Introduction to Databases

PR3: Building river bridges: accessing a relational database from application programs

Name	Daniel
Surnames	Maestre Sánchez
UOC Username	dmaestresan

Contenido

Exercise 1

DBAccesor class

```
package edu.uoc.practica.bd.util;

import java.io.IOException;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Properties;

public class DBAccessor {

    private String dbname;
    private String host;
    private String port;
    private String user;
    private String passwd;
    private String schema;

    /**
     * Initializes the class loading the database properties file and assigns values
to the instance
     * variables.
     *
     * @throws RuntimeException Properties file could not be found.
     */
    public void init() {
        Properties prop = new Properties();
        InputStream propStream =
this.getClass().getClassLoader().getResourceAsStream("db.properties");

        try {
            prop.load(propStream);
            this.host = prop.getProperty("host");
            this.port = prop.getProperty("port");
            this.dbname = prop.getProperty("dbname");
            this.user = prop.getProperty("user");
            this.passwd = prop.getProperty("passwd");
            this.schema = prop.getProperty("schema");
```

```
        } catch (IOException e) {
            String message = "ERROR: db.properties file could not be found";
            System.err.println(message);
            throw new RuntimeException(message, e);
        }
    }

    /**
     * Obtains a {@link Connection} to the database, based on the values of the
     * <code>db.properties</code> file.
     *
     * @return DB connection or null if a problem occurred when trying to connect.
     */
    public Connection getConnection() {
        Connection conn = null;

        try {
            // TODO Implement the DB connection
            // Load PostgreSQL JDBC driver
            Class.forName("org.postgresql.Driver");

            // Build the connection URL
            String url = String.format("jdbc:postgresql://%s:%s/%s", this.host,
this.port, this.dbname);

            // Connect to the database
            conn = DriverManager.getConnection(url, this.user, this.passwd);

            // TODO Sets the search_path
            // Set the search_path to the schema
            try (Statement stmt = conn.createStatement()) {
                stmt.execute("SET search_path TO " + this.schema);
            }

            System.out.println("##########Successful connection to the
database.##########");
        } catch (ClassNotFoundException e) {
            System.err.println("ERROR: JDBC driver for PostgreSQL not found.");
            e.printStackTrace();
        } catch (SQLException e) {
            System.err.println("ERROR: Could not connect to the database.");
            e.printStackTrace();
        }
```

```
        return conn;
    }

}
```

Db.properties

```
host localhost
#That is the port to connect the specific server and version 16 in my compueter
port 5433
dbname PR3
user postgres
passwd newpassword
schema ubd_20241
```

Exercise1PrintReportOverQuery

```
package edu.uoc.practica.bd.uocdb.exercise1;

import edu.uoc.practica.bd.util.Column;
import edu.uoc.practica.bd.util.DBAccessor;
import edu.uoc.practica.bd.util.Report;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class Exercise1PrintReportOverQuery {

    public static void main(String[] args) {
        Exercise1PrintReportOverQuery app = new Exercise1PrintReportOverQuery();
        app.run();
    }
```

```
    private void run() {
        DBAccessor dbaccessor = new DBAccessor();
        dbaccessor.init();
        Connection conn = dbaccessor.getConnection();

        if (conn != null) {
            Statement stmt = null;
            ResultSet resultSet = null;

            try {
                List<Column> columns = Arrays.asList(
                        new Column("Zone", 12, "zone_name"),
                        new Column("Capital", 12, "capital_town"),
                        new Column("Climate", 15, "climate"),
                        new Column("Region", 20, "region"),
                        new Column("Last selling", 12, "last_selling"),
                        new Column("Total", 5, "total_quantity")
                );

                Report report = new Report();
                report.setColumns(columns);
                List<Object> list = new ArrayList<>();

                // TODO Execute SQL sentence
                String sql = "SELECT zone_name, capital_town, climate, region,
last_selling, total_quantity FROM best_selling_zones";
                stmt = conn.createStatement();
                resultSet = stmt.executeQuery(sql);

                // TODO Loop over results and get the main values
                while (resultSet.next()) {
                    Exercise1Row row = new Exercise1Row(
                            resultSet.getString("zone_name"),
                            resultSet.getString("capital_town"),
                            resultSet.getString("climate"),
                            resultSet.getString("region"),
                            resultSet.getString("last_selling"),
                            resultSet.getLong("total_quantity")
                    );
                    list.add(row);
                }

                // TODO End loop
                report.printReport(list);
```

```
            } catch (SQLException e) {
                System.err.println("ERROR: List not available");
                e.printStackTrace();
            } finally {
                // TODO Close All resources
                try {
                    if (resultSet != null) resultSet.close();
                    if (stmt != null) stmt.close();
                    conn.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

Result

