

## ***Unidad 2***

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

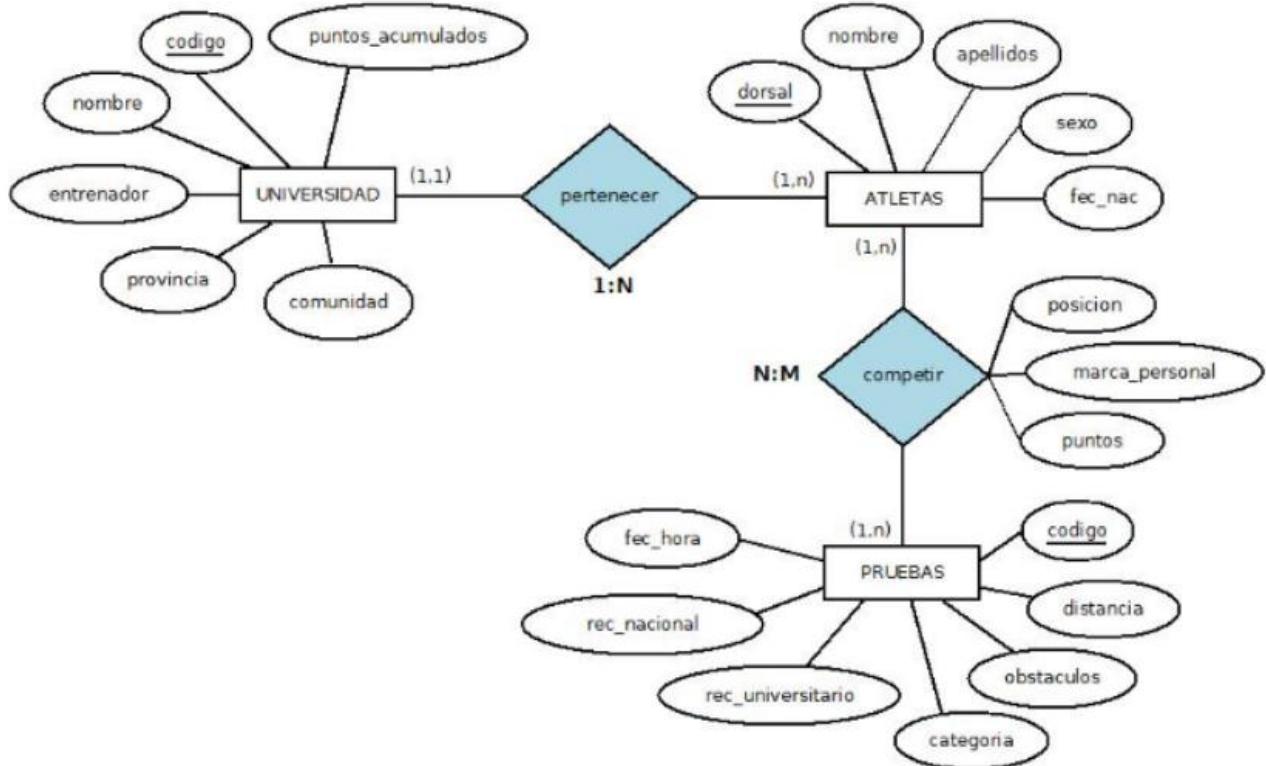
## **Tarea AD02**

### **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).

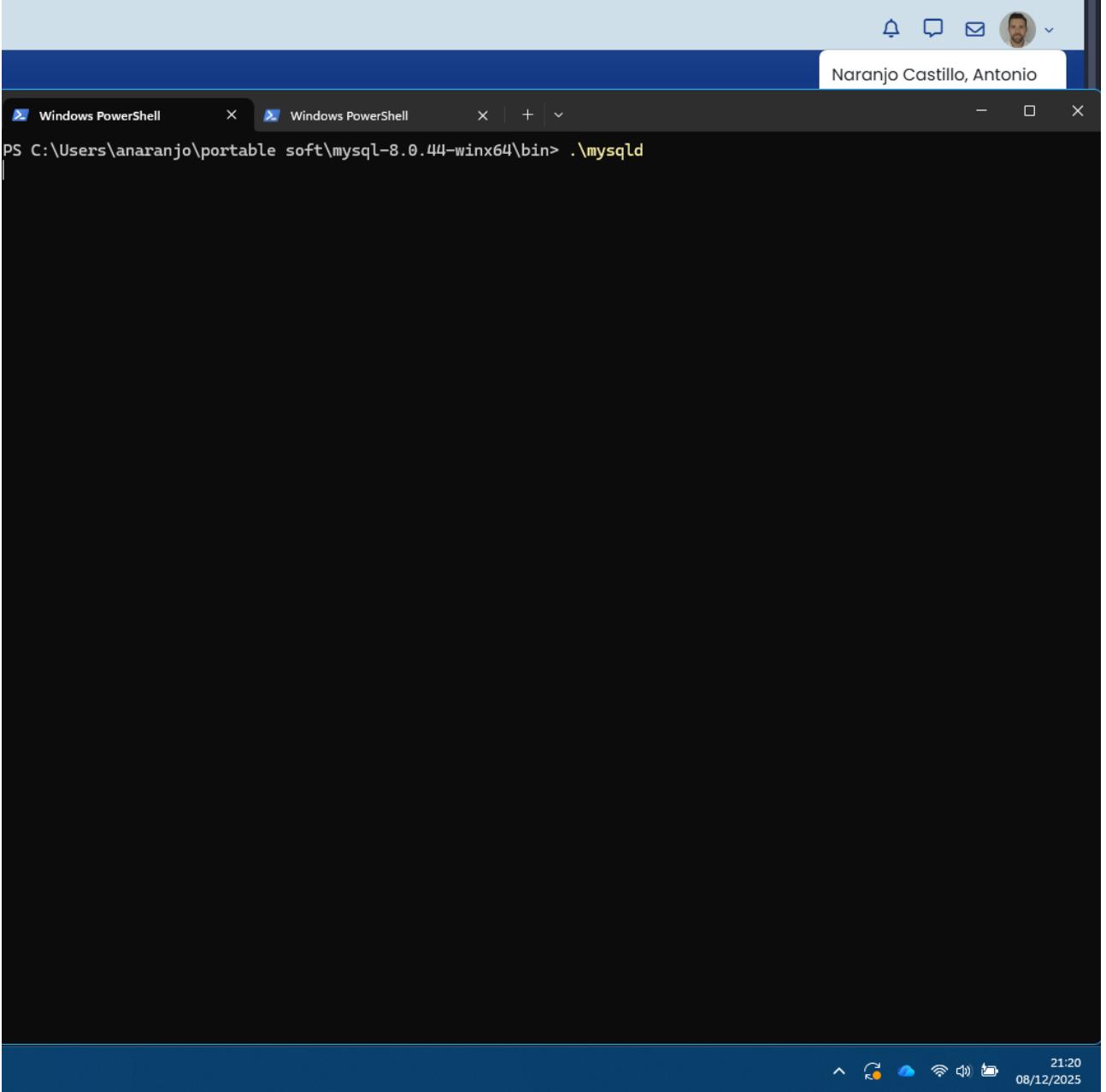
2. Calcular la cantidad total de puntos que ha obtenido un atleta determinado (identificado por su dorsal) en las diferentes pruebas (la aplicación debe permitir al usuario indicar el número de dorsal).
- C. Modificar, a través de una sentencia preparada y parametrizada, la distancia de una prueba determinada 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.



A screenshot of a Windows operating system desktop. At the top, there is a taskbar with several icons: a search bar, a file icon, a recycle bin icon, and others. Below the taskbar is a blue header bar with the text "Naranjo Castillo, Antonio". The main area shows two open windows titled "Windows PowerShell". The left window has the command "PS C:\Users\anaranjo\portable soft\mysql-8.0.44-winx64\bin> .\mysqld" entered. The right window is partially visible. The bottom of the screen features a dark blue footer bar with various system icons and the text "21:20 08/12/2025".

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.

The screenshot shows a Windows PowerShell window with two tabs. The active tab displays the MySQL command-line interface. The session starts with the command `.\mysql -u root -p`, where the password is entered at the prompt. The MySQL monitor then displays its welcome message and connection details. The user then runs `SHOW DATABASES;` to list existing databases, which shows the standard MySQL system databases: information\_schema, mysql, performance\_schema, and sys. Finally, the user executes a SQL script named `Script_campeonatoAtletismo.sql` using the `SOURCE` command. The script contains multiple SQL statements, all of which are executed successfully (indicated by "Query OK" messages). The PowerShell taskbar at the bottom right shows the date as 08/12/2025 and the time as 21:32.

```
PS C:\Users\anaranjo\portable soft\mysql-8.0.44-winx64\bin> .\mysql -u root -p
Enter password: **** 2
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; 3
+-----+
| 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; 4
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.

The screenshot shows a Windows desktop environment with the following details:

- Taskbar:** Displays icons for File Explorer, Microsoft Edge, File History, Task View, Taskbar settings, and a user profile picture.
- PowerShell Windows:** Two windows titled "Windows PowerShell" are open. The left window shows the output of a MySQL query on the "prueba" table. The right window shows the output of a MySQL query on the "competir" table.
- MySQL Terminal:** A separate window titled "MySQL - MySQL Workbench" displays the results of the following queries:

```

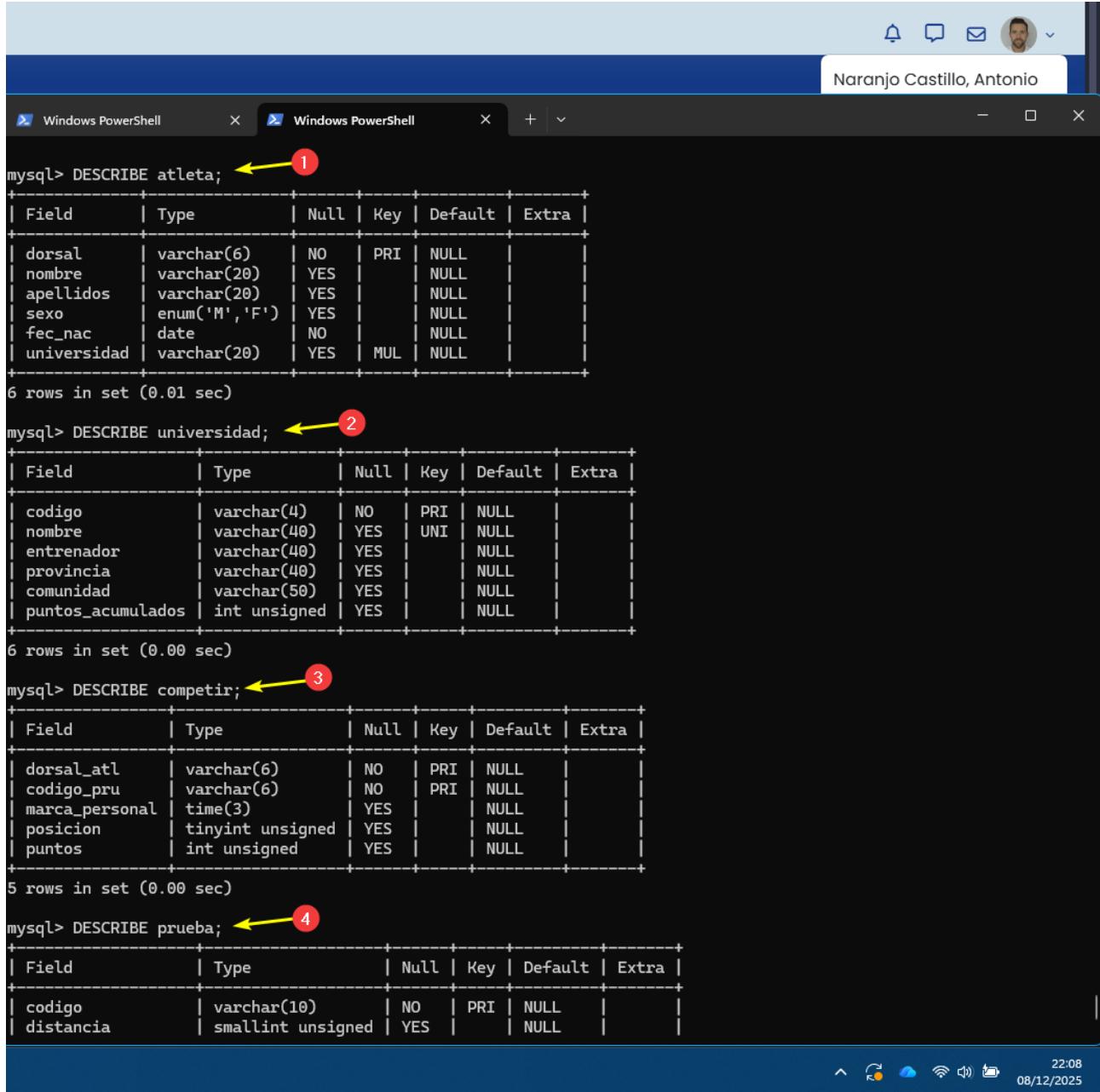
mysql> SELECT * FROM prueba; 1
+-----+-----+-----+-----+-----+-----+-----+
| codigo | distancia | obstaculos | categoria | fec_hora | rec_universitario | rec_nacional |
+-----+-----+-----+-----+-----+-----+-----+
| 100LF | 100 | N | F | 2017-02-03 09:30:00 | 00:00:12.020 | 00:00:11.060 |
| 100LM | 100 | 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 |
| 30000F | 3000 | S | F | 2017-02-05 09:00:00 | 00:11:12.380 | 00:09:07.320 |
| 30000M | 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 | 30000M | 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 | 30000F | 00:11:24.290 | 2 | 20 |
| 0153 | 100LM | 00:00:10.210 | 1 | 30 |
| 0153 | 1500M | 00:04:03.380 | 1 | 30 |
| 0153 | 30000M | 00:10:43.030 | 4 | 10 |
| 0153 | 400VM | 00:00:50.420 | 1 | 30 |
| 0154 | 1500M | 00:05:02.380 | 9 | 0 |
| 0154 | 30000M | 00:11:22.540 | 7 | 0 |
| 0154 | 400VM | 00:00:52.080 | 6 | 0 |
+-----+-----+-----+-----+-----+

```

The MySQL session ends with the timestamp 08/12/2025 22:06.

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



