

Pregunta 1

Tareas a llevar a cabo

1. Implementación de DBAccessor.java
2. Implementación de Exercise1PrintReportOverQuery.java

Configuración de la Conexión: DBAccessor

La clase DBAccessor ha de utilizar para gestionar las conexiones a la base de datos PostgreSQL de manera eficiente.

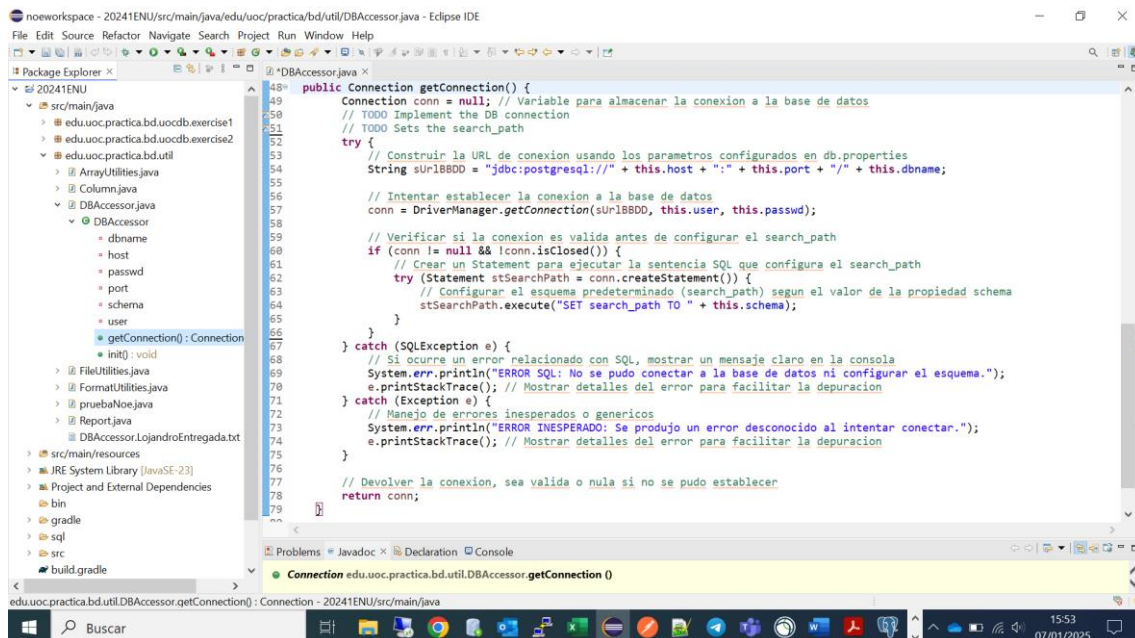
Por un lado se realiza la carga de Configuración desde db.properties. La clase lee los parámetros necesarios para la conexión (como host, port, dbname, user, passwd y schema) desde el archivo db.properties utilizando la clase Properties.

Por otro lado, realiza el establecimiento de la Conexión. La conexión a la base de datos se establece mediante el método getConnection, que utiliza DriverManager.getConnection para generar una conexión con la URL construida a partir de los parámetros cargados.

Lleva a cabo la configuración del search_path, ya que una vez establecida la conexión, se ha de ejecutar un comando SQL (SET search_path TO ...) para configurar el esquema predeterminado.

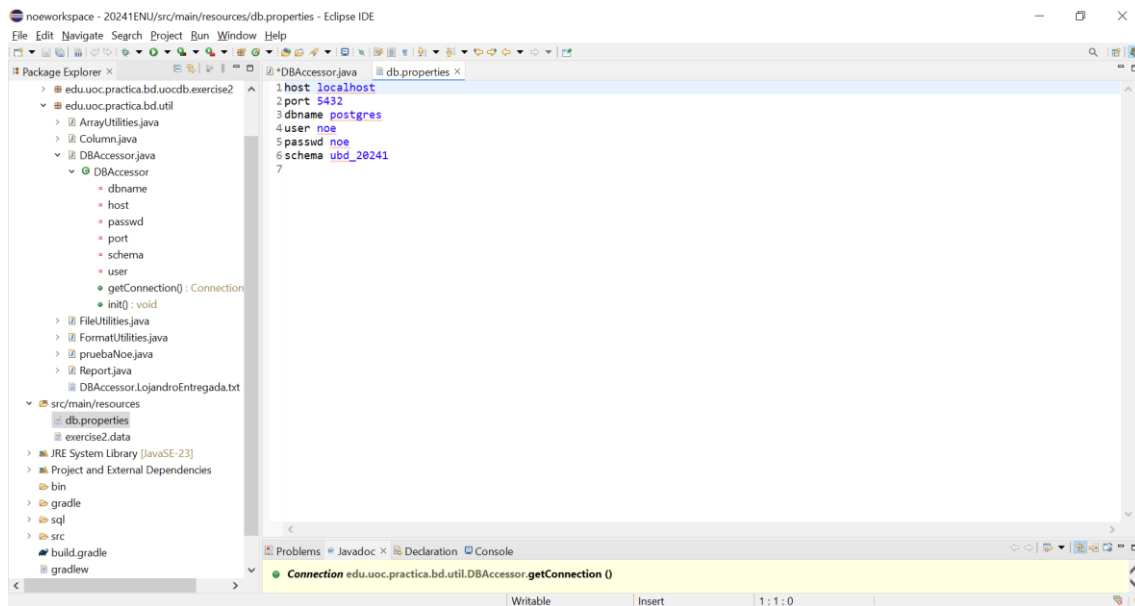
La clase ha de manejar de forma robusta posibles errores de conexión o SQL, mostrando mensajes según lo indicado en el enunciado.

Código resultante, método getConnection:

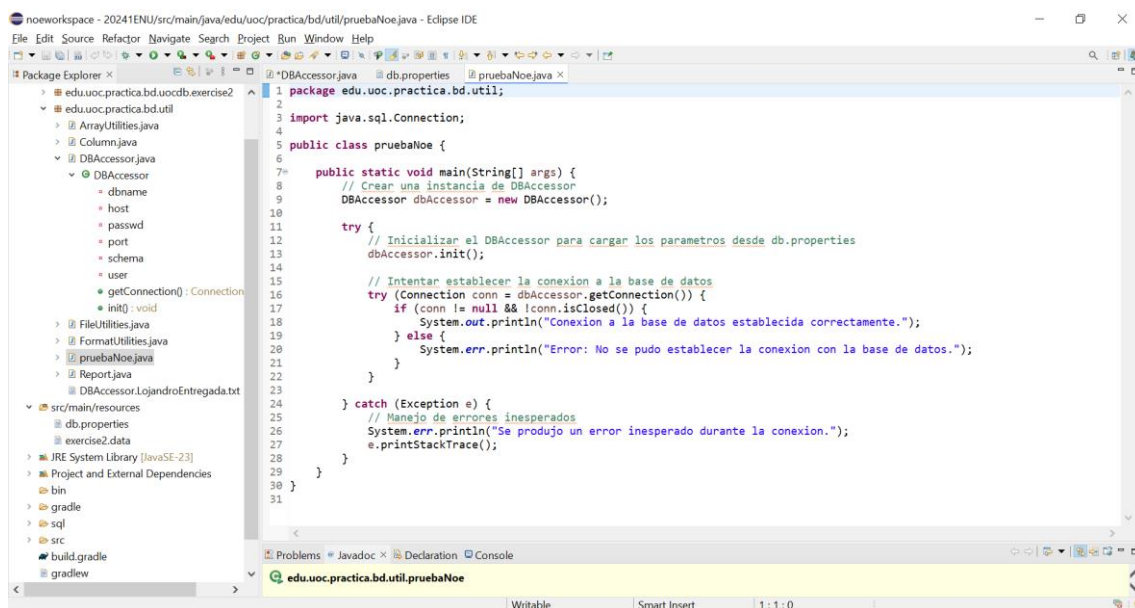


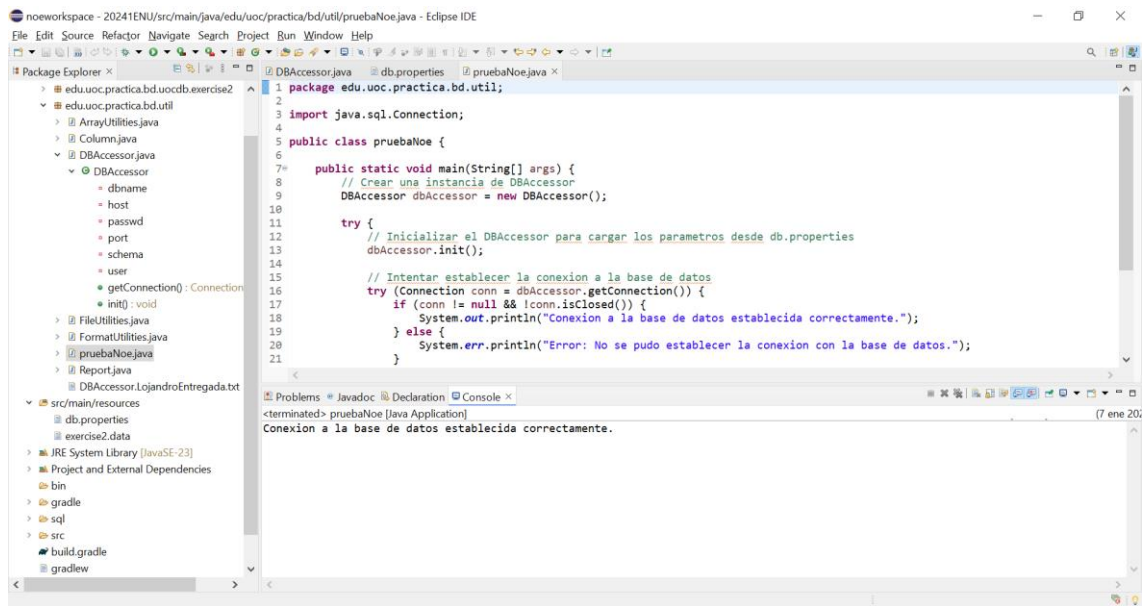
```
48 public Connection getConnection() {
49     Connection conn = null; // Variable para almacenar la conexión a la base de datos
50     // TODO Implement the DB connection
51     // TODO Sets the search_path
52     try {
53         // Construir la URL de conexión usando los parámetros configurados en db.properties
54         String urlBDD = "jdbc:postgresql://" + this.host + ":" + this.port + "/" + this.dbname;
55
56         // Intentar establecer la conexión a la base de datos
57         conn = DriverManager.getConnection(urlBDD, this.user, this.passwd);
58
59         // Verificar si la conexión es válida antes de configurar el search_path
60         if (conn != null && !conn.isClosed()) {
61             // Crear un Statement para ejecutar la sentencia SQL que configura el search_path
62             try (Statement stSearchPath = conn.createStatement()) {
63                 // Configurar el esquema predeterminado (search_path) según el valor de la propiedad schema
64                 stSearchPath.execute("SET search_path TO " + this.schema);
65             }
66         }
67     } catch (SQLException e) {
68         // Si ocurre un error relacionado con SQL, mostrar un mensaje claro en la consola
69         System.err.println("ERROR SQL: No se pudo conectar a la base de datos ni configurar el esquema.");
70         e.printStackTrace(); // Mostrar detalles del error para facilitar la depuración
71     } catch (Exception e) {
72         // Manejo de errores inesperados o genéricos
73         System.err.println("ERROR INESPERADO: Se produjo un error desconocido al intentar conectar.");
74         e.printStackTrace(); // Mostrar detalles del error para facilitar la depuración
75     }
76
77     // Devolver la conexión, sea válida o nula si no se pudo establecer
78     return conn;
79 }
```

El archivo properties tiene el siguiente contenido:



Para validar de manera unitaria este componente he implementado una clase de pruebas que únicamente realiza un getConnection y vemos su resultado:





```

1 package edu.uoc.practica.bd.util;
2
3 import java.sql.Connection;
4
5 public class pruebaNoe {
6
7     public static void main(String[] args) {
8         // Crear una instancia de DBAccessor
9         DBAccessor dbAccessor = new DBAccessor();
10
11         try {
12             // Inicializar el DBAccessor para cargar los parametros desde db.properties
13             dbAccessor.init();
14
15             // Intentar establecer la conexion a la base de datos
16             try (Connection conn = dbAccessor.getConnection()) {
17                 if (conn != null && !conn.isClosed()) {
18                     System.out.println("Conexion a la base de datos establecida correctamente.");
19                 } else {
20                     System.err.println("Error: No se pudo establecer la conexion con la base de datos.");
21                 }
22             }
23         }
24     }
25 }

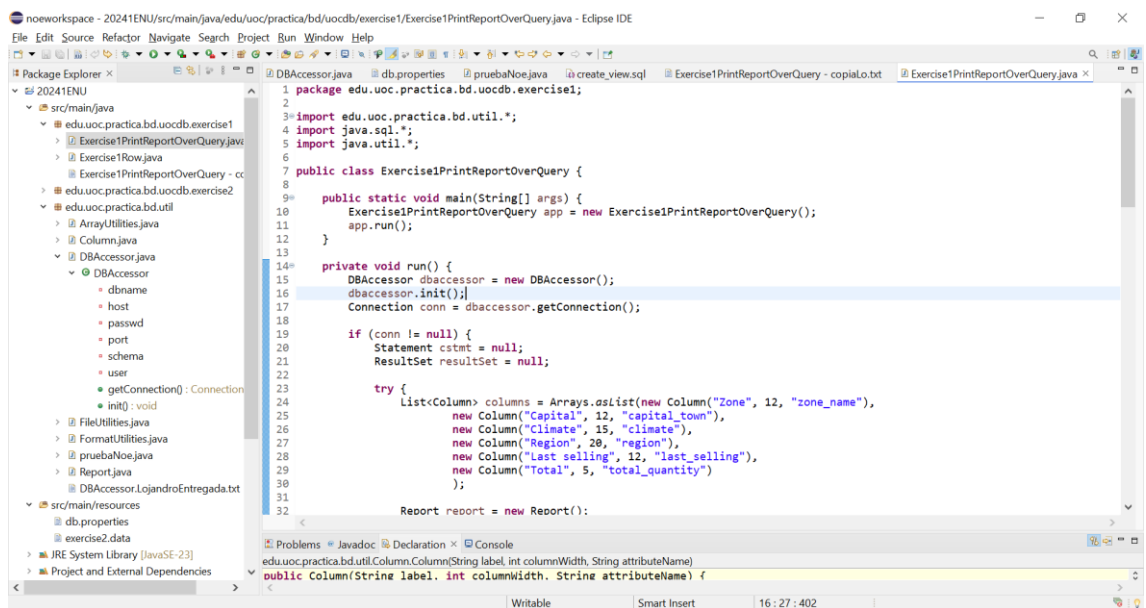
```

Problems | Javadoc | Declaration | Console ×

<terminated> pruebaNoe [Java Application]

Conexion a la base de datos establecida correctamente.