```
PS C:\Users\maest\Desktop\UOC\5 Semester\Introduction to Databases\PR\3\20241ENU-1>
c:; cd 'c:\Users\maest\Desktop\UOC\5 Semester\Introduction to Databases\PR\3\20241EN
U-1'; & 'C:\Program Files\Java\jdk-17\bin\java.exe' '@C:\Users\maest\AppData\Local\T
emp\cp_cwt2zreuj9wxx1atv8coil96m.argfile' 'edu.uoc.practica.bd.uocdb.exercise1.Exerc
ise1PrintReportOverQuery'
##########Successful connection to the database.##########
Zone         Capital      Climate         Region               Last selling Total
============ ============ =============== ==================== ============ =====
La Rioja     Logroto      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
```

Exercise 2

Exercise2InsertAndUpdateDataFromFile

```
package edu.uoc.practica.bd.uocdb.exercise2;

import edu.uoc.practica.bd.util.DBAccessor;
import edu.uoc.practica.bd.util.FileUtilities;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Types;
import java.util.List;

public class Exercise2InsertAndUpdateDataFromFile {

    private FileUtilities fileUtilities;

    public Exercise2InsertAndUpdateDataFromFile() {
        super();
        fileUtilities = new FileUtilities();
    }

    public static void main(String[] args) {
        Exercise2InsertAndUpdateDataFromFile app = new
Exercise2InsertAndUpdateDataFromFile();
        app.run();
    }

    private void run() {
        List<List<String>> fileContents = null;

        try {
            fileContents = fileUtilities.readFileFromClasspath("exercise2.data");
        } catch (FileNotFoundException e) {
            System.err.println("ERROR: File not found");
            e.printStackTrace();
        } catch (IOException e) {
            System.err.println("ERROR: I/O error");
```

```
            e.printStackTrace();
        }

        if (fileContents == null) {
            return;
        }

        DBAccessor dbaccessor = new DBAccessor();
        dbaccessor.init();
        Connection conn = dbaccessor.getConnection();

        if (conn == null) {
            return;
        }

        // The aim of the following program fragment is to ensure the robustness of
the program, because your database inserts/updates may have errors.  What is done is
to confirm the operations at the end of the program if everything went well.

        try {
            conn.setAutoCommit(false); // Disable autoCommit to handle transactions
manually
        } catch (SQLException e) {
            System.err.println("ERROR: Unable to disable autoCommit.");
            e.printStackTrace();
            return;
        }

        // TODO Prepare everything before updating or inserting
        String updateWinerySQL = "UPDATE WINERY SET winery_phone = ?,
sales_representative = ? WHERE winery_id = ?";
        String insertWinerySQL = "INSERT INTO WINERY (winery_id, winery_name, town,
established_year, winery_phone, sales_representative) VALUES (?, ?, ?, ?, ?, ?)";
        String selectZoneSQL = "SELECT zone_id FROM ZONE WHERE zone_id = ?";
        String insertZoneSQL = "INSERT INTO ZONE (zone_id, zone_name, capital_town,
climate, region) VALUES (?, ?, ?, ?, ?)";
        String insertWineSQL = "INSERT INTO WINE (wine_name, vintage,
alcohol_content, category, color, winery_id, zone_id, stock, price) VALUES (?, ?, ?,
?, ?, ?, ?, 0, ?)";//As in the input variables(exercise2.data) there is no stock
variable and it cannot be null, I have set it to 0 to guarantee a good functioning.