```
mysql> DESCRIBE atleta; 1
+-----+-----+-----+-----+-----+-----+
| 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)

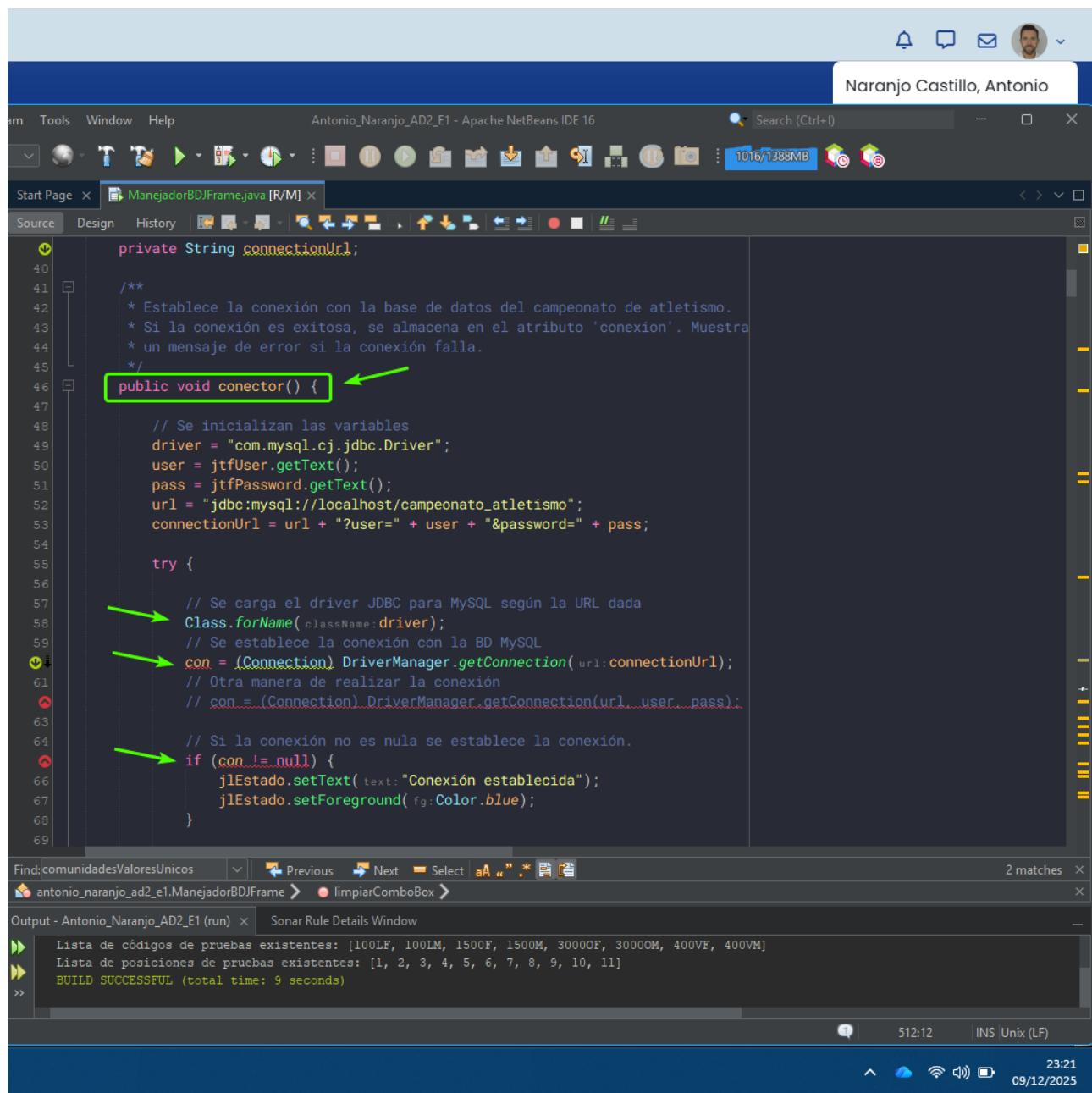
mysql> DESCRIBE universidad; 2
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| codigo | varchar(4) | NO | PRI | NULL |       |
| nombre | varchar(40) | YES | UNI | NULL |       |
| entrenador | varchar(40) | YES |     | NULL |       |
| 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; 3
+-----+-----+-----+-----+-----+-----+
| 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; 4
+-----+-----+-----+-----+-----+-----+
| 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 `getConnnection` 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`.

The screenshot shows the Apache NetBeans IDE 16 interface. The main window displays the Java code for `ManejadorBDJFrame.java`. The code handles connection establishment, cleaning up existing items in dropdown menus, and populating three `JComboBox` components (`jcbComunidad`, `jcbDorsales`, `jcbCodigoPrueba`) with unique values from a database. It also handles exceptions related to class loading and SQL errors. Below the code editor is the Output window, which shows the results of a Sonar Rule Details Window analysis and a successful build message. The status bar at the bottom indicates the time as 23:59 and the date as 09/12/2025.

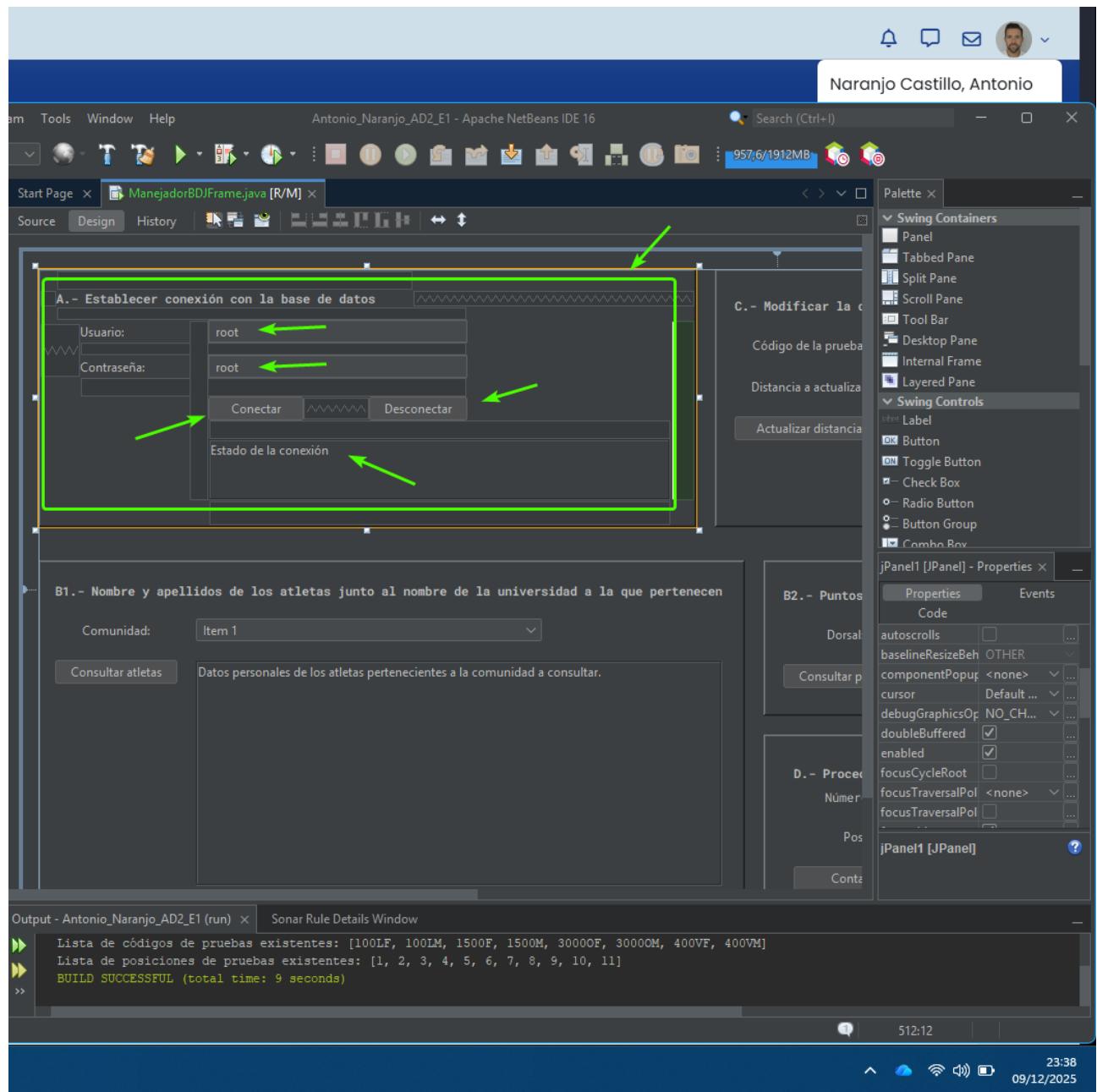
```
66     jlEstado.setText("Conexión establecida");
67     jlEstado.setForeground(Color.blue);
68 }
69
70 // Limpieza a fondo de los items de todos los ComboBox
71 limpiarComboBox(contenedor: this.getContentPane());
72
73 // Alimentar cada combobox por sus valores únicos
74 for (String comunidad : comunidadesValoresUnicos(con)) {
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 }
```

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, 3000F, 3000M, 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)

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

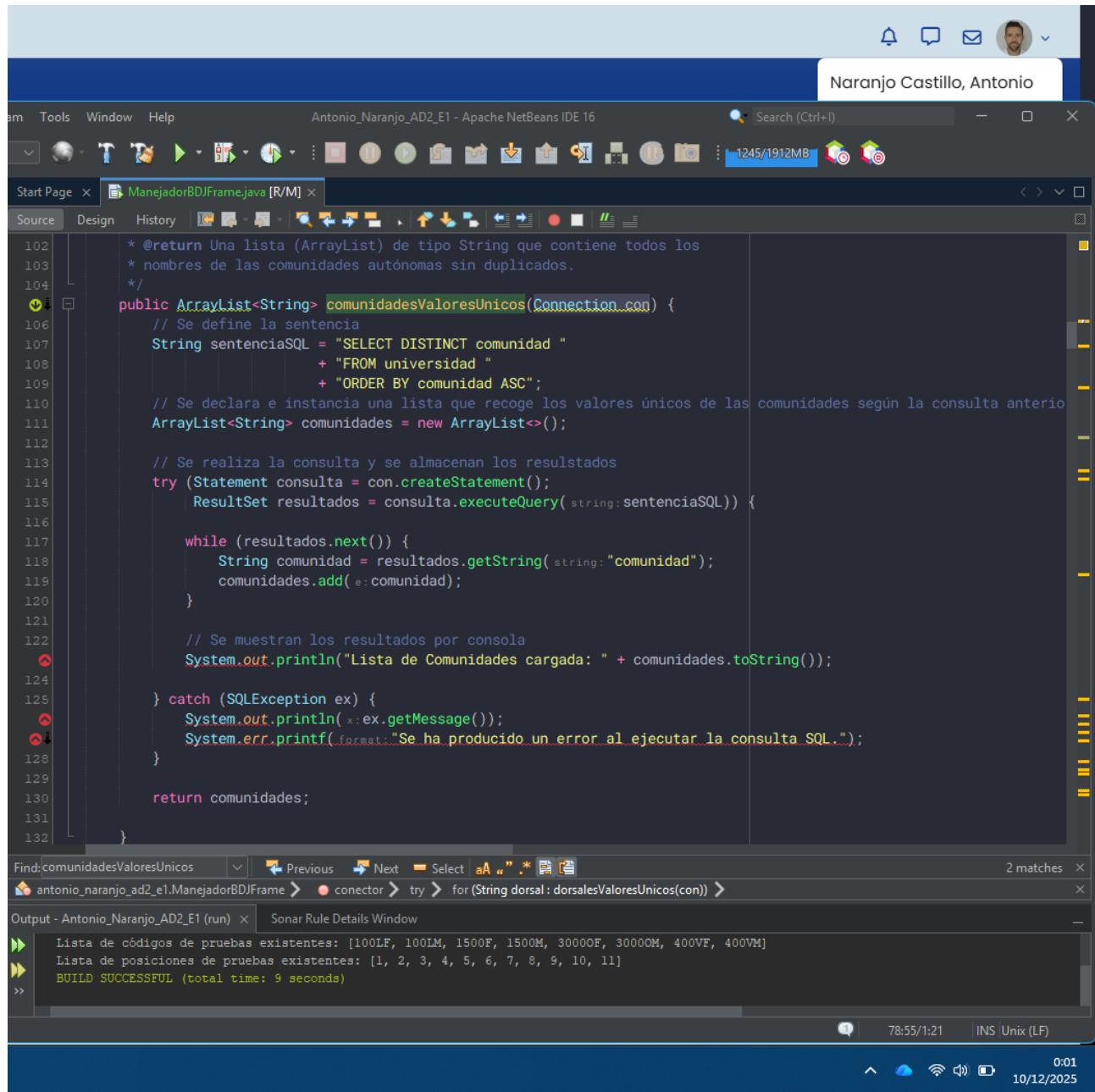
Se diseña una interfaz usando Swing con un panel reservado para establecer la conexión, incluyendo campos de texto para introducir usuario y contraseña, botones para activar/desactivar la conexión y una etiqueta para mostrar el resultado obtenido.