La solución de lo que quedaba pendiente de rellenar a la que se ha llegado es la siguiente:



```

1 package edu.uoc.practica.bd.util;
2
3 import edu.uoc.practica.bd.util.*;
4 import java.sql.*;
5 import java.util.*;
6
7 public class Exercise1PrintReportOverQuery {
8
9     public static void main(String[] args) {
10         Exercise1PrintReportOverQuery app = new Exercise1PrintReportOverQuery();
11         app.run();
12     }
13
14     private void run() {
15         DBAccessor dbAccessor = new DBAccessor();
16         dbAccessor.init();
17         Connection conn = dbAccessor.getConnection();
18
19         if (conn != null) {
20             Statement stmt = null;
21             ResultSet resultSet = null;
22
23             try {
24                 List<Column> columns = Arrays.asList(new Column("Zone", 12, "zone_name"),
25                     new Column("Capital", 12, "capital_town"),
26                     new Column("Climate", 15, "climate"),
27                     new Column("Region", 20, "region"),
28                     new Column("Last selling", 12, "last_selling"),
29                     new Column("Total", 5, "total_quantity")
30                 );
31
32                 Report report = new Report();
33
34                 // ... (rest of the code)
35             } catch (SQLException e) {
36                 e.printStackTrace();
37             }
38         }
39     }
40 }

```

Problems | Javadoc | Declaration | Console

edu.uoc.practica.bd.util.Column(String label, int columnWidth, String attributeName) {

public Column(String label, int columnWidth, String attributeName) {

```

31 Report report = new Report();
32 report.setColumns(columns);
33 List<Object> list = new ArrayList<Object>();
34
35 ///////////////////////////////////////////////////
36 //INI
37
38 //Crear un Statement para ejecutar la consulta SQL
39 stmt = conn.createStatement();
40
41 //Ejecutar la consulta. la consulta usa la vista best_selling_zones
42 resultSet = stmt.executeQuery(
43     "select zone_name, capital_town, climate, region, last_selling, total_quantity " +
44     "from best_selling_zones"
45 );
46
47 /* Realizamos la consulta sobre la vista creada mediante el script entregado:
48 create view best_selling_zones as
49 select z.zone_name,
50 z.capital_town,
51 z.climate,
52 z.region,
53 MAX(o.order_date) as last_selling,
54 sum(l.quantity) AS total_quantity
55 from zone z
56 NATURAL JOIN wine w
57 NATURAL JOIN order_line l
58 NATURAL JOIN customer_order o
59 group by z.zone_name, z.capital_town, z.climate, z.region
60 order by total_quantity DESC
61 LIMIT 5;
62 */

```

```

64 //Crear una variable para guardar cada fila del resultado
65 Exercise1Row ejercicio1Row = null;
66
67 //Leer las filas del resultado una a una
68 while (resultSet.next()) {
69     // Por cada fila, obtener los valores de las columnas y crear un objeto Exercise1Row
70     ejercicio1Row = new Exercise1Row(
71         resultSet.getString("zone_name"), // Columna zone_name
72         resultSet.getString("capital_town"), // Columna capital_town
73         resultSet.getString("climate"), // Columna climate
74         resultSet.getString("region"), // Columna region
75         resultSet.getString("last_selling"), // Columna last_selling
76         resultSet.getLong("total_quantity") // Columna total_quantity
77     );
78
79     // Agregar el objeto Exercise1Row a la lista
80     list.add(ejercicio1Row);
81 }
82
83 //Verificar si se obtuvieron resultados
84 if (list.isEmpty()) {
85     System.out.println("No se encontraron registros en la vista best_selling_zones.");
86 } else {
87     // Llamar a printReport para mostrar la lista de resultados
88     report.printReport(list);
89 }
90
91 //FIN
92 ///////////////////////////////////////////////////
93 }
94 catch (Exception e) {
95     // Mostrar un mensaje si ocurre un error inesperado

```

```

96     System.err.println("ERROR: List not available.\n");
97     e.printStackTrace();
98 }
99 finally {
100     // Cerrar todos los recursos abiertos
101     if (resultSet != null)
102         try {
103             resultSet.close();
104         } catch (Exception ex) {}
105     if (stmt != null)
106         try {
107             stmt.close();
108         } catch (Exception ex) {}
109     if (conn != null)
110         try {
111             conn.close();
112         } catch (Exception ex) {}
113 }
114 }
115 }
116 }
117

```

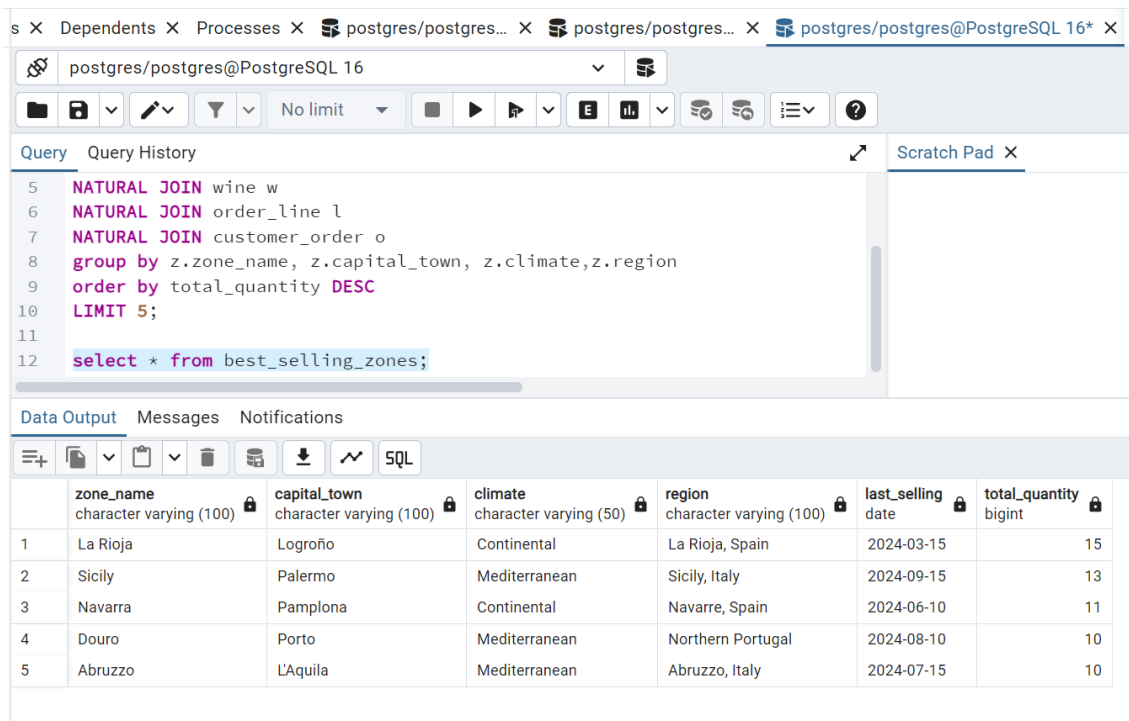
Se ha intentado comentar suficientemente el código para facilitar la corrección. Incluyo, de todos modos, por aquí una breve explicación:

En la clase Exercise1PrintReportOverQuery, se utilizó DBAccessor para conectar con la base de datos y realizar una consulta sobre la vista BEST_SELLING_ZONES. Los pasos específicos fueron:

1. Preparación de la Vista: Se ejecutó el script create_view.sql que define la vista BEST_SELLING_ZONES.
2. Consulta y Procesamiento de Resultados: Con la conexión proporcionada por DBAccessor, se utilizó un Statement para ejecutar la siguiente consulta SQL:
SELECT zone_name, capital_town, climate, region, last_selling, total_quantity
FROM best_selling_zones;
Los datos obtenidos se almacenaron en una lista de objetos Exercise1Row, representando cada fila de la consulta.
3. Presentación del Reporte: La lista resultante fue pasada a la clase Report para imprimir los datos en un formato tabular.
4. Gestión de Recursos y Errores: Se cerraron todos los recursos (Connection, Statement, ResultSet) en un bloque finally para evitar fugas de memoria. En caso de error, se muestra un mensaje genérico ERROR: List not available, cumpliendo con los requisitos del ejercicio.

Esta implementación permite realizar la consulta de manera eficiente y garantizar que la configuración y conexión a la base de datos sean consistentes en todo el programa.

Cuando lo ejecutemos, para saber si todo va bien deberíamos contrastar con lo que se muestra en BBDD directamente:



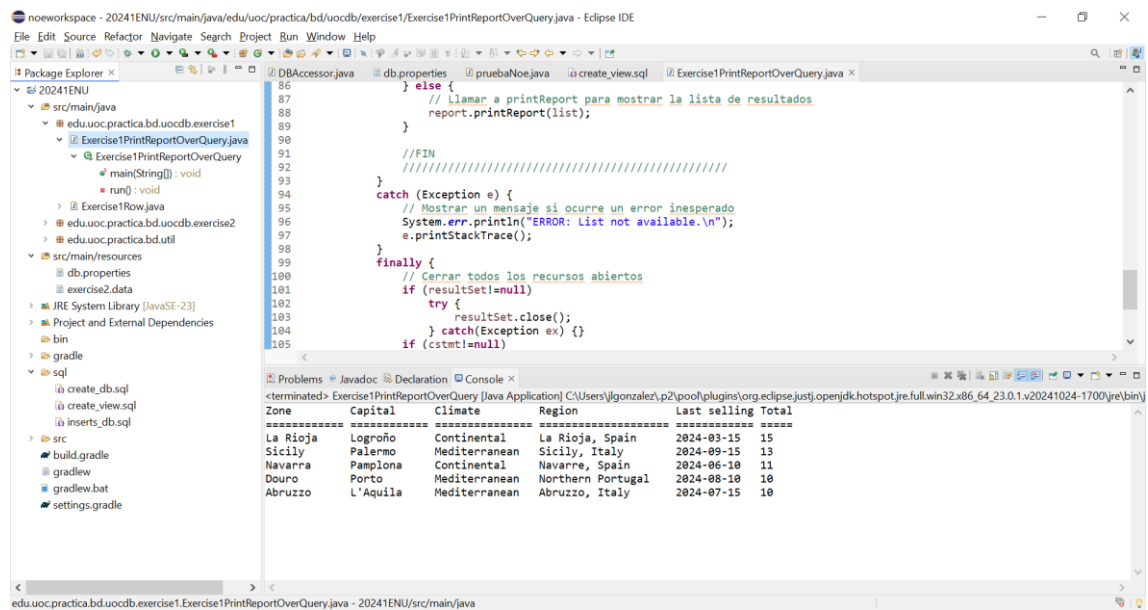
The screenshot shows a PostgreSQL client interface with a query editor and a results table. The query editor contains the following SQL code:

```
5 NATURAL JOIN wine w
6 NATURAL JOIN order_line l
7 NATURAL JOIN customer_order o
8 group by z.zone_name, z.capital_town, z.climate, z.region
9 order by total_quantity DESC
10 LIMIT 5;
11
12 select * from best_selling_zones;
```

The results table displays the following data:

	zone_name character varying (100)	capital_town character varying (100)	climate character varying (50)	region character varying (100)	last_selling date	total_quantity bigint
1	La Rioja	Logroño	Continental	La Rioja, Spain	2024-03-15	15
2	Sicily	Palermo	Mediterranean	Sicily, Italy	2024-09-15	13
3	Navarra	Pamplona	Continental	Navarre, Spain	2024-06-10	11
4	Douro	Porto	Mediterranean	Northern Portugal	2024-08-10	10
5	Abruzzo	L'Aquila	Mediterranean	Abruzzo, Italy	2024-07-15	10

Procedemos a ejecutar nuestra clase Java:



```
noeworkspace - 20241ENU/src/main/java/edu/uoc/practica/bd/uocdb/exercise1/Exercise1PrintReportOverQuery.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer | DBAccessor.java | pruebaNoe.java | create_view.sql | Exercise1PrintReportOverQuery.java
20241ENU
├── src/main/java
│   ├── edu.uoc.practica.bd.uocdb.exercise1
│   │   ├── Exercise1PrintReportOverQuery.java
│   │   │   ├── main(String[]) : void
│   │   │   └── run() : void
│   ├── Exercise1Row.java
│   ├── edu.uoc.practica.bd.uocdb.exercise2
│   ├── edu.uoc.practica.bd.utill
│   ├── src/main/resources
│   │   ├── db.properties
│   │   ├── exercise2.data
│   ├── JRE System Library [JavaSE-23]
│   ├── Project and External Dependencies
│   ├── bin
│   ├── gradle
│   └── sql
│       ├── create_db.sql
│       ├── create_view.sql
│       └── inserts_db.sql
└── src
    ├── build.gradle
    ├── gradlew
    ├── gradlew.bat
    └── settings.gradle

Exercise1PrintReportOverQuery.java
86         } else {
87             // Llamar a printReport para mostrar la lista de resultados
88             report.printReport(list);
89         }
90     }
91     //FIN
92     //////////////////////////////////////
93 }
94
95 catch (Exception e) {
96     // Mostrar un mensaje si ocurre un error inesperado
97     System.err.println("ERROR: List not available.\n");
98     e.printStackTrace();
99 }
100 finally {
101     // Cerrar todos los recursos abiertos
102     if (resultSet!=null)
103         try {
104             resultSet.close();
105         } catch (Exception ex) {}
106     if (cstmt!=null)
```