        try (
            PreparedStatement psUpdateWinery =
conn.prepareStatement(updateWinerySQL);
```

```
            PreparedStatement psInsertWinery =
conn.prepareStatement(insertWinerySQL);
            PreparedStatement psSelectZone = conn.prepareStatement(selectZoneSQL);
            PreparedStatement psInsertZone = conn.prepareStatement(insertZoneSQL);
            PreparedStatement psInsertWine = conn.prepareStatement(insertWineSQL)
        ) {
            // TODO Update or insert the wine, winery and zone from every row in
file
            for (List<String> row : fileContents) {
                // Update winery
                setPSUpdateWinery(psUpdateWinery, row);
                int rowsUpdated = psUpdateWinery.executeUpdate();
                if (rowsUpdated > 0) {
                    System.out.println("Winery updated: " +
getValueIfNotNull(row.toArray(new String[0]), 7));
                } else {
                    // Insert winery if not updated
                    setPSInsertWinery(psInsertWinery, row);
                    psInsertWinery.executeUpdate();
                    System.out.println("New winery inserted: " +
getValueIfNotNull(row.toArray(new String[0]), 7));
                }


                // Insert winery if not updated
                if (rowsUpdated == 0) {
                    setPSInsertWinery(psInsertWinery, row);
                    psInsertWinery.executeUpdate();
                }

                // Check if zone exists
                setPSSelectZone(psSelectZone, row);
                ResultSet rsZone = psSelectZone.executeQuery();
                if (!rsZone.next()) {
                    setPSInsertZone(psInsertZone, row);
                    psInsertZone.executeUpdate();
                    System.out.println("New zone inserted: " +
getValueIfNotNull(row.toArray(new String[0]), 13));
                } else {
                    System.out.println("Zone already exists: " +
getValueIfNotNull(row.toArray(new String[0]), 13));
                }
```

```
                // Insert wine
                setPSInsertWine(psInsertWine, row);
                psInsertWine.executeUpdate();
                System.out.println("New wine inserted: " +
getValueIfNotNull(row.toArray(new String[0]), 0));

            }

            // TODO Validate transaction
            conn.commit();
            System.out.println("##########Transaction committed
successfully.##########");
        } catch (SQLException e) {
            System.err.println("ERROR: Transaction failed, rolling back.");
            e.printStackTrace();
            if (conn != null) {
                try {
                    conn.rollback();
                } catch (SQLException rollbackEx) {
                    rollbackEx.printStackTrace();
                }
            }
        }
        // TODO Close resources and check exceptions
        finally {
            if (conn != null) {
                try {
                    conn.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        }
    }


    private void setPSUpdateWinery(PreparedStatement updateStatement, List<String>
row)
            throws SQLException {
        String[] rowArray = (String[]) row.toArray(new String[0]);

        setValueOrNull(updateStatement, 1, getValueIfNotNull(rowArray, 10)); //
winery_phone
```

```
        setValueOrNull(updateStatement, 2, getValueIfNotNull(rowArray, 11)); //
sales_representative
        setValueOrNull(updateStatement, 3,
                getIntegerFromStringOrNull(getValueIfNotNull(rowArray, 6))); //
winery_id
    }

    private void setPSInsertWinery(PreparedStatement insertStatement, List<String>
row)
            throws SQLException {
        String[] rowArray = (String[]) row.toArray(new String[0]);