## 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(string:sentenciaSQL)) {
116
117            while (resultados.next()) {
118                String comunidad = resultados.getString(string:"comunidad");
119                comunidades.add(e: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(x:ex.getMessage());
127            System.err.printf(format:"Se ha producido un error al ejecutar la consulta SQL.");
128        }
129
130        return comunidades;
131    }
132}

```

Find:comunidadesValoresUnicos

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

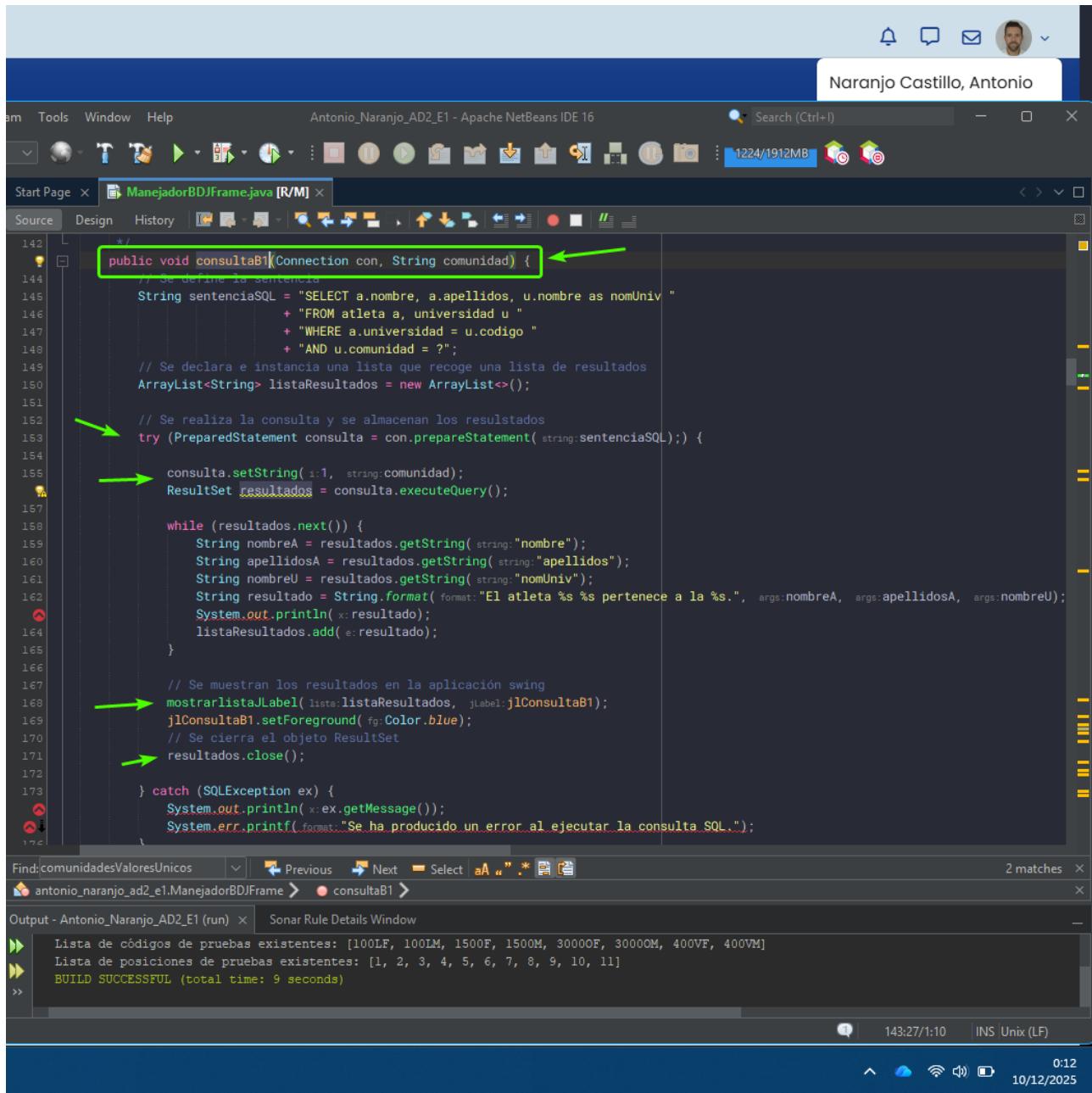
Lista de códigos de pruebas existentes: [100LF, 100LM, 1500F, 1500M, 30000F, 30000M, 400VF, 400VM]  
 Lista de posiciones de pruebas existentes: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]  
 BUILD SUCCESSFUL (total time: 9 seconds)

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 */
143     public void consultaB1( Connection con, String comunidad) {
144         // Se define la sentencia
145         String sentenciasSQL = "SELECT a.nombre, a.apellidos, u.nombre as nomUniv "
146             + "FROM atleta a, universidad u "
147             + "WHERE a.universidad = u.codigo "
148             + "AND u.comunidad = ?";
149         // Se declara e instancia una lista que recoge una lista de resultados
150         ArrayList<String> listaResultados = new ArrayList<>();
151
152         // Se realiza la consulta y se almacenan los resultados
153         try (PreparedStatement consulta = con.prepareStatement(string:sentenciasSQL)) {
154
155             consulta.setString( i:1, string:comunidad);
156             ResultSet resultados = consulta.executeQuery();
157
158             while (resultados.next()) {
159                 String nombreA = resultados.getString( string:"nombre");
160                 String apellidosA = resultados.getString( string:"apellidos");
161                 String nombreU = resultados.getString( string:"nomUniv");
162                 String resultado = String.format( format:"El atleta %s %s pertenece a la %s.", args:nombreA, args:apellidosA, args:nombreU);
163                 System.out.println( x:resultado);
164                 listaResultados.add( e:resultados);
165             }
166
167             // Se muestran los resultados en la aplicación swing
168             mostrarListaLabel( lista:listaResultados, jLabel:jlConsultaB1);
169             jlConsultaB1.setForeground( fg:Color.blue);
170             // Se cierra el objeto ResultSet
171             resultados.close();
172
173         } catch (SQLException ex) {
174             System.out.println( x:ex.getMessage());
175             System.err.printf( format:"Se ha producido un error al ejecutar la consulta SQL.");
176         }
177     }

```

Find:comunidadesValoresUnicos Previous Next Select aA “.\*” Output - Antonio\_Naranjo\_AD2\_E1 (run) 2 matches x 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, 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) 0:12  
 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.

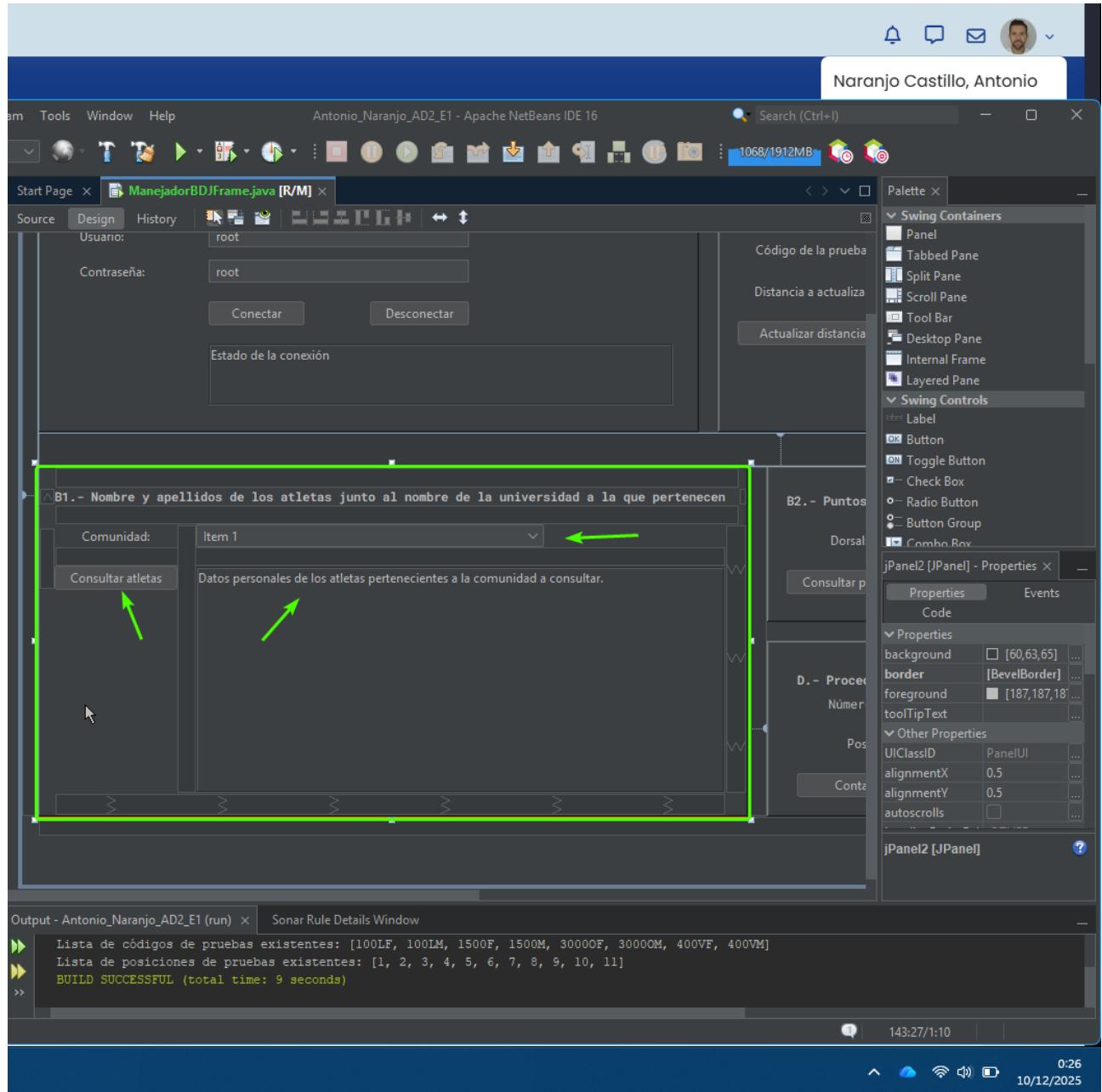
The screenshot shows the Apache NetBeans IDE 16 interface. The title bar displays "Antonio\_Naranjo\_AD2\_E1 - Apache NetBeans IDE 16". The main window shows the Java code for "ManejadorBDJFrame.java". The code defines a method `mostrarListaJLabel` that takes an `ArrayList<String>` and a `JLabel` as parameters. It uses a `StringBuilder` to build a string with HTML line breaks (`<br>`) and then sets the `setText` method of the `JLabel` to this string. Below the code, there is a comment block with a `/**` header, which describes the method's purpose of recovering unique dorsal values from the 'competir' table and returning them as a list of strings. The bottom part of the interface shows the "Output" window with the results of a build process, including the command run, the list of existing test codes, the list of existing test positions, and the message "BUILD SUCCESSFUL (total time: 9 seconds)".

```
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("<html>");
197     // Contenido de la lista más salto de linea
198     for (String str : lista) {
199         sb.append(str).append("<br>");
200     }
201     // Cerrar etiqueta html
202     sb.append("</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
213 */

Find:comunidadesValoresUnicos | Previous | Next | Select | aA | .* | Sonar Rule Details Window
antonio_naranjo_ad2_e1.ManejadorBDJFrame > consultaB1 >
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)
143:27:1:10 | 0:19 | INS Unix (LF)
10/12/2025
```

