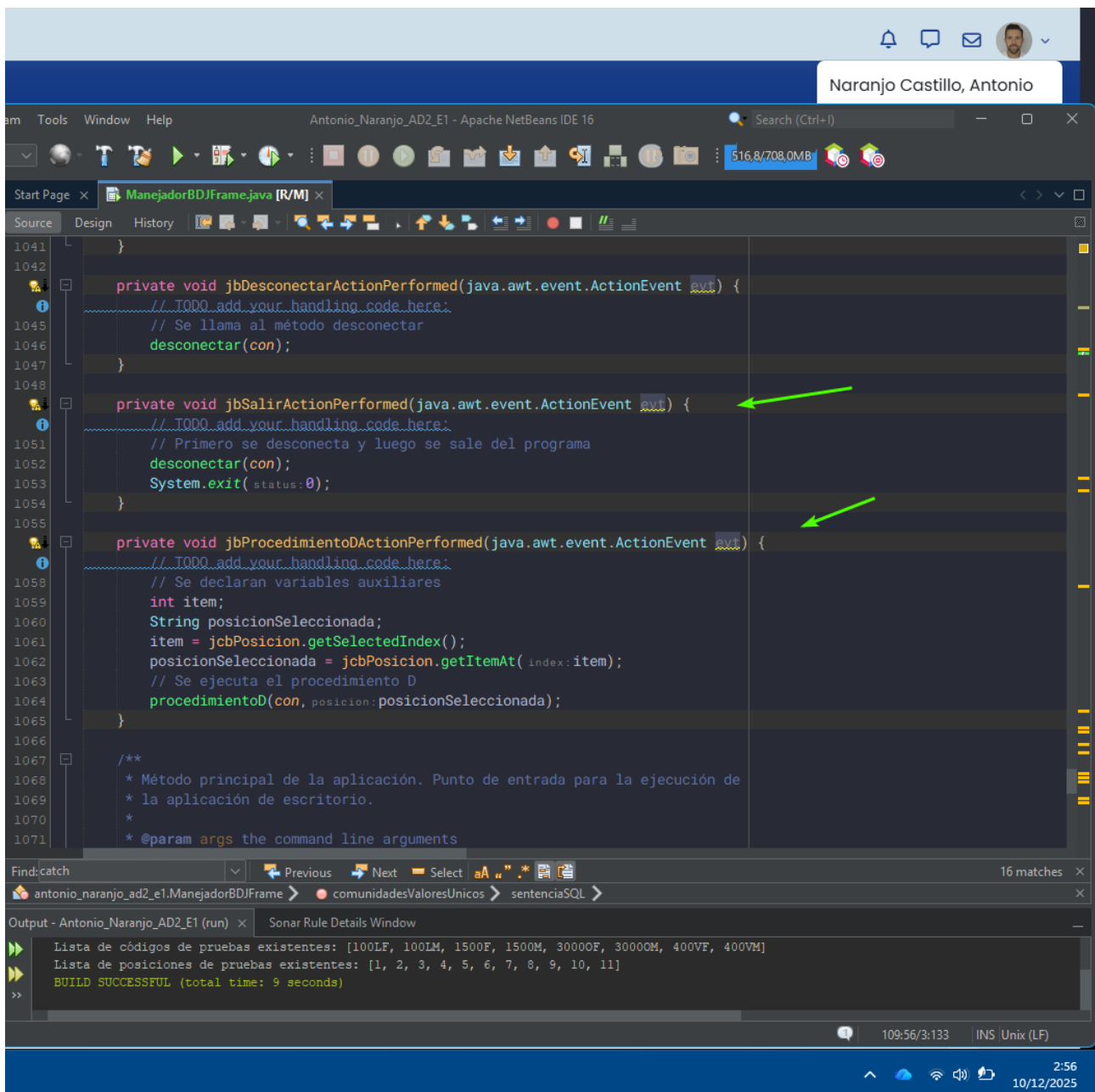
```
>> Lista de códigos de pruebas existentes: [100LF, 100IM, 1500F, 1500M, 30000F, 30000M, 400VF, 400VM]
Lista de posiciones de pruebas existentes: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
BUILD SUCCESSFUL (total time: 9 seconds)
```

109:56/3:133 INS Unix (LF) 2:53 10/12/2025

- Botón actualización y consulta Ejercicio C.
- Botón desconectar.



- Botón Salir y desconectar (para salir del programa).
- Botón procedimiento Ejercicio D.



## e) Puesta en servicio de la aplicación

Se presenta la interfaz gráfica elaborada con Swing para el aplicativo de la tarea.

- Ventana superior izquierda: Ejercicio A, establecimiento de la conexión a la base de datos.
- Ventana inferior izquierda: Ejercicio B1.
- Ventana central derecha: Ejercicio B2.
- Ventana superior derecha: Ejercicio C.
- Ventana inferior derecha: Ejercicio D.

The screenshot displays the 'FORMACIÓN PROFESIONAL' application interface. The top navigation bar includes 'Área personal', 'Mis cursos', and 'Calendario'. The main menu shows 'Curso', 'Participantes', and 'Calificaciones'. The user 'Naranjo Castillo, Antonio' is logged in.

**A.- Establecer conexión con la base de datos**

Usuario:   
Contraseña:   
   
[Conexión establecida](#)

**C.- Modificar la distancia de una prueba determinada indicada por su código**

Código de la prueba:   
Distancia a actualizar:   
 [La distancia de la prueba nº 100LF ha sido actualizada a 100 m.](#)

**B1.- Nombre y apellidos de los atletas junto al nombre de la universidad a la que pertenecen**

Comunidad:   
  
[El atleta Jaime Pérez López pertenece a la Universidad de Almería.](#)  
[El atleta Carlota Campillo Pérez pertenece a la Universidad de Almería.](#)  
[El atleta Juan Martínez García pertenece a la Universidad de Granada.](#)  
[El atleta Alex Castillo Giménez pertenece a la Universidad de Granada.](#)  
[El atleta Luis Suliman Tez pertenece a la Universidad de Sevilla.](#)  
[El atleta Vanesa Pérez Soriano pertenece a la Universidad de Sevilla.](#)  
[El atleta Mauro Silva Torres pertenece a la Universidad de Sevilla.](#)  
[El atleta Silvia Sanz Barberó pertenece a la Universidad de Sevilla.](#)  
[El atleta Alba Gil Muñoz pertenece a la Universidad de Sevilla.](#)

**B2.- Puntos totales que ha obtenido un determinado atleta**

Dorsal:   
 [El atleta con dorsal nº 0151 ha conseguido 20 puntos.](#)

**D.- Procedimiento atletas\_posicion**

Número total de atletas que han acabado en una determinada posición  
Posición:   
 [El número de atletas que han acabado en la posición nº 1 = 5.](#)