        setValueOrNull(insertStatement, 1,
                getIntegerFromStringOrNull(getValueIfNotNull(rowArray, 6)));  //
winery_id
        setValueOrNull(insertStatement, 2, getValueIfNotNull(rowArray, 7)); //
winery_name
        setValueOrNull(insertStatement, 3, getValueIfNotNull(rowArray, 8)); // town
        setValueOrNull(insertStatement, 4,
                getIntegerFromStringOrNull(getValueIfNotNull(rowArray, 9)));  //
established_year
        setValueOrNull(insertStatement, 5, getValueIfNotNull(rowArray, 10));  //
winery_phone
        setValueOrNull(insertStatement, 6, getValueIfNotNull(rowArray, 11));  //
sales_representative
    }

    private void setPSSelectZone(PreparedStatement updateStatement, List<String>
row)
            throws SQLException {
        String[] rowArray = (String[]) row.toArray(new String[0]);

        setValueOrNull(updateStatement, 1,
                getIntegerFromStringOrNull(getValueIfNotNull(rowArray, 12))); //
zone_id
    }

    private void setPSInsertZone(PreparedStatement insertStatement, List<String>
row)
            throws SQLException {
        String[] rowArray = (String[]) row.toArray(new String[0]);

        setValueOrNull(insertStatement, 1,
```

```
                getIntegerFromStringOrNull(getValueIfNotNull(rowArray, 12)));  //
zone_id
        setValueOrNull(insertStatement, 2, getValueIfNotNull(rowArray, 13)); //
zone_name
        setValueOrNull(insertStatement, 3, getValueIfNotNull(rowArray, 14)); //
capital_town
        setValueOrNull(insertStatement, 4, getValueIfNotNull(rowArray, 15));  //
climate
        setValueOrNull(insertStatement, 5, getValueIfNotNull(rowArray, 16));  //
region
    }

    private void setPSInsertWine(PreparedStatement insertStatement, List<String>
row)
            throws SQLException {
        String[] rowArray = (String[]) row.toArray(new String[0]);

        setValueOrNull(insertStatement, 1, getValueIfNotNull(rowArray, 0)); //
wine_name
        setValueOrNull(insertStatement, 2,
                getDoubleFromStringOrNull(getValueIfNotNull(rowArray, 1)));  //
vintage
        setValueOrNull(insertStatement, 3,
                getDoubleFromStringOrNull(getValueIfNotNull(rowArray, 2))); //
alcohol_content
        setValueOrNull(insertStatement, 4, getValueIfNotNull(rowArray, 3));  //
category
        setValueOrNull(insertStatement, 5, getValueIfNotNull(rowArray, 4));  //
color
        setValueOrNull(insertStatement, 6,
                getIntegerFromStringOrNull(getValueIfNotNull(rowArray, 6)));  //
winery_id
        setValueOrNull(insertStatement, 7,
                getIntegerFromStringOrNull(getValueIfNotNull(rowArray, 12)));  //
zone_id
        setValueOrNull(insertStatement, 8,
                getDoubleFromStringOrNull(getValueIfNotNull(rowArray, 5)));  //
price
    }

    private Integer getIntegerFromStringOrNull(String integer) {
        return (integer != null) ? Integer.valueOf(integer) : null;
    }
```

```
    private Double getDoubleFromStringOrNull(String doubl) {
        return (doubl != null) ? Double.valueOf(doubl) : null;
    }

    private String getValueIfNotNull(String[] rowArray, int index) {
        return (index < rowArray.length && rowArray[index].length() > 0) ?
rowArray[index] : null;
    }

    private void setValueOrNull(PreparedStatement preparedStatement, int
parameterIndex,
                                Integer value) throws SQLException {
        if (value == null) {
            preparedStatement.setNull(parameterIndex, Types.INTEGER);
        } else {
            preparedStatement.setInt(parameterIndex, value);
        }
    }

    private void setValueOrNull(PreparedStatement preparedStatement, int
parameterIndex,
                                Double value) throws SQLException {
        if (value == null) {
            preparedStatement.setNull(parameterIndex, Types.DOUBLE);
        } else {
            preparedStatement.setDouble(parameterIndex,
Double.valueOf(value.doubleValue()));
        }
    }

    private void setValueOrNull(PreparedStatement preparedStatement, int
parameterIndex, String value)
            throws SQLException {
        if (value == null || value.length() == 0) {
            preparedStatement.setNull(parameterIndex, Types.VARCHAR);
        } else {
            preparedStatement.setString(parameterIndex, value);
        }
    }