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

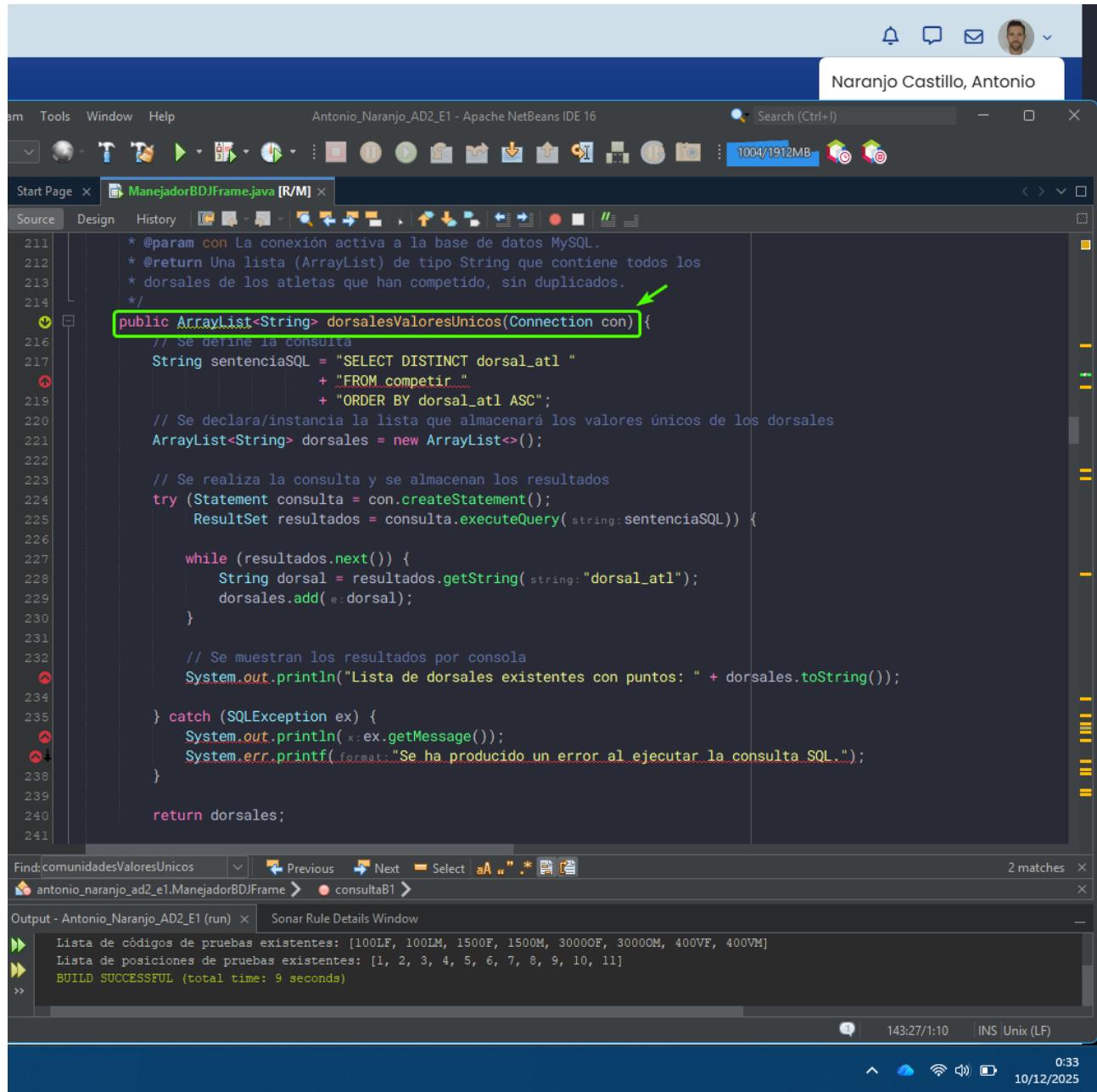
En la interfaz gráfica Swing, se asigna un panel para la consulta en cuestión, completado por un comboBox que recoge los valores únicos de las comunidades, un botón que ejecuta el método anterior consultaB1() y una etiqueta que presenta los resultados obtenidos de dicha consulta.



## 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.



The screenshot shows the Apache NetBeans IDE 16 interface. The title bar reads "Antonio\_Naranjo\_AD2\_E1 - Apache NetBeans IDE 16". The main window displays Java code in the "ManejadorBDJFrame.java" file. A green arrow points to the method signature:

```
public ArrayList<String> dorsalesValoresUnicos(Connection con){
```

The code implements a method to execute a SQL query that retrieves unique dorsal numbers from the "competir" table, ordered by dorsal number. It uses a try-catch block to handle SQL exceptions and prints the results to the console. The output panel at the bottom shows the executed command and its successful completion.

```
Find:comunidadesValoresUnicos |> Previous Next Select aA “*” |> Find > antonio_naranjo_ad2_e1.ManejadorBDJFrame > consultaB1 >
Output - Antonio_Naranjo_AD2_E1 (run) > Sonar Rule Details Window
>> Lista de códigos de pruebas existentes: [100LF, 100LM, 1500F, 1500M, 3000F, 3000M, 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)
```

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(string:sentenciaSQL)) {
266
267         consulta.setString(1, string:dorsal);
268         ResultSet resultados = consulta.executeQuery();
269
270         while (resultados.next()) {
271             puntos = resultados.getInt(string:"puntosTotales");
272             mensajeSalida = String.format(format:"El atleta con dorsal nº %s ha conseguido %d puntos.", args:dorsal, args:puntos);
273             System.out.println(x:mensajeSalida);
274         }
275
276         // Se muestra el resultado en la aplicación swing
277         jlConsultaB2.setText(text:mensajeSalida);
278         jlConsultaB2.setForeground(fg:Color.blue);
279         // Se cierra el objeto ResultSet
280         resultados.close();
281
282     } catch (SQLException ex) {
283         System.out.println(x:ex.getMessage());
284         System.err.printf(format:"Se ha producido un error al ejecutar la consulta SQL.");
285     }
286 }

```

Find:comunidadesValoresUnicos Previous Next Select aA « » \* Output - Antonio\_Naranjo\_AD2\_E1 (run) Sonar Rule Details Window

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

```

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)

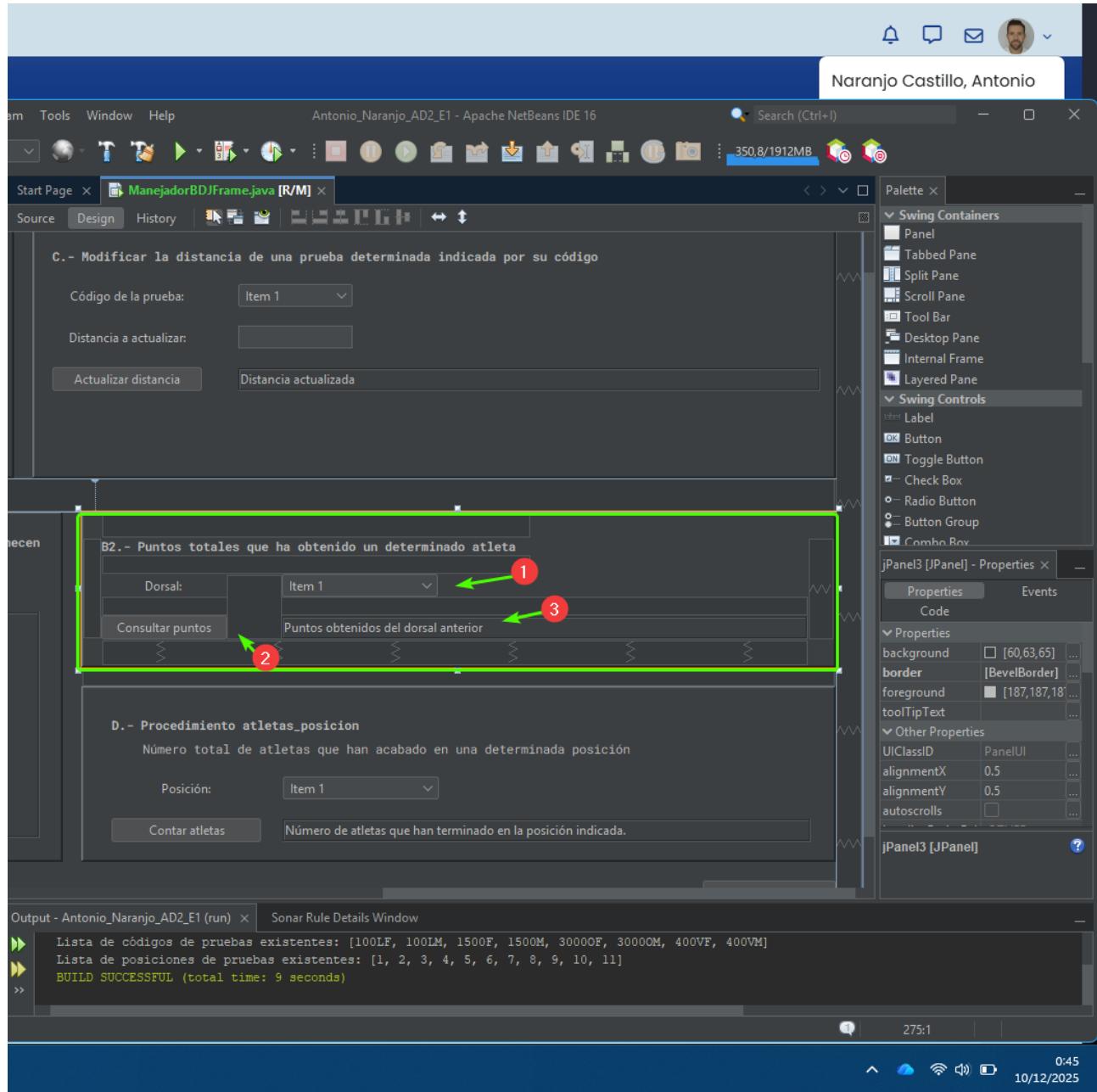
```

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.

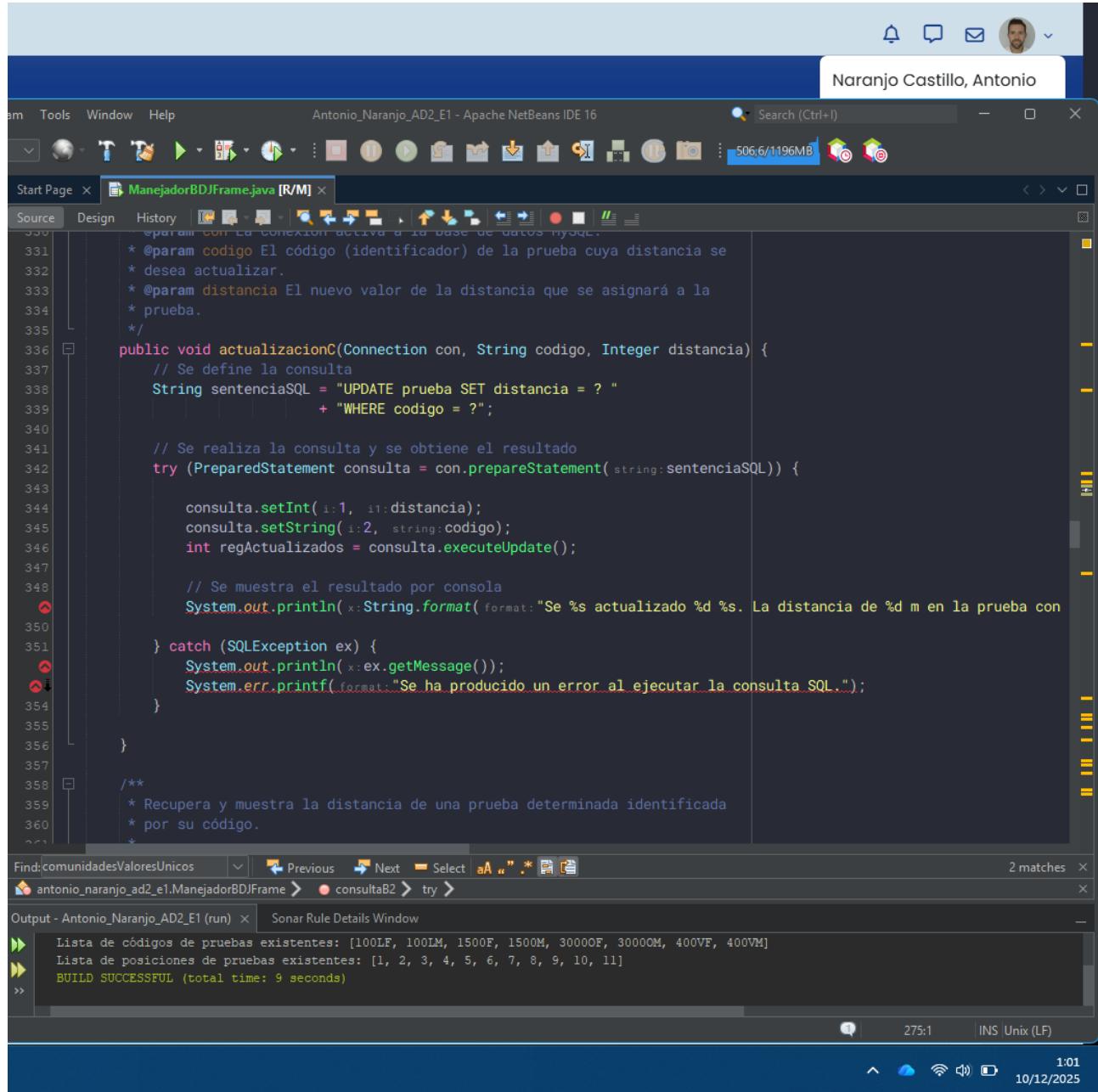
The screenshot shows the Apache NetBeans IDE interface with the following details:

- Title Bar:** Antonio\_Naranjo\_AD2\_E1 - Apache NetBeans IDE 16
- User Profile:** Naranjo Castillo, Antonio
- Toolbar:** Standard NetBeans toolbar with icons for file operations, search, and help.
- Code Editor:** The main window displays the Java code for `ManejadorBDJFrame.java`. The code implements a method `codigoPruebaValoresUnicos` which performs a SQL query to get unique test codes from the `prueba` table, sorts them ascendingly, and prints the results to the console.
- Annotations:** Two green arrows highlight specific lines of code:
  - An arrow points to the `SELECT DISTINCT` part of the SQL query.
  - An arrow points to the `codigo.add(e.codigo);` line where the results are added to the list.
- Find Bar:** Shows the search term `comunidadesValoresUnicos` with 2 matches found.
- Output Window:** Displays the results of the executed SQL query:

```
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 anteriores, 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.