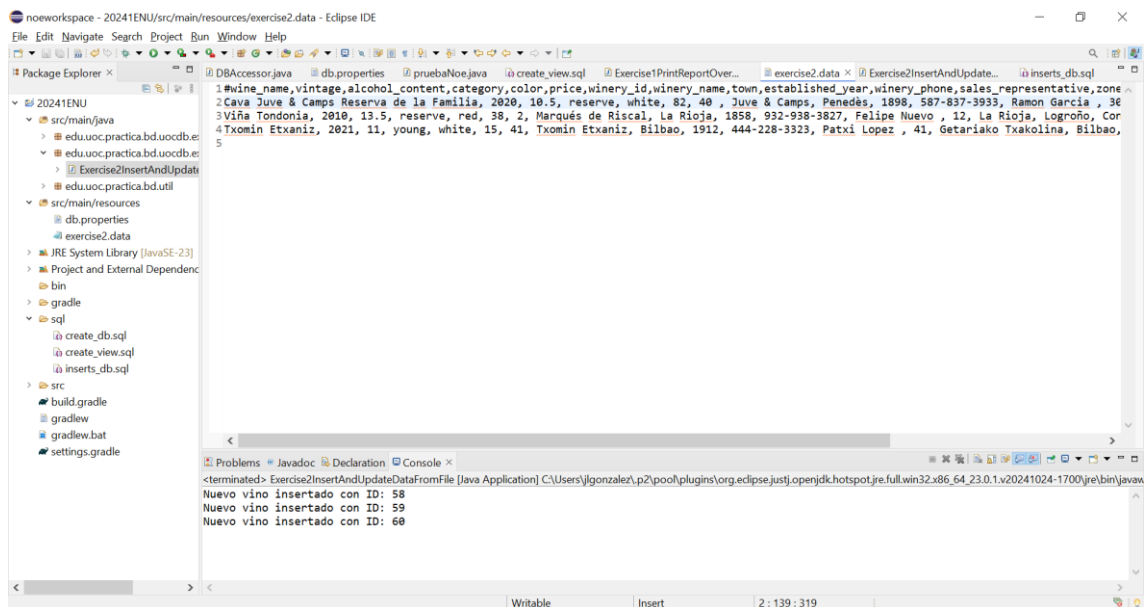
```
<terminated> Exercise1PrintReportOverQuery [Java Application] C:\Users\jgonzalez\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.23.0.1.v20241024-1700\jre\bin\j
=====
Zone      Capital    Climate    Region      Last selling Total
=====
La Rioja  Logroño    Continental La Rioja, Spain 2024-03-15 15
Sicily    Palermo    Mediterranean Sicily, Italy 2024-09-15 13
Navarra   Pamplona   Continental   Navarre, Spain 2024-06-10 11
Douro     Porto      Mediterranean Northern Portugal 2024-08-10 10
Abruzzo   L'Aquila   Mediterranean Abruzzo, Italy 2024-07-15 10
```

edu.uoc.practica.bd.uocdb.exercise1.Exercise1PrintReportOverQuery.java - 20241ENU/src/main/java

Pregunta 2

En esta actividad, se desarrolló un programa en Java que permite procesar un archivo de datos y realizar operaciones de inserción y actualización sobre una base de datos PostgreSQL. El objetivo principal fue gestionar información relacionada con bodegas, zonas y vinos, actualizando los registros existentes o añadiendo nuevos cuando fuese necesario.

El archivo proporcionado, llamado `exercise2.data`, contiene información estructurada que se utiliza para sincronizar los datos de la base de datos. Este archivo incluye detalles como el identificador de la bodega, datos de contacto, información de las zonas geográficas y características de los vinos.



Se ha retocado el primer registro ya que se asociaba un vino a una zona equivocada.

Para abordar esta actividad, se emplearon conceptos clave de programación en Java con JDBC, incluyendo:

- Uso de transacciones (commit y rollback) para garantizar la consistencia de los datos.
- Sentencias preparadas (PreparedStatement) para mejorar la seguridad y el rendimiento de las operaciones.
- Manejo de excepciones para controlar errores durante la ejecución del programa y garantizar la estabilidad del sistema.

Se nos proporciona una clase `Exercise2InsertAndUpdateDataFromFile` a modo de guía con una serie de funcionalidades ya implementadas, como son el tratamiento del fichero de entrada y una serie de métodos auxiliares que utilizaremos para las sentencias sql.

The screenshot shows the Eclipse IDE with the file `Exercise2InsertAndUpdateDataFromFile.java` open. The code defines a `DBAccessor` class with a `getConnection()` method and a `setPSUpdateWinery` method. The `getConnection()` method initializes a `DBAccessor` instance, calls `init()`, and returns the `getConnection()` result. The `setPSUpdateWinery` method takes a `PreparedStatement` and a `List<String>` of row data, and sets values for `winery_phone` and `sales_representative` in the database. The IDE interface includes a Package Explorer on the left showing the project structure, a central editor with the code, and a bottom panel with Problems, Javadoc, Declaration, and Console views.

```
43 }
44
45 DBAccessor dbaccessor = new DBAccessor();
46 dbaccessor.init();
47 Connection conn = dbaccessor.getConnection();
48
49 if (conn == null) {
50     return;
51 }
52
53
54 // TODO Prepare everything before updating or inserting
55
56 try {
57     // TODO Update or insert the wine, winery and zone from every row in file
58
59     // TODO Validate transaction
60 }
61 // TODO Close resources and check exceptions
62 finally {
63 }
64 }
65
66
67 private void setPSUpdateWinery(PreparedStatement updateStatement, List<String> row)
68     throws SQLException {
69     String[] rowArray = (String[]) row.toArray(new String[0]);
70
71     setValueOrNull(updateStatement, 1, getValueIfNotNull(rowArray, 10)); // winery_phone
72     setValueOrNull(updateStatement, 2, getValueIfNotNull(rowArray, 11)); // sales_representative
73     setValueOrNull(updateStatement, 3, ...
74 }
```

El código al que he llegado es el siguiente:

The screenshot shows the Eclipse IDE with the file `Exercise2InsertAndUpdateDataFromFile.java` open. The code defines a `Exercise2InsertAndUpdateDataFromFile` class that inherits from `FileUtilities`. It includes a `main` method and a `run` method. The `run` method reads data from a file named `exercise2.data` and processes it. The IDE interface includes a Package Explorer on the left showing the project structure, a central editor with the code, and a bottom panel with Problems, Javadoc, Declaration, and Console views.

```
1 package edu.uoc.practica.bd.uocdb.exercise2;
2
3 import edu.uoc.practica.bd.util.DBAccessor;
4 import edu.uoc.practica.bd.util.FileUtilities;
5 import java.io.FileNotFoundException;
6 import java.io.IOException;
7 import java.sql.Connection;
8 import java.sql.PreparedStatement;
9 import java.sql.ResultSet;
10 import java.sql.SQLException;
11 import java.sql.Types;
12 import java.util.List;
13
14 public class Exercise2InsertAndUpdateDataFromFile {
15
16     private FileUtilities fileUtilities;
17
18     public Exercise2InsertAndUpdateDataFromFile() {
19         super();
20         fileUtilities = new FileUtilities();
21     }
22
23     public static void main(String[] args) {
24         Exercise2InsertAndUpdateDataFromFile app = new Exercise2InsertAndUpdateDataFromFile();
25         app.run();
26     }
27
28     private void run() {
29         List<List<String>> fileContents = null;
30
31         try {
32             fileContents = fileUtilities.readFileFromClasspath("exercise2.data");
33         }
34     }
35 }
```



```
28 private void run() {
29     List<List<String>> fileContents = null;
30
31     try {
32         fileContents = FileUtilities.readFileFromClasspath("exercise2.data");
33     } catch (FileNotFoundException e) {
34         System.err.println("Error: El archivo de datos no se encontro.");
35         e.printStackTrace();
36     } catch (IOException e) {
37         System.err.println("Error: Ocurrio un problema de entrada/salida al leer el archivo.");
38         e.printStackTrace();
39     }
40
41     if (fileContents == null) {
42         return;
43     }
44
45     DBAccessor dbaccessor = new DBAccessor();
46     dbaccessor.init();
47     Connection conn = dbaccessor.getConnection();
48
49     if (conn == null) {
50         return;
51     }
52
53     ///////////////////////////////////////////////////
54     //INI
55
56     try {
57         // Iniciamos una transaccion
58         conn.setAutoCommit(false);
59
60         // Definimos las consultas SQL para las operaciones necesarias
61         // sqlCheckWinery: verificar si la bodega ya existe en la base de datos
62         String sqlCheckWinery = "select * from winery where winery_id = ?";
63         // sqlUpdateWinery: actualizar los datos de la bodega si ya existe
64         String sqlUpdateWinery = "update winery set winery_phone = ?, sales_representative = ? where winery_id = ?";
65         // sqlInsertWinery: insertar una nueva bodega si no existe
66         String sqlInsertWinery = "insert into winery (winery_id, winery_name, town, established_year, winery_phone, sales_r
67         // sqlSelectZone: verificar si la zona ya existe en la base de datos
68         String sqlSelectZone = "select * from zone where zone_id = ?";
69         // sqlInsertZone: insertar una nueva zona si no existe
70         String sqlInsertZone = "insert into zone (zone_id, zone_name, capital_town, climate, region) values (?, ?, ?, ?, ?)";
71         // sqlInsertWine: insertar un nuevo vino con stock inicial 0
72         String sqlInsertWine = "insert into wine (wine_name, vintage, alcohol_content, category, color, winery_id, zone_id,
73
74         // Creamos los preparedStatement para las consultas definidas anteriormente
75         PreparedStatement checkWineryStmt = conn.prepareStatement(sqlCheckWinery);
76         PreparedStatement updateWineryStmt = conn.prepareStatement(sqlUpdateWinery);
77         PreparedStatement insertWineryStmt = conn.prepareStatement(sqlInsertWinery);
78         PreparedStatement selectZoneStmt = conn.prepareStatement(sqlSelectZone);
79         PreparedStatement insertZoneStmt = conn.prepareStatement(sqlInsertZone);
80         PreparedStatement insertWineStmt = conn.prepareStatement(sqlInsertWine, PreparedStatement.RETURN_GENERATED_KEYS);
81
82         // Iteramos sobre cada fila del archivo de datos para procesar la informacion
83         for (List<String> datosFila : fileContents) {
```

```
52
53 ///////////////////////////////////////////////////
54 //INI
55
56 try {
57     // Iniciamos una transaccion
58     conn.setAutoCommit(false);
59
60     // Definimos las consultas SQL para las operaciones necesarias
61     // sqlCheckWinery: verificar si la bodega ya existe en la base de datos
62     String sqlCheckWinery = "select * from winery where winery_id = ?";
63     // sqlUpdateWinery: actualizar los datos de la bodega si ya existe
64     String sqlUpdateWinery = "update winery set winery_phone = ?, sales_representative = ? where winery_id = ?";
65     // sqlInsertWinery: insertar una nueva bodega si no existe
66     String sqlInsertWinery = "insert into winery (winery_id, winery_name, town, established_year, winery_phone, sales_r
67     // sqlSelectZone: verificar si la zona ya existe en la base de datos
68     String sqlSelectZone = "select * from zone where zone_id = ?";
69     // sqlInsertZone: insertar una nueva zona si no existe
70     String sqlInsertZone = "insert into zone (zone_id, zone_name, capital_town, climate, region) values (?, ?, ?, ?, ?)";
71     // sqlInsertWine: insertar un nuevo vino con stock inicial 0
72     String sqlInsertWine = "insert into wine (wine_name, vintage, alcohol_content, category, color, winery_id, zone_id,
73
74     // Creamos los preparedStatement para las consultas definidas anteriormente
75     PreparedStatement checkWineryStmt = conn.prepareStatement(sqlCheckWinery);
76     PreparedStatement updateWineryStmt = conn.prepareStatement(sqlUpdateWinery);
77     PreparedStatement insertWineryStmt = conn.prepareStatement(sqlInsertWinery);
78     PreparedStatement selectZoneStmt = conn.prepareStatement(sqlSelectZone);
79     PreparedStatement insertZoneStmt = conn.prepareStatement(sqlInsertZone);
80     PreparedStatement insertWineStmt = conn.prepareStatement(sqlInsertWine, PreparedStatement.RETURN_GENERATED_KEYS);
81
82     // Iteramos sobre cada fila del archivo de datos para procesar la informacion
83     for (List<String> datosFila : fileContents) {
```

```
52
53 ///////////////////////////////////////////////////
54 //INI
55
56 try {
57     // Iniciamos una transaccion
58     conn.setAutoCommit(false);
59
60     // Definimos las consultas SQL para las operaciones necesarias
61     // sqlCheckWinery: verificar si la bodega ya existe en la base de datos
62     String sqlCheckWinery = "select * from winery where winery_id = ?";
63     // sqlUpdateWinery: actualizar los datos de la bodega si ya existe
64     String sqlUpdateWinery = "update winery set winery_phone = ?, sales_representative = ? where winery_id = ?";
65     // sqlInsertWinery: insertar una nueva bodega si no existe
66     String sqlInsertWinery = "insert into winery (winery_id, winery_name, town, established_year, winery_phone, sales_r
67     // sqlSelectZone: verificar si la zona ya existe en la base de datos
68     String sqlSelectZone = "select * from zone where zone_id = ?";
69     // sqlInsertZone: insertar una nueva zona si no existe
70     String sqlInsertZone = "insert into zone (zone_id, zone_name, capital_town, climate, region) values (?, ?, ?, ?, ?)";
71     // sqlInsertWine: insertar un nuevo vino con stock inicial 0
72     String sqlInsertWine = "insert into wine (wine_name, vintage, alcohol_content, category, color, winery_id, zone_id,
73
74     // Creamos los preparedStatement para las consultas definidas anteriormente
75     PreparedStatement checkWineryStmt = conn.prepareStatement(sqlCheckWinery);
76     PreparedStatement updateWineryStmt = conn.prepareStatement(sqlUpdateWinery);
77     PreparedStatement insertWineryStmt = conn.prepareStatement(sqlInsertWinery);
78     PreparedStatement selectZoneStmt = conn.prepareStatement(sqlSelectZone);
79     PreparedStatement insertZoneStmt = conn.prepareStatement(sqlInsertZone);
80     PreparedStatement insertWineStmt = conn.prepareStatement(sqlInsertWine, PreparedStatement.RETURN_GENERATED_KEYS);
81
82     // Iteramos sobre cada fila del archivo de datos para procesar la informacion
83     for (List<String> datosFila : fileContents) {
```

```

119 insertWineStmt.executeUpdate(); // Ejecutamos el INSERT
120 try (ResultSet generatedKeys = insertWineStmt.getGeneratedKeys()) {
121     if (generatedKeys.next()) {
122         // Mostramos el ID del nuevo vino insertado
123         System.out.println("Nuevo vino insertado con ID: " + generatedKeys.getInt(1));
124     } else {
125         throw new SQLException("No se pudo recuperar el ID del vino insertado.");
126     }
127 }
128 // Confirmamos los cambios realizados en la base de datos
129 conn.commit();
130 } catch (Exception e) {
131     // Si ocurre un error, revertimos todos los cambios realizados en la transaccion
132     System.err.println("Error: ocurrió un problema durante la transaccion. se revirtieron los cambios.");
133     try {
134         conn.rollback();
135     } catch (Exception ex) {
136         System.err.println("Error: fallo al intentar revertir los cambios.");
137     }
138 } finally {
139     // Cerramos todos los recursos para liberar memoria y evitar problemas
140     try {
141         if (conn != null) conn.close();
142     } catch (Exception ex) {
143         System.err.println("Error: no se pudieron cerrar correctamente los recursos.");
144     }
145 }
146 //FIN
147 ///////////////////////////////////////////////////////////////////