}
```

Exercise2.data

#wine_name,vintage,alcohol_content,category,color,price,winery_id,winery_name,town,established_year,winery_phone,sales_representative,zone_id,zone_name,capital_town,climate, region
Cava Juve & Camps Reserva de la Familia, 2020, 10.5, reserve, white, 82, 40 , Juve & Camps, Penedès, 1898, 587-837-3933, Ramon Garcia , 32, Cava, Penedès, Continental, Catalunya Spain
Viña Tondonia, 2010, 13.5, reserve, red, 38, 2, Marqués de Riscal, La Rioja, 1858, 932-938-3827, Felipe Nuevo , 12, La Rioja, Logroño, Continental, La Rioja Spain
Txomin Etxaniz, 2021, 11, young, white, 15, 41, Txomin Etxaniz, Bilbao, 1912, 444-228-3323, Patxi Lopez , 41, Getariako Txakolina, Bilbao, Oceanic, Euskadi Spain

Result

To see the result we will use this database query

set search_path to ubd_20241;
SELECT * FROM WINERY;
SELECT * FROM ZONE;
SELECT * FROM WINE;

Result without running Exercise2InsertAndUpdateDataFromFile program

	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	stock integer	price numeric (8,2)	prizes integer
34	34	Bodegas Ateca Honoro Vera	2019	14.50	young	red	33	34	83	47.95	[null]
35	35	Borsao Tres Picos	2018	15.00	young	red	34	35	67	71.95	[null]
36	36	San Valero Particular	2019	13.50	young	red	35	36	50	52.75	[null]
37	37	Celler de Capçanes Mas Donís	2018	14.00	young	red	36	37	67	62.35	[null]
38	38	Celler Piñol L'Avi Arrufi	2017	14.50	reserve	red	37	38	42	119.95	[null]
39	39	Castello di Ama Chianti Classico	2018	13.50	young	red	38	39	83	95.95	[null]
40	40	Marchesi di Barolo Barolo	2016	14.00	reserve	red	39	40	33	239.95	1
41	41	Alión	2018	14.50	reserve	red	3	31	39	195.95	[null]
42	42	Torres Pazo das Bruxas	2019	11.00	young	white	1	1	50	24.95	2
43	43	Enate Merlot-Merlot	2016	13.50	reserve	red	31	32	67	43.95	[null]
44	44	López de Heredia Viña Bosconia	2010	14.00	reserve	red	10	12	33	119.50	4
45	45	Pago de Carraovejas	2018	14.50	reserve	red	13	31	70	95.95	6
46	46	El Sequé	2017	14.00	reserve	red	12	31	20	49.50	2
47	47	Hacienda Monasterio	2018	14.00	reserve	red	2	31	25	69.95	1
48	48	Pingus	2015	15.00	grand reserve	red	16	31	10	1200.00	5
49	49	Flor de Pingus	2017	14.50	reserve	red	16	31	20	249.95	3
50	50	Dominio del Águila Reserva	2016	14.50	reserve	red	17	31	15	95.95	2
51	51	Mauro	2019	14.00	reserve	red	18	31	60	79.95	1
52	52	Aalto PS	2018	14.50	reserve	red	19	31	45	150.00	4
53	53	Artadi Pagos Viejos	2017	14.50	grand reserve	red	10	12	12	225.95	3
54	54	Remelluri Reserva	2018	14.00	reserve	red	11	12	55	49.95	2
55	55	Viña Real Crianza	2018	14.00	young	red	12	12	120	15.50	1
56	56	Lan Reserva	2017	14.00	reserve	red	13	12	60	19.95	[null]
57	57	Marqués de Murrieta Reserva	2016	14.00	reserve	red	14	12	30	26.50	4

A total of 57 elements.