The screenshot shows the Apache NetBeans IDE 16 interface. The title bar displays "Antonio\_Naranjo\_AD2\_E1 - Apache NetBeans IDE 16". The menu bar includes File, Tools, Window, Help. The toolbar has various icons for file operations like Open, Save, Print, and others. The code editor window is open with the file "ManejadorBDJFrame.java [R/M]". The code implements a method named "actualizacionC" which updates a database table "prueba" by setting the "distancia" column to a new value for a specific "codigo". The code uses JDBC to prepare and execute an UPDATE statement. The output window at the bottom shows the command-line interface with the results of the executed SQL query.

```
param con La conexión activa a la base de datos MySQL.
 * @param codigo El código (identificador) de la prueba cuya distancia se
 * desea actualizar.
 * @param distancia El nuevo valor de la distancia que se asignará a la
 * prueba.
 */
public void actualizacionC(Connection con, String codigo, Integer distancia) {
    // Se define la consulta
    String sentenciaSQL = "UPDATE prueba SET distancia = ? "
                           + "WHERE codigo = ?";

    // Se realiza la consulta y se obtiene el resultado
    try (PreparedStatement consulta = con.prepareStatement(string:sentenciaSQL)) {

        consulta.setInt(i:1, ii:distancia);
        consulta.setString(i:2, string:codigo);
        int regActualizados = consulta.executeUpdate();

        // Se muestra el resultado por consola
        System.out.println(x:String.format(format:"Se %s actualizado %d %. La distancia de %d m en la prueba con
            );
    } catch (SQLException ex) {
        System.out.println(x:ex.getMessage());
        System.err.printf(format:"Se ha producido un error al ejecutar la consulta SQL.");
    }
}

/**
 * Recupera y muestra la distancia de una prueba determinada identificada
 * por su código.
 *
```

Find:comunidadesValoresUnicos | Previous | Next | Select | aA « » \* | Find

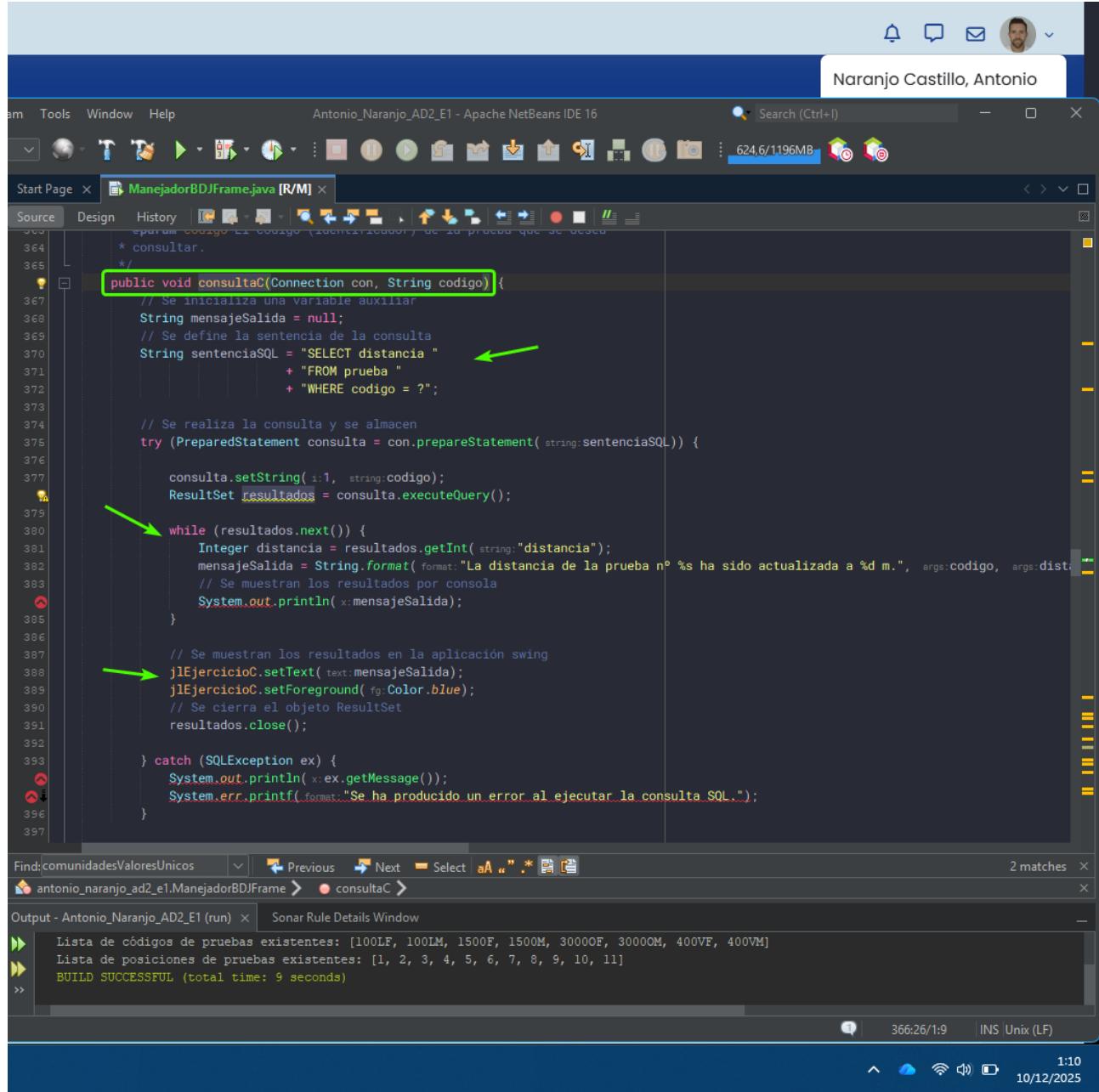
antonio\_naranjo\_ad2\_e1.ManejadorBDJFrame > consultoB2 > try >

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

```
Lista de códigos de pruebas existentes: [100LF, 100LM, 1500F, 1500M, 3000F, 3000M, 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)
```

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 almacen
375         try (PreparedStatement consulta = con.prepareStatement(string:sentenciaSQL)) {
376
376             consulta.setString( i:1, string:codigo);
377             ResultSet resultados = consulta.executeQuery();
378
379             while (resultados.next()) {
380                 Integer distancia = resultados.getInt(string:"distancia");
381                 mensajeSalida = String.format(format:"La distancia de la prueba nº %s ha sido actualizada a %d m.", args:codigo, args:distancia);
382                 // Se muestran los resultados por consola
383                 System.out.println(x:mensajeSalida);
384             }
385
386             // Se muestran los resultados en la aplicación swing
387             jLEjercicioC.setText(text:mensajeSalida);
388             jLEjercicioC.setForeground(fg:Color.blue);
389             // Se cierra el objeto ResultSet
390             resultados.close();
391
392         } catch (SQLException ex) {
393             System.out.println(x:ex.getMessage());
394             System.err.printf(format:"Se ha producido un error al ejecutar la consulta SQL.");
395         }
396     }
397 
```

Find:comunidadesValoresUnicos Previous Next Select aA “.\*”

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

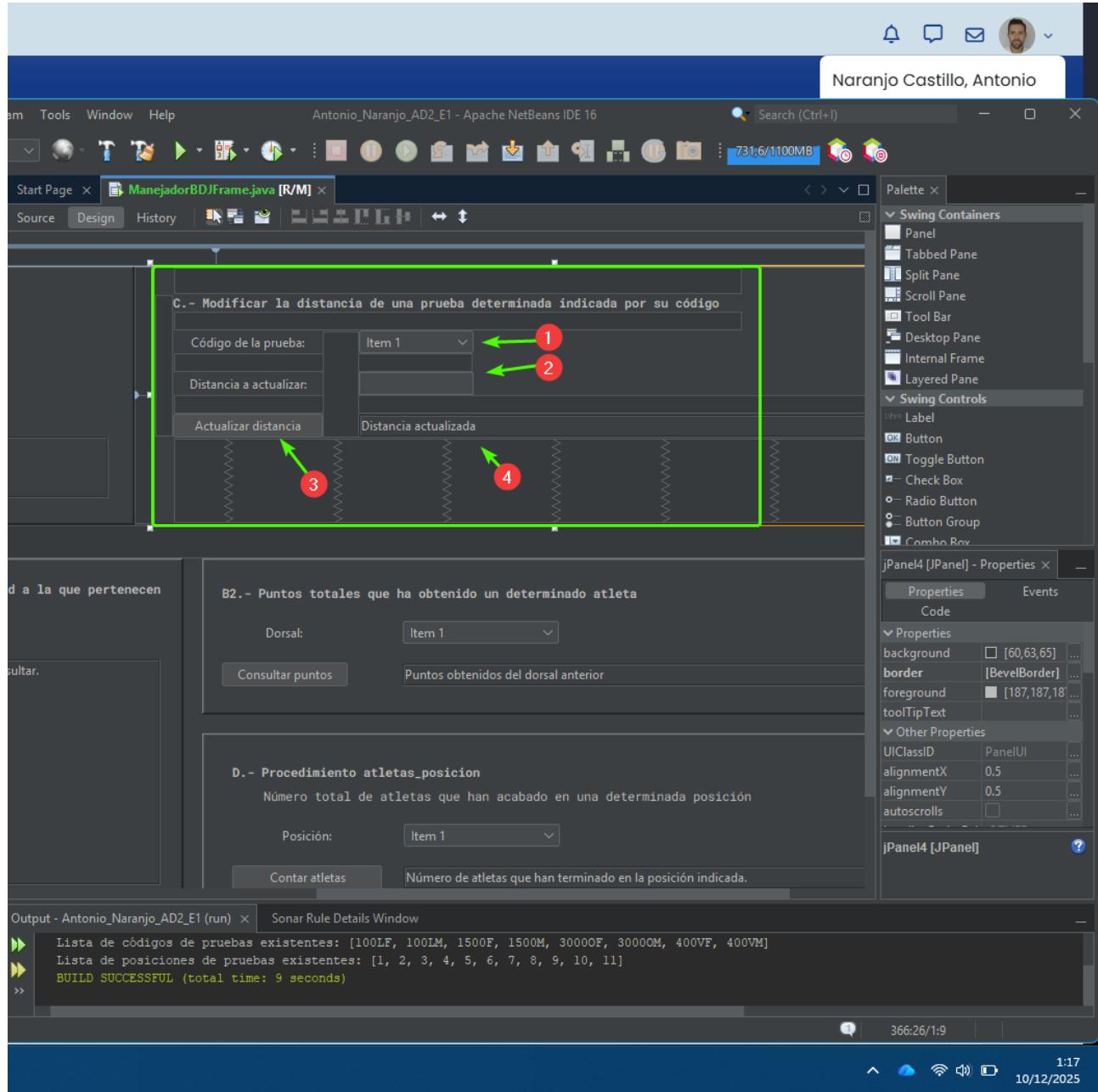
- ▶ Lista de códigos de pruebas existentes: [100LF, 100LM, 1500F, 1500M, 3000E, 3000M, 400VF, 400VM]
- ▶ Lista de posiciones de pruebas existentes: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
- » BUILD SUCCESSFUL (total time: 9 seconds)

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.

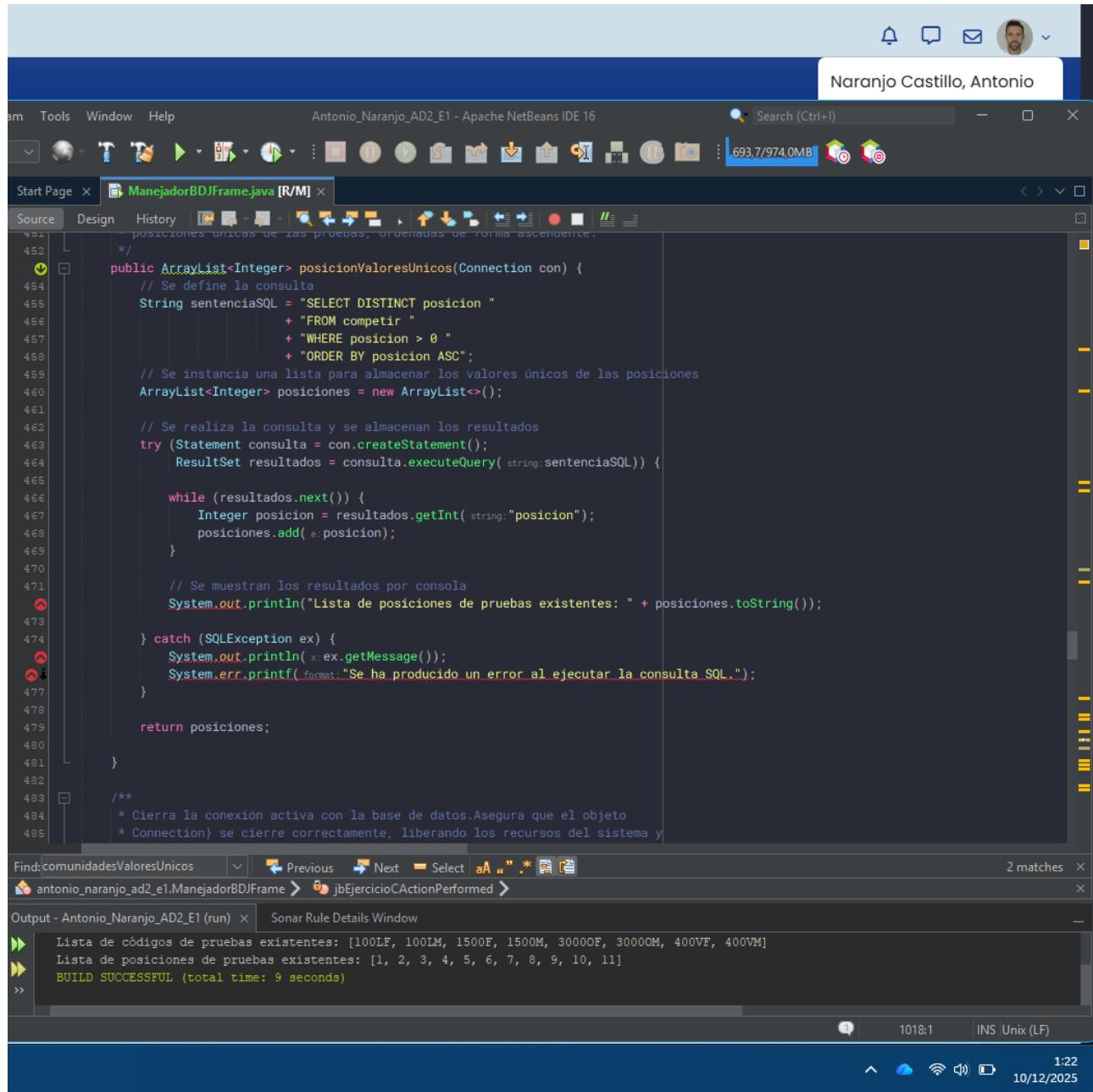
Se recogen los datos obtenidos, en este caso una fila, y se muestra en la aplicación Swing, por medio de una etiqueta. Finalmente se cierra el objeto ResultSet.

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 actualizacionC() 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 ascendentemente de las posiciones de todas las pruebas.



The screenshot shows the Apache NetBeans IDE 16 interface. The title bar reads "Antonio\_Naranjo\_AD2\_E1 - Apache NetBeans IDE 16". The main window displays the Java code for a class named "ManejadorBDJFrame.java". The code implements a method to retrieve unique test positions from a database. The IDE's status bar at the bottom right shows the date and time as "10/12/2025 1:22".