```

```

136 }
137 }
138 //FIN
139 ///////////////////////////////////////////////////////////////////
140 }
141
142 private void setPSCheckWinery(PreparedStatement stmt, List<String> row) throws SQLException {}
143 private void setPSUpdateWinery(PreparedStatement stmt, List<String> row) throws SQLException {}
144 private void setPSInsertWinery(PreparedStatement stmt, List<String> row) throws SQLException {}
145 private void setPSSelectZone(PreparedStatement stmt, List<String> row) throws SQLException {}
146 private void setPSInsertZone(PreparedStatement stmt, List<String> row) throws SQLException {}
147 private void setPSInsertWine(PreparedStatement stmt, List<String> row) throws SQLException {}
148 private Integer getIntegerFromStringOrNull(String integer) {}
149 private Double getDoubleFromStringOrNull(String doubl) {}
150 private String getValueIfNotNull(String[] rowArray, int index) {}
151 private void setValueOrNull(PreparedStatement stmt, int parameterIndex, Integer value) throws SQLException {}
152 private void setValueOrNull(PreparedStatement stmt, int parameterIndex, Double value) throws SQLException {}
153 private void setValueOrNull(PreparedStatement stmt, int parameterIndex, String value) throws SQLException {}
154 }
155
156 ///////////////////////////////////////////////////////////////////
157 //FIN
158 ///////////////////////////////////////////////////////////////////

```

Comento nuevamente lo más relevante para facilitar la corrección:

- Inicio de la Transacción: Antes de procesar el archivo, se configura la conexión para trabajar en modo transaccional (`conn.setAutoCommit(false)`), lo que permite agrupar las operaciones como una unidad atómica. Esto garantiza que, en caso de error, todos los cambios realizados hasta ese punto puedan deshacerse mediante un rollback.
- Preparación de Sentencias SQL:
Se definieron varias sentencias SQL en forma de `PreparedStatement` para las diferentes operaciones necesarias:
 - o Verificación de registros existentes: `SELECT` para comprobar si una bodega o una zona ya existe.
 - o Actualización de registros existentes: `UPDATE` para modificar datos de contacto o representantes de ventas en bodegas existentes.
 - o Inserción de nuevos registros: `INSERT` para agregar nuevas bodegas, zonas y vinos con un stock inicial de 0.

- Procesamiento de Datos:
El programa itera sobre cada fila del archivo exercise2.data y realiza las siguientes acciones:
 - o Bodega:
Comprueba si la bodega ya existe mediante un SELECT.
Si existe, actualiza los datos correspondientes con un UPDATE.
Si no existe, inserta una nueva bodega con un INSERT.
 - o Zona:
Comprueba si la zona ya existe mediante un SELECT.
Si no existe, la inserta en la base de datos con un INSERT.
 - o Vino:
Inserta un nuevo registro de vino con un stock inicial de 0 utilizando un INSERT.
Si se inserta correctamente, muestra el nuevo identificador del vino generado automáticamente.
- Gestión de Errores: En caso de que ocurra un error durante el procesamiento, se captura la excepción y se revierte la transacción completa con un rollback. Esto asegura que la base de datos quede en un estado consistente, sin cambios parciales.
- Confirmación de la Transacción: Si todas las operaciones se completan sin errores, se confirman los cambios mediante un commit, consolidando las modificaciones en la base de datos.
- Liberación de Recursos Finalmente, se cierran todos los recursos abiertos, incluidos los PreparedStatement y la conexión (Connection), para liberar memoria y evitar fugas de recursos.

Las sentencias sql utilizadas en este ejercicio son:

- o Verificar si una bodega ya existe
`SELECT * FROM winery WHERE winery_id = ?;`
- o Actualizar datos de una bodega existente
`UPDATE winery
SET winery_phone = ?, sales_representative = ?
WHERE winery_id = ?;`
- o Insertar una nueva bodega
`INSERT INTO winery (winery_id, winery_name, town, established_year,
winery_phone, sales_representative)
VALUES (?, ?, ?, ?, ?, ?);`
- o Verificar si una zona ya existe
`SELECT * FROM zone WHERE zone_id = ?;`
- o Insertar una nueva zona
`INSERT INTO zone (zone_id, zone_name, capital_town, climate, region)
VALUES (?, ?, ?, ?, ?);`
- o Insertar un nuevo
`INSERT INTO wine (wine_name, vintage, alcohol_content, category, color,
winery_id, zone_id, price, stock)
VALUES (?, ?, ?, ?, ?, ?, ?, ?, 0);`

Para demostrar el buen funcionamiento, se ejecuta la clase

```

121 conn.commit();
122 } catch (Exception e) {
123     // Si ocurre un error, revertimos todos los cambios realizados en la transaccion
124     System.err.println("Error: ocurrió un problema durante la transaccion. se revirtieron los cambios.");
125     try {
126         conn.rollback();
127     } catch (Exception ex) {
128         System.err.println("Error: fallo al intentar revertir los cambios.");
129     }
130 } finally {
131     // Cerramos todos los recursos para liberar memoria y evitar problemas
132     try {
133         if (conn != null) conn.close();
134     } catch (Exception ex) {
135         System.err.println("Error: no se pudieron cerrar correctamente los recursos.");
136     }
137 }
138
Problems | Javadoc | Declaration | Console x
<terminated> Exercise2InsertAndUpdateDataFromFile [Java Application]
Nuevo vino insertado con ID: 58
Nuevo vino insertado con ID: 59
Nuevo vino insertado con ID: 60

```

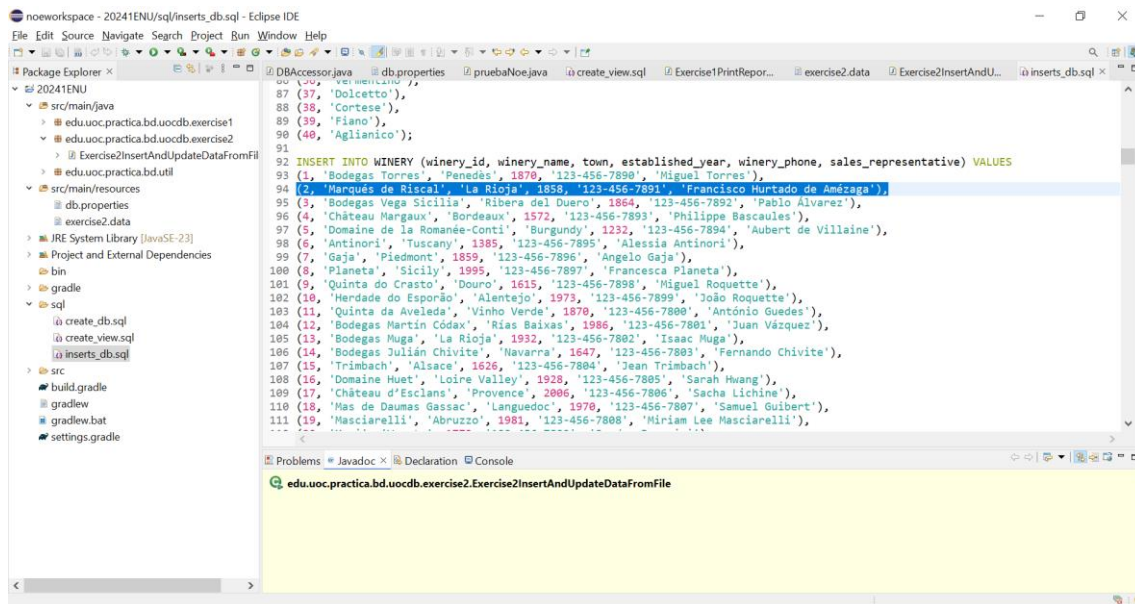
Si pasamos a revisar la base de datos, vemos que los vinos se insertaron:

wine_id [PK] integer	wine_name character varying (100)	vintage integer	alcohol_content numeric (4,2)	category character varying (50)	color character varying (20)	winery_id integer	zone_id integer	status integer
1	58 Cava Juve & Camps Reserva de la Familia	2020	10.50	reserve	white	40	30	
2	59 Viña Tondonia	2010	13.50	reserve	red	2	12	
3	60 Txomin Etxaniz	2021	11.00	young	white	41	41	

En lo que respecta a la bodegas afectadas, vamos a ver que datos han quedado tras la ejecución:

winery_id [PK] integer	winery_name character varying (100)	town character varying (50)	established_year integer	winery_phone character varying (13)	sales_representative character varying (100)
1	40 Juve & Camps	Penedès	1898	587-837-3933	Ramon Garcia
2	2 Marqués de Riscal	La Rioja	1858	932-938-3827	Felipe Nuevo
3	41 Txomin Etxaniz	Bilbao	1912	444-228-3323	Patxi Lopez

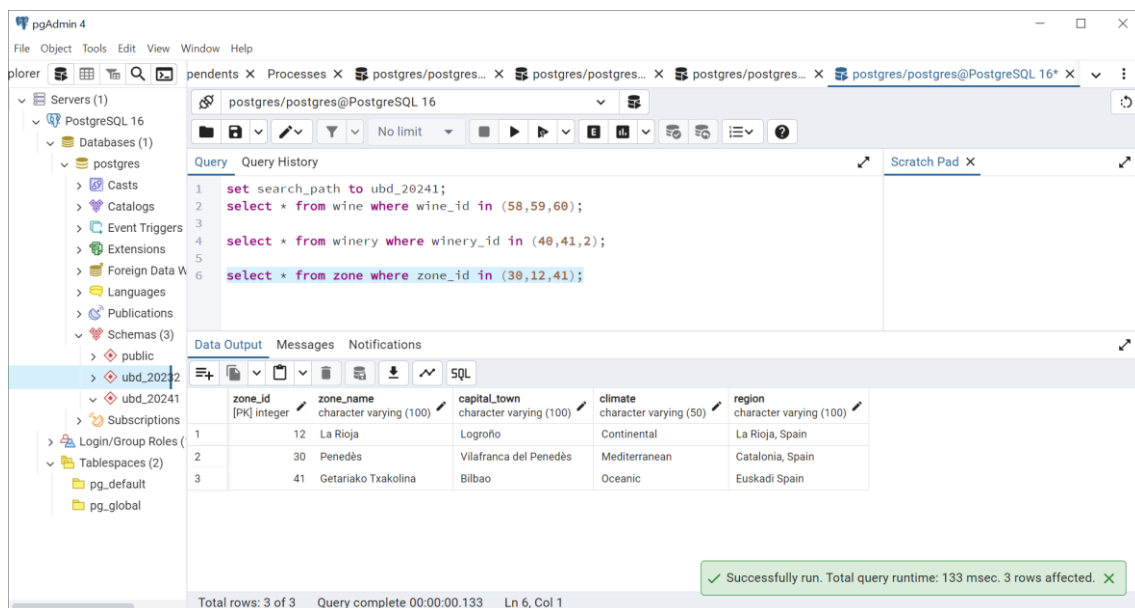
La bodega 2, tal y como se ve en el fichero de inserts, existía



Y como se ve se ha actualizado los datos de teléfono y sales_representative.

Las bodegas 40 y 41 se insertaron correctamente, ya que no estaban incluidas en el fichero de inserts de creación.

En cuanto a las zonas, tras la ejecución de mi código solución queda así:



Para finalizar, se va a modificar el código para mostrar por consola cada una de las acciones sobre bbdd y que las ejecuciones de prueba que puedan llevarse a cabo para la corrección puedan ser más sencillas.

Recreamos la base de datos y volvemos a ejecutar y verificando el código:

noeworkspace - 20241ENU/src/main/java/edu/uoc/practica/bd/uocdb/exercise2/Exercise2InsertAndUpdateDataFromFile.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- 20241ENU
 - src/main/java
 - edu.uoc.practica.bd.uocdb.e
 - Exercise2InsertAndUpdateDataFromFile.java
- src/main/resources
 - db.properties
 - exercise2.data
- JRE System Library [JavaSE-23]
- Project and External Dependencies
 - bin
 - gradle
 - create_db.sql
 - create_view.sql
 - inserts_db.sql

DBAccessor.java db.properties pruebaNoe.java create_view.sql Exercise1PrintReportOver... exercise2.data Exercise2InsertAndUpdate... inserts_db.sql

```
98 }
99 rswinery.close(); // Cerramos el ResultSet para liberar recursos
100
101 // 2. Verificamos si la zona ya existe
102 setPSselectZone(selectZoneStmt, datosFila); // Configuramos los parametros del SELECT
103 if (!selectZoneStmt.executeQuery().next()) {
104     // Si la zona no existe, la insertamos en la base de datos
105     setPSinsertZone(insertZoneStmt, datosFila); // Configuramos los parametros del INSERT
106     insertZoneStmt.executeUpdate(); // Ejecutamos el INSERT
107     System.out.println("\tAlta zone con zone_id "+datosFila.get(12));
108 }
109
110 // 3. Insertamos un nuevo vino
111 setPSinsertWine(insertWineStmt, datosFila); // Configuramos los parametros del INSERT
112 insertWineStmt.executeUpdate(); // Ejecutamos el INSERT
113 try (ResultSet generatedKeys = insertWineStmt.getGeneratedKeys()) {
114     if (generatedKeys.next()) {
115         // Mostramos el ID del nuevo vino insertado
116         System.out.println("\tNuevo vino insertado con ID: " + generatedKeys.getInt(1));
117     } else {
118     }
119 }
```

Problems Javadoc Declaration Console

<terminated> Exercise2InsertAndUpdateDataFromFile [Java Application]

- Registro a tratar Cava Juve & Camps Reserva de la Familia
 - Alta winery con winery_id 40
 - Nuevo vino insertado con ID: 58
- Registro a tratar Viña Tondonia
 - Actualizacion winery con winery_id 2
 - Nuevo vino insertado con ID: 59
- Registro a tratar Txomin Etzaniz
 - Alta winery con winery_id 41
 - Alta zone con zone_id 41
 - Nuevo vino insertado con ID: 60

edu.uoc.practica.bd.uocdb.exercise2.Exercise2InsertAndUpdateDataFromFile.java - 20241ENU/src/main/java