Result by running the Exercise2InsertAndUpdateDataFromFile program

	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	stock integer	price numeric (8,2)	prizes integer
37	37	Celler de Capçanes Mas Donís	2018	14.00	young	red	36	37	67	62.35	[null]
38	38	Celler Piñol L'Avi Arrufi	2017	14.50	reserve	red	37	38	42	119.95	[null]
39	39	Castello di Ama Chianti Classico	2018	13.50	young	red	38	39	83	95.95	[null]
40	40	Marchesi di Barolo Barolo	2016	14.00	reserve	red	39	40	33	239.95	1
41	41	Alión	2018	14.50	reserve	red	3	31	39	195.95	[null]
42	42	Torres Pazo das Bruxas	2019	11.00	young	white	1	1	50	24.95	2
43	43	Enate Merlot-Merlot	2016	13.50	reserve	red	31	32	67	43.95	[null]
44	44	López de Heredia Viña Bosconia	2010	14.00	reserve	red	10	12	33	119.50	4
45	45	Pago de Carraovejas	2018	14.50	reserve	red	13	31	70	95.95	6
46	46	El Sequé	2017	14.00	reserve	red	12	31	20	49.50	2
47	47	Hacienda Monasterio	2018	14.00	reserve	red	2	31	25	69.95	1
48	48	Pingus	2015	15.00	grand reserve	red	16	31	10	1200.00	5
49	49	Flor de Pingus	2017	14.50	reserve	red	16	31	20	249.95	3
50	50	Dominio del Águila Reserva	2016	14.50	reserve	red	17	31	15	95.95	2
51	51	Mauro	2019	14.00	reserve	red	18	31	60	79.95	1
52	52	Aalto PS	2018	14.50	reserve	red	19	31	45	150.00	4
53	53	Artadi Pagos Viejos	2017	14.50	grand reserve	red	10	12	12	225.95	3
54	54	Remelluri Reserva	2018	14.00	reserve	red	11	12	55	49.95	2
55	55	Viña Real Crianza	2018	14.00	young	red	12	12	120	15.50	1
56	56	Lan Reserva	2017	14.00	reserve	red	13	12	60	19.95	[null]
57	57	Marqués de Murrieta Reserva	2016	14.00	reserve	red	14	12	30	26.50	4
58	58	Cava Juve & Camps Reserva de la Familia	2020	10.50	reserve	white	40	32	0	82.00	[null]
59	59	**Viña Tondonia**	2010	13.50	reserve	red	2	12	0	38.00	[null]
60	60	Txomin Etxaniz	2021	11.00	young	white	41	41	0	15.00	[null]

A total of 60 elements, 3 have been added.

Terminal result after completion of Exercise2InsertAndUpdateDataFromFile program

```
PS C:\Users\maest\Desktop\UOC\5 Semester\Introduction to Databases\PR\3\20241ENU-1>
c:; cd 'c:\Users\maest\Desktop\UOC\5 Semester\Introduction to Databases\PR\3\20241EN
U-1'; & 'C:\Program Files\Java\jdk-17\bin\java.exe' '@C:\Users\maest\AppData\Local\T
emp\cp_cwt2zreuj9wxx1atv8coil96m.argfile' 'edu.uoc.practica.bd.uocdb.exercise2.Exerc
ise2InsertAndUpdateDataFromFile'
##########Successful connection to the database.##########
Winery updated: Juve & Camps
Zone already exists: Cava
New wine inserted: Cava Juve & Camps Reserva de la Familia
Winery updated: MarquÃ©s de Riscal
Zone already exists: La Rioja
New wine inserted: ViÃ±a Tondonia
Winery updated: Txomin Etxaniz
Zone already exists: Getariako Txakolina
New wine inserted: Txomin Etxaniz
##########Transaction committed successfully.##########
```