```
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 * Cierra la conexión activa con la base de datos.Asegura que el objeto
484 * Connection} se cierre correctamente, liberando los recursos del sistema y
485 */
```

Find:comunidadesValoresUnicos | Find Next Previous Select aA “.\*”

antonio\_naranjo.ad2.e1.ManejadorBDJFrame > jbEjercicioCActionPerformed

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)

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

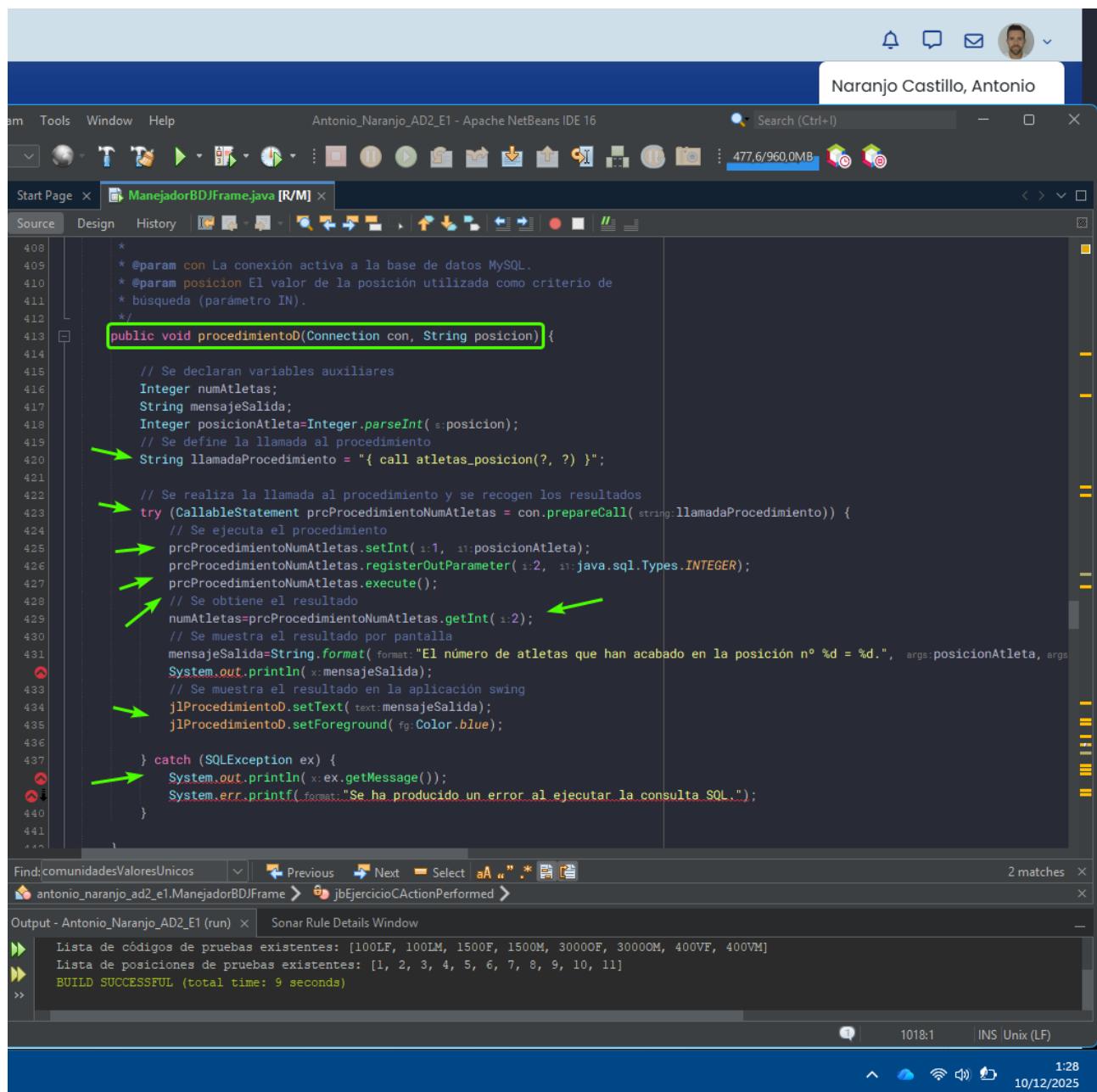
Se implementa el método procedimientoD() que recibe como argumentos el objeto Connection y un objeto String posicion.

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 gráfica.



```
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( string:llamadaProcedimiento)) {
424         // Se ejecuta el procedimiento
425         prcProcedimientoNumAtletas.setInt( 1, 1:posicionAtleta);
426         prcProcedimientoNumAtletas.registerOutParameter( 2, 1:java.sql.Types.INTEGER);
427         prcProcedimientoNumAtletas.execute();
428
429         // Se obtiene el resultado
430         numAtletas=prcProcedimientoNumAtletas.getInt( 2);
431         // Se muestra el resultado por pantalla
432         mensajeSalida=String.format("El número de atletas que han acabado en la posición nº %d = %d.", args:posicionAtleta, args
433         System.out.println( x:mensajeSalida);
434         // Se muestra el resultado en la aplicación swing
435         jlProcedimientoD.setText( text:mensajeSalida);
436         jlProcedimientoD.setForeground( fg:Color.blue);
437
438     } catch (SQLException ex) {
439         System.out.println( x:ex.getMessage());
440         System.err.printf("Se ha producido un error al ejecutar la consulta SQL.");
441     }
442 }
```

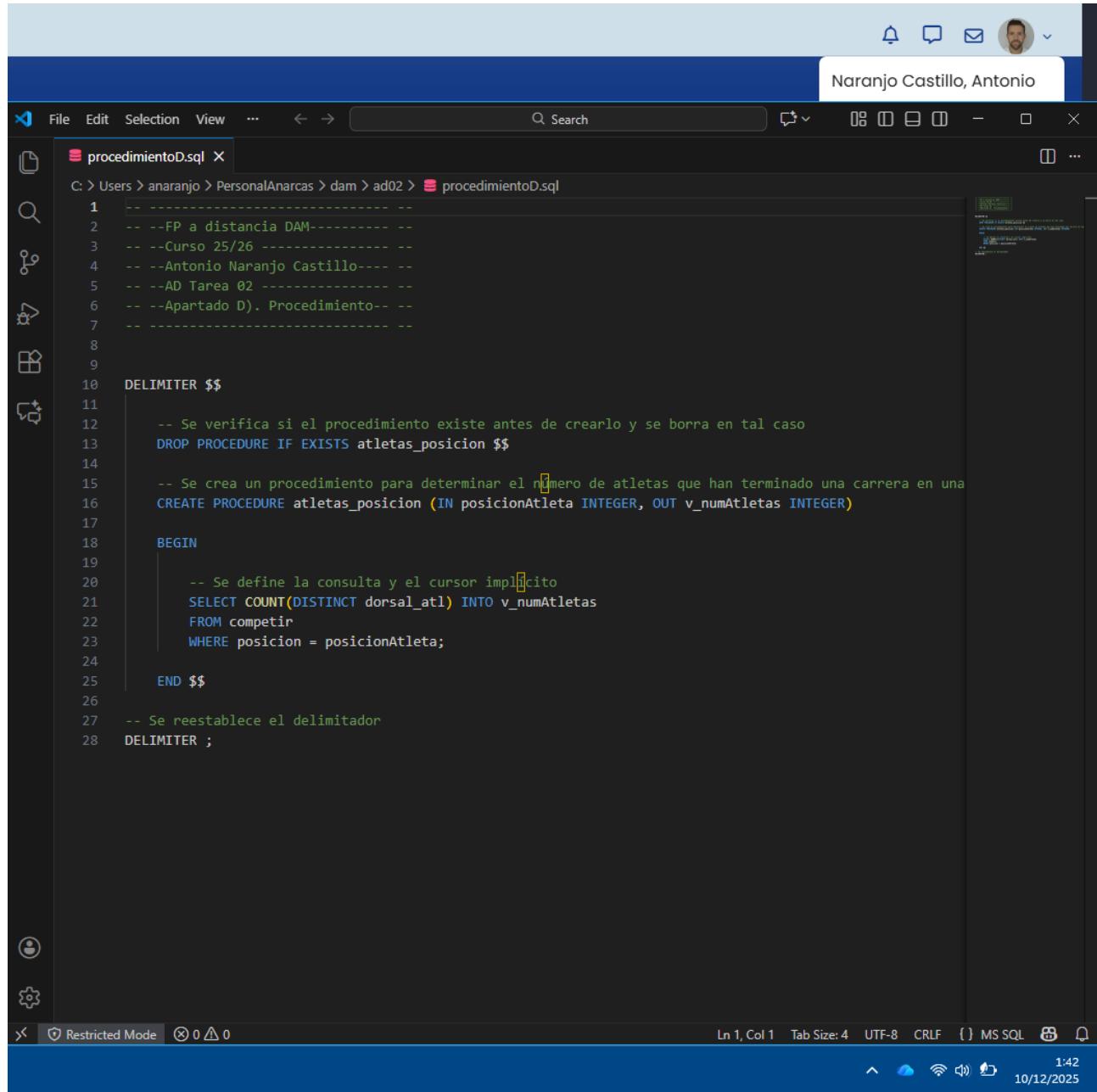
Find:comunidadesValoresUnicos Previous Next Select aA “\*” ⌂ 2 matches ×  
antonio\_naranjo\_ad2\_e1.ManejadorBDJFrame > jbEjercicioCActionPerformed >

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)

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.



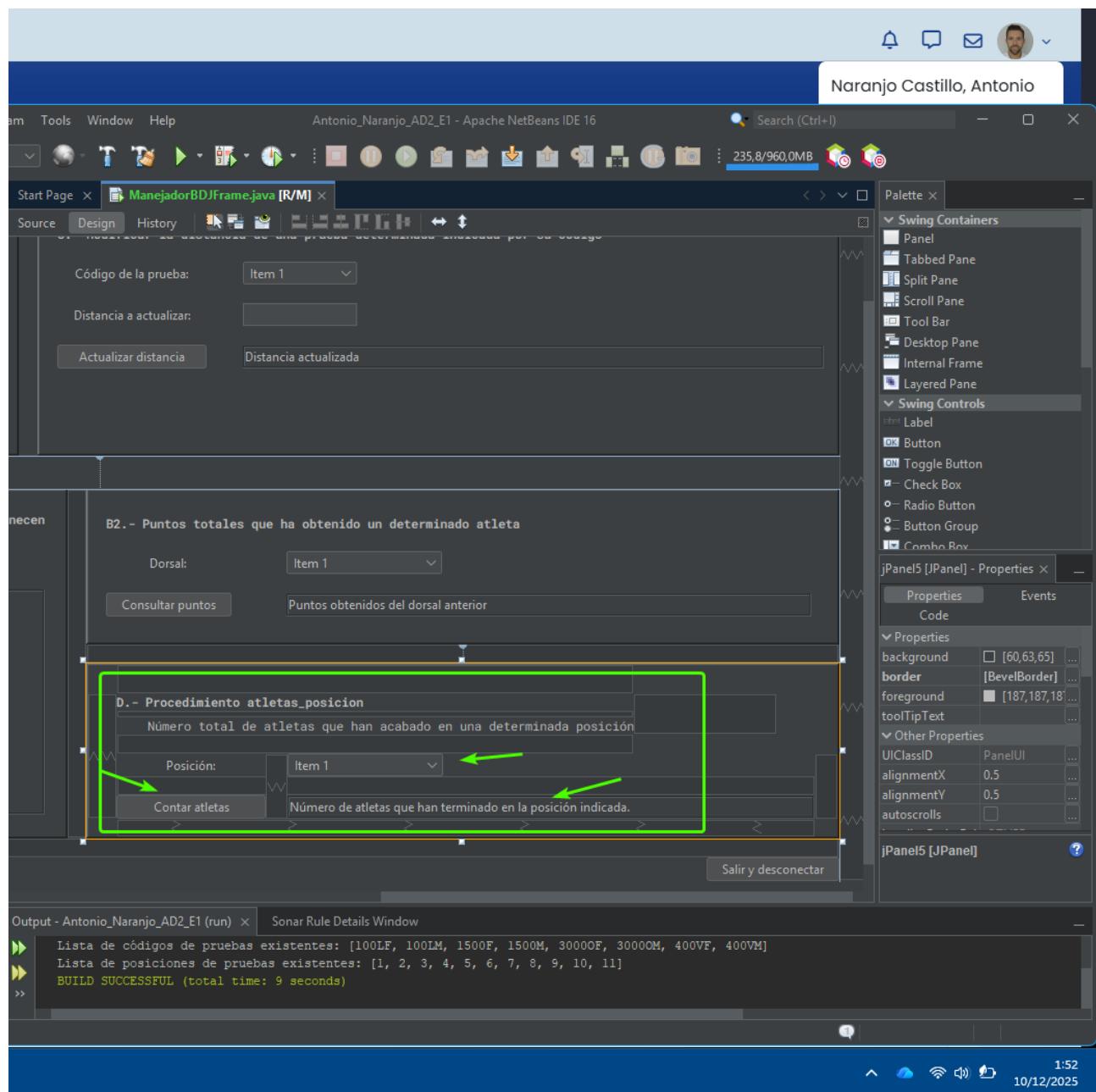
```
1 -- FP a distancia DAM-
2 -- Curso 25/26 -
3 -- Antonio Naranjo Castillo --
4 -- AD Tarea 02 --
5 -- Apartado D). Procedimiento --
6 --
7 --
8 --
9 
10 DELIMITER $$ 
11 
12 -- Se verifica si el procedimiento existe antes de crearlo y se borra en tal caso
13 DROP PROCEDURE IF EXISTS atletas_posicion $$ 
14 
15 -- Se crea un procedimiento para determinar el n mero 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 impl cito
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\_posición y una etiqueta que presenta el valor del resultado del tal procedimiento.

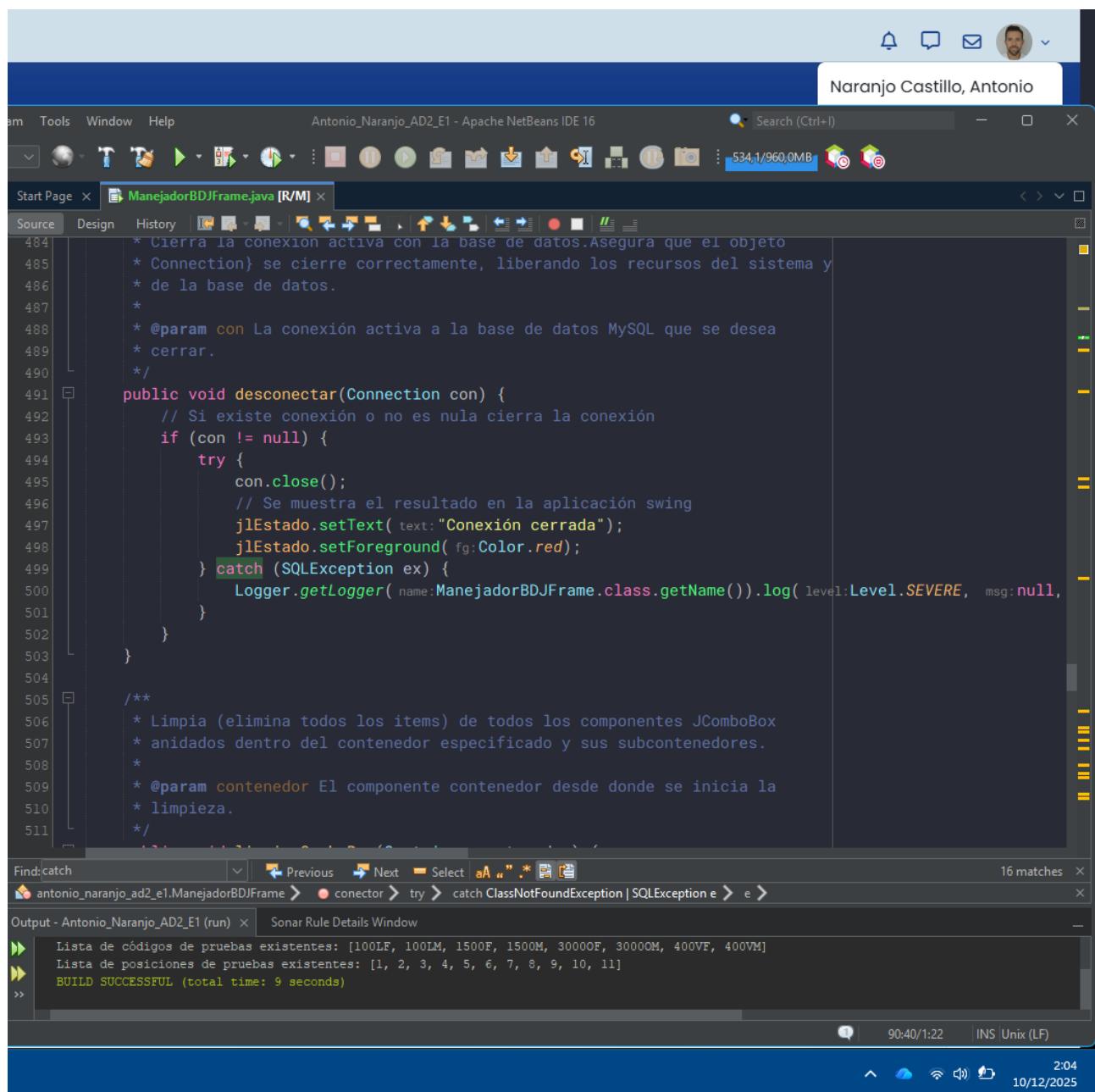


## 8. Gestión de Excepciones y Cierre de Recursos

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 conectar().

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( name:ManejadorBDJFrame.class.getName()).log( level:Level.SEVERE, msg:null,
501         }
502     }
503 }
504 /**
505 * Limpia (elimina todos los items) de todos los componentes JComboBox
506 * anidados dentro del contenedor especificado y sus subcontenedores.
507 *
508 * @param contenedor El componente contenedor desde donde se inicia la
509 * limpieza.
510 */
511
```

Find:catch | Previous | Next | Select | aA | .\* | Find

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

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

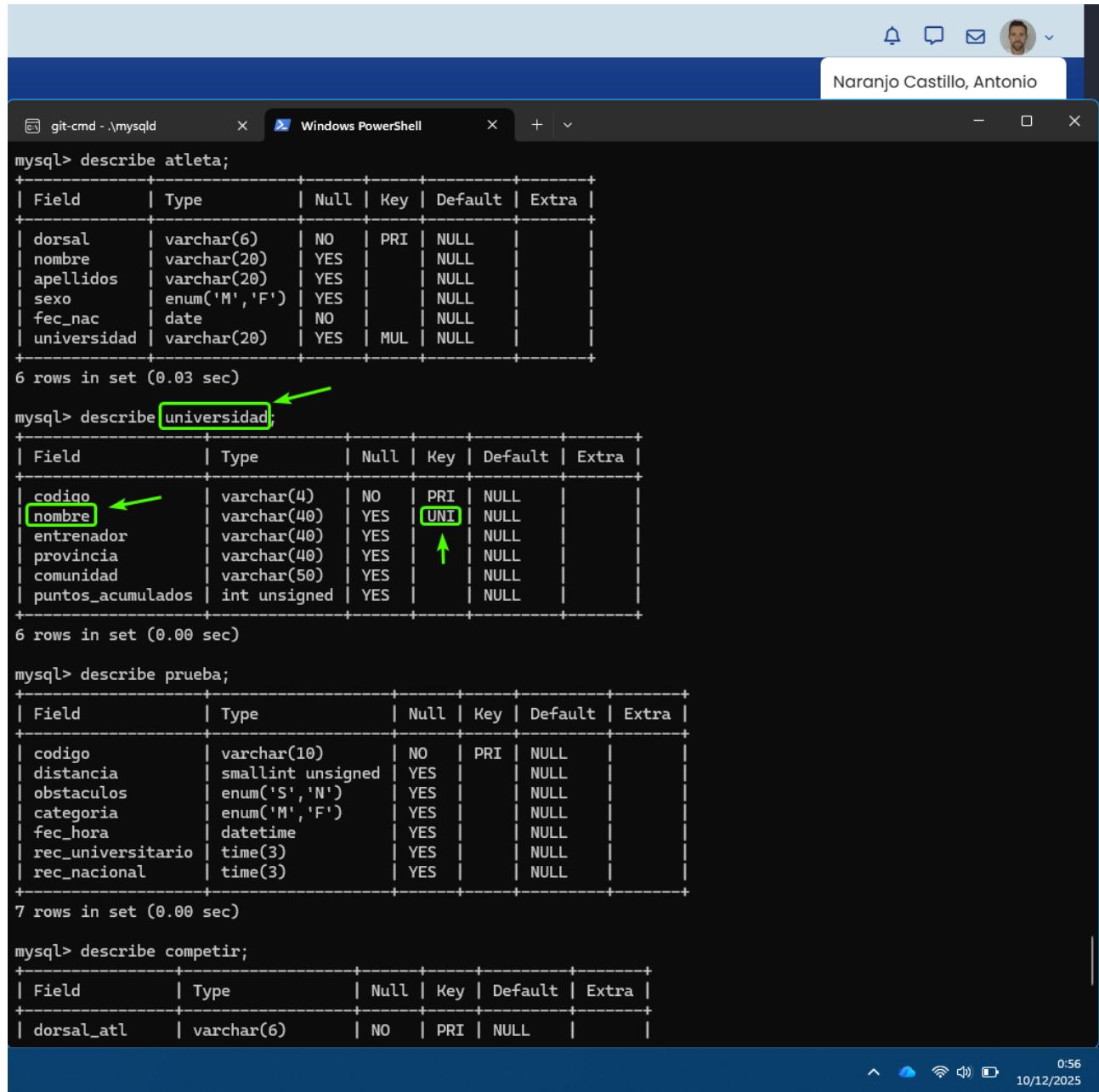
Lista de códigos de pruebas existentes: [100LF, 100LM, 1500F, 1500M, 30000F, 30000M, 400VF, 400VM]  
Lista de posiciones de pruebas existentes: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]  
BUILD SUCCESSFUL (total time: 9 seconds)

90:40/1:22 | 2:04  
INS Unix (LF)  
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.



```
git-cmd - \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)

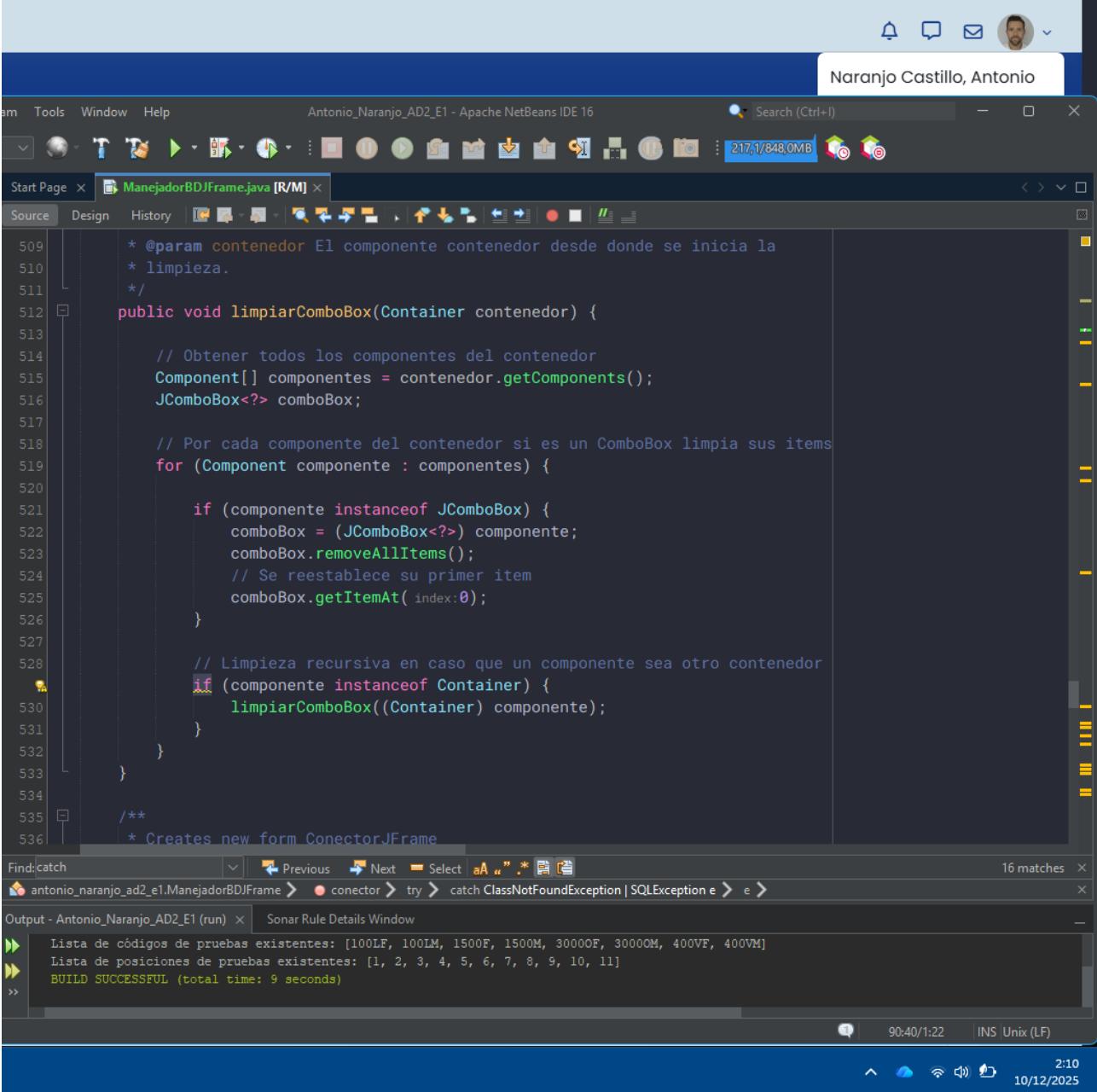
mysql> describe universidad;
+-----+-----+-----+-----+-----+-----+
| Field | Type  | Null | Key  | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| codigo | varchar(4) | NO   | PRI   | NULL    |       |
| nombre  | varchar(40) | YES  | UNI   | NULL    |       |
| entrenador | varchar(40) | YES  |        | 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.



The screenshot shows the Apache NetBeans IDE 16 interface. The code editor window is open with the file `ManejadorBDJFrame.java`. The code implements a recursive method to clear all items from a `JComboBox` and its descendants. The output window at the bottom shows the build log, indicating a successful build with 16 matches found.

```
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.getSelectedItem();
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 | Find Next | Select aA ".\*" | Sonar Rule Details Window | 16 matches | antonio\_naranjo\_ad2\_e1.ManejadorBDJFrame > conector > try > catch ClassNotFoundException | SQLException e > e >

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

```
Lista de códigos de pruebas existentes: [100LF, 100LM, 1500F, 1500M, 3000F, 3000M, 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 | 2:10 | 10/12/2025 | INS Unix (LF)

## 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.

```
git-cmd - \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 |
+-----+
```

- 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 > Windows PowerShell > + >

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 |
| 3000OF |
| 3000OM |
| 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 ascendenteamente.
  - 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.

Naranjo Castillo, Antonio

```
git-cmd -\mysqld x Windows PowerShell x + v
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

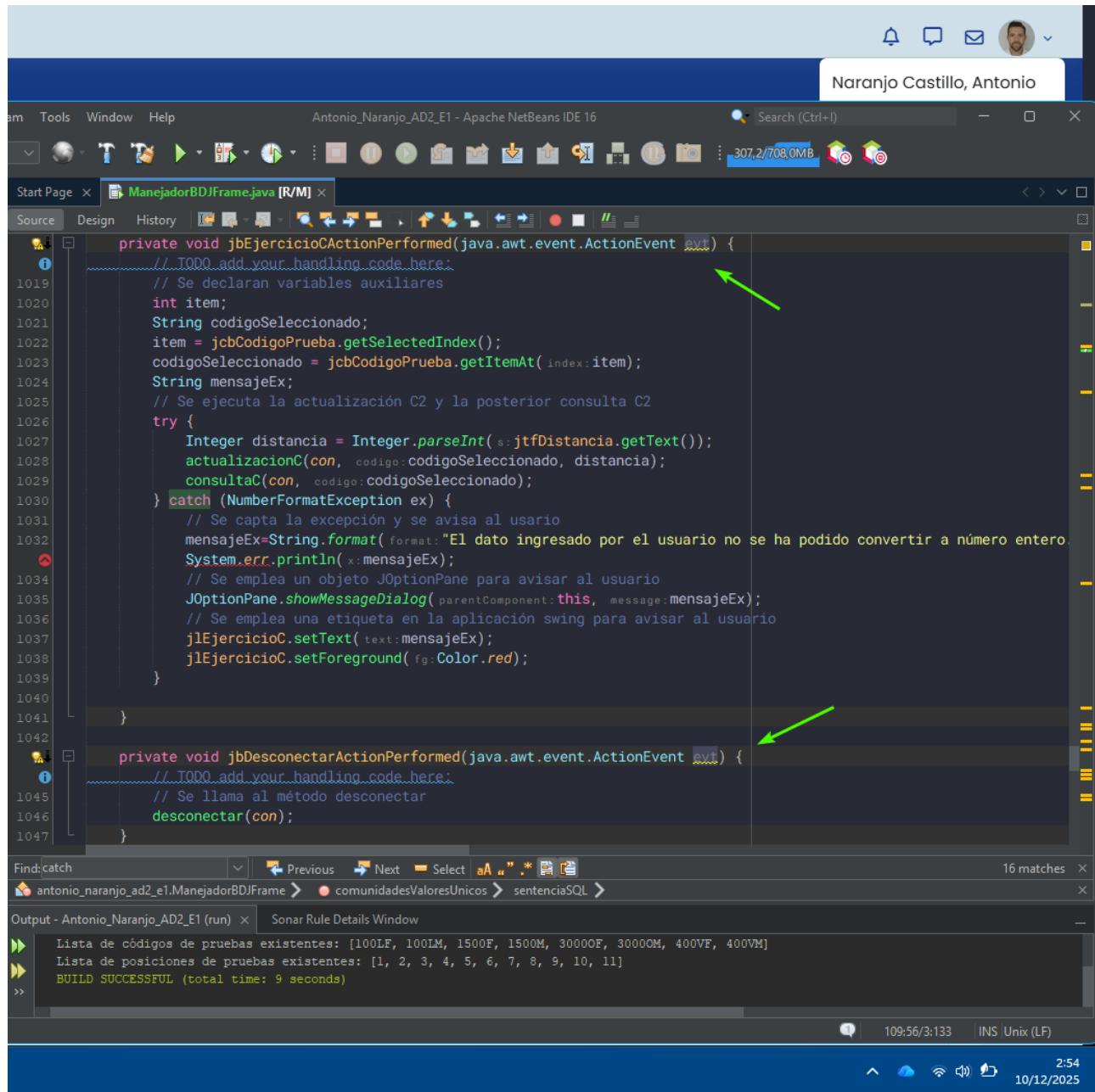
- Botón conectar Ejercicio A.
- Botón consulta Ejercicio B1.
- Botón consulta Ejercicio B2.

```
private void jbConectarActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here.  
    // Se llama al método conector  
    conector();  
}  
  
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);  
}  
  
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);  
}  
  
private void jbEjercicioCActionPerformed(java.awt.event.ActionEvent evt) {
```

Find:catch Previous Next Select aA ".\*" Find... 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, 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)  
109:56/3:133 INS Unix (LF)  
2:53 10/12/2025

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



```
private void jbActualizarActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here  
    // Se declaran variables auxiliares  
    int item;  
    String codigoSeleccionado;  
    item = jcbCodigoPrueba.getSelectedIndex();  
    codigoSeleccionado = jcbCodigoPrueba.getItemAt(index:item);  
    String mensajeEx;  
    // Se ejecuta la actualización C2 y la posterior consulta C2  
    try {  
        Integer distancia = Integer.parseInt(jTextFieldDistancia.getText());  
        actualizarC(con, codigo:codigoSeleccionado, distancia);  
        consultaC(con, codigo:codigoSeleccionado);  
    } catch (NumberFormatException ex) {  
        // Se capta la excepción y se avisa al usuario  
        mensajeEx=String.format("El dato ingresado por el usuario no se ha podido convertir a número entero.  
System.err.println(x:mensajeEx);  
        // Se emplea un objeto JOptionPane para avisar al usuario  
        JOptionPane.showMessageDialog(parentComponent:this, message:mensajeEx);  
        // Se emplea una etiqueta en la aplicación swing para avisar al usuario  
        jLabelEjercicioC.setText(text:mensajeEx);  
        jLabelEjercicioC.setForeground(fg:Color.red);  
    }  
}  
  
private void jbDesconectarActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here  
    // Se llama al método desconectar  
    desconectar(con);  
}
```

Find:catch | Previous | Next | Select | Sonar Rule Details Window | 16 matches | X

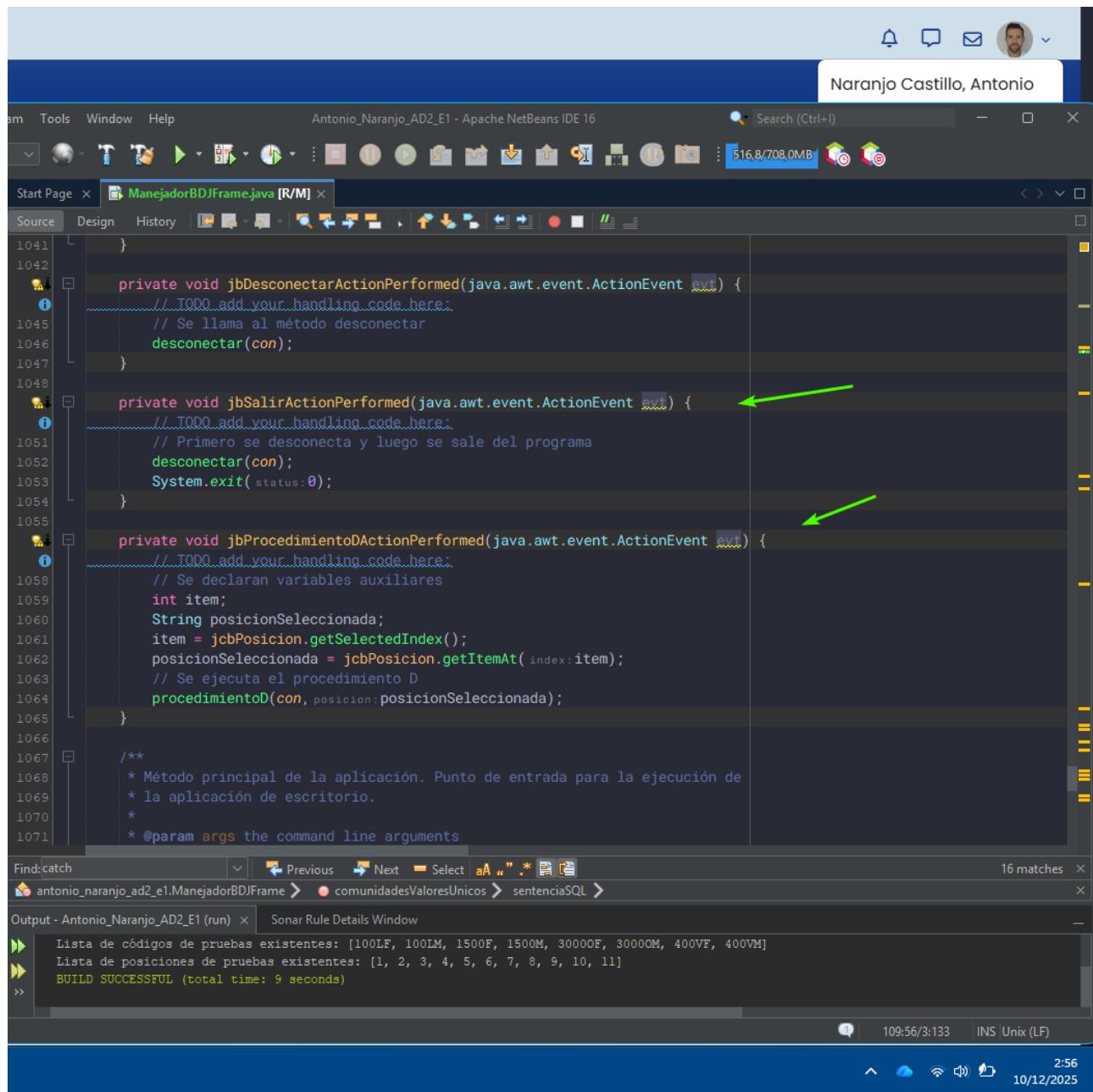
antonio\_naranjo\_ad2\_e1.ManejadorBDJFrame > comunidadesValoresUnicos > sentenciaSQL >

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)
```

109:56/3:133 | INS Unix (LF) | 2:54 | 10/12/2025

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



The screenshot shows the Apache NetBeans IDE interface with the following details:

- Title Bar:** Antonio\_Naranjo\_AD2\_E1 - Apache NetBeans IDE 16
- Toolbar:** Standard NetBeans toolbar with various icons for file operations, search, and project management.
- Status Bar:** Shows memory usage (516,8/708,0MB) and other system information.
- Code Editor:** Displays the `ManejadorBDJFrame.java` file. The code includes methods for handling button actions and executing procedures. Two specific methods are highlighted with green arrows:
  - `jbSalirActionPerformed(java.awt.event.ActionEvent evt)`: Handles the action for exiting the application.
  - `jbProcedimientoDActionPerformed(java.awt.event.ActionEvent evt)`: Handles the action for executing procedure D.
- Output Window:** Shows the build log with the message "BUILD SUCCESSFUL (total time: 9 seconds)".
- Bottom Status:** Shows the date (10/12/2025), time (2:56), and system status indicators.

### 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.

