Gazi University Faculty of Engineering

Project final report
Gestion of a sport event

Work made by:
Diego de Luis Ballesteros, ST NO: 23118080904, diegodeluis@gazi.edu.tr
Course:
Database applications
Academic year:
2023/2024

Index

Introduction

Motivations

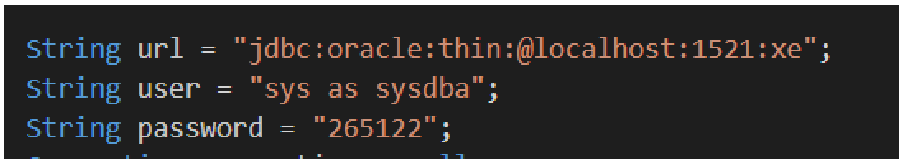
To carry out this project, I have relied on an entity-relationship diagram (Figure 1.1) for sports events. I obtained this diagram from the internet at my university in Spain. I have implemented a series of tables in accordance with the diagram. Not all of them are present because I have attempted to simplify the implementation to avoid redundancy. For example, I removed the "sponsors" table because it performs the same function as the "collaborates" table, and I merged the "material" and "have" tables into one, etc.

My sports event management system features a versatile interface that allows users to modify multiple elements of the database through various interfaces, offering flexibility in event management. The use of an Oracle SQL database connection emphasizes the system's commitment to robust and widely-used technology, ensuring increased security, scalability, and performance. Below, I will name different sports management systems and compare them to my system.

On one hand, we have SportsEngine, which provides a comprehensive platform covering team management, event organization, registrations, and communications, making it an attractive option for those seeking an integrated solution. On the other hand, TeamSnap specializes in team and sports event management, offering specific features for scheduling, task assignment, team communication, and stat tracking, catering to a more team-focused approach.

When considering versatility versus specialization, my system stands out by providing adaptable solutions, while SportsEngine and TeamSnap excel in comprehensive and team-focused management, respectively. The emphasis on Oracle SQL database technology highlights the solid and secure infrastructure supporting my system.

About the project (functionalities, requirements, etc.)



```
String url = "jdbc:oracle:thin:@localhost:1521:xe";
String user = "sys as sysdba";
String password = "265122";
```

Figure 1.0

To be able to use the project on the computer, it is necessary to have the Oracle database installed or any other Oracle SQL database that can be accessed from Java. In Java, it is sufficient to add the components explained below. I will use my connection as an example in the explanation (figure 1.0). SEE REFERENCE 1 AND REFERENCE 2

Protocol: jdbc indicates that the JDBC protocol will be used to connect to the database.

Driver: oracle:thin indicates that the thin driver from Oracle will be used.

Host: localhost indicates that the database is on the same computer as the application making the connection. If the database is on another computer, you need to specify the IP address or domain name of that computer.

Port: 1521 is the default Oracle Net port for the database listener. If the listener is configured to use a different port, you must specify that port in this part of the connection.

SID: xe is the SID of the database. The SID is a unique identifier for an instance of the Oracle Database.

JDBC is a library in java, it is already in the lib folder. You need to add it to the project to use this connection. **User and password** will be the user and password of your oracle connection.

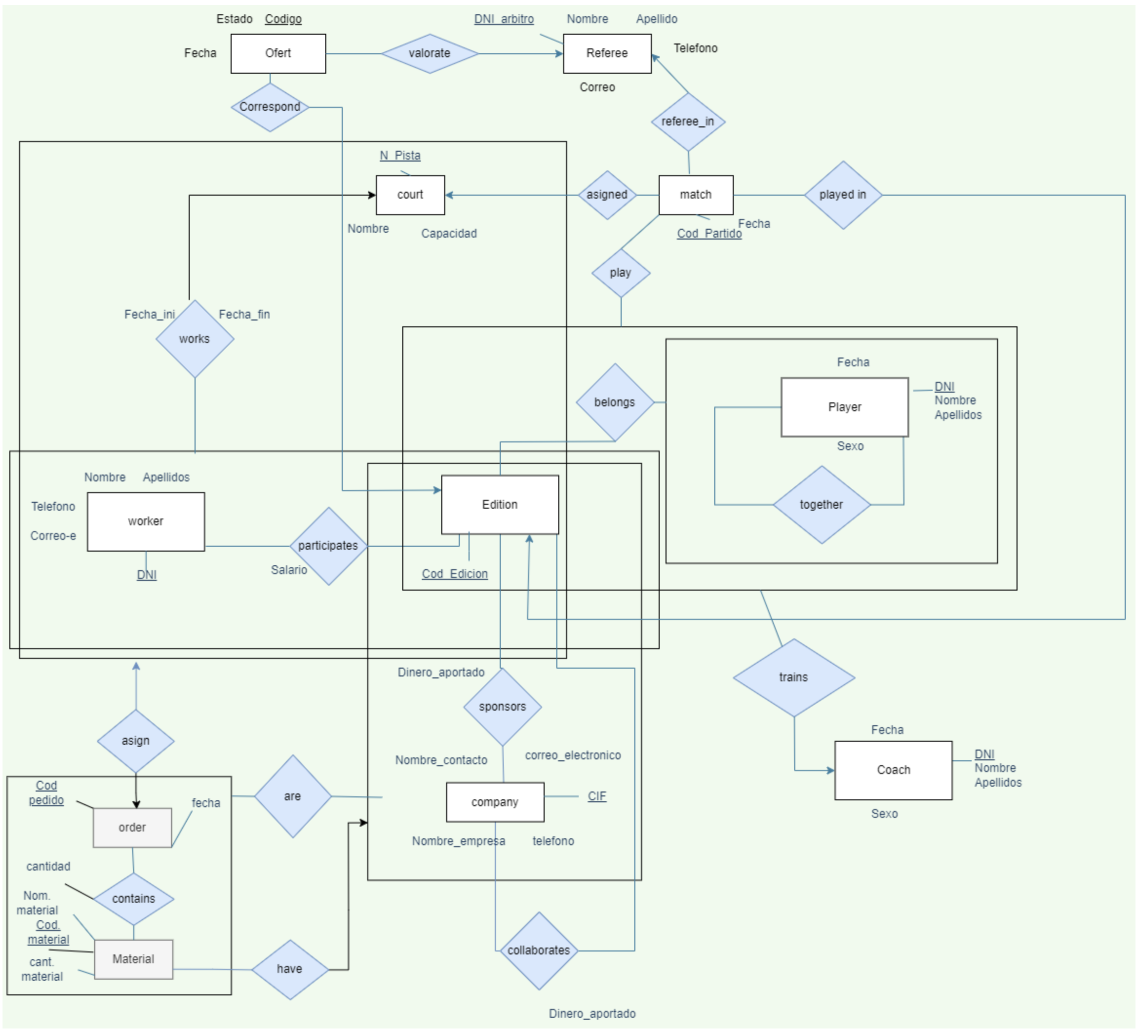


Figure 1.1

My database application is divided into two parts. The first part is system.sql, where all the tables I will use are defined, and they need to be added to the database beforehand. Additionally, there will be a series of data that also needs to be added to the database for the ease of using the application. This way, we simulate as if our sports event is already created, and we only want to modify certain parameters with the application.

The second part is the Java application. In this part, there will be a src folder where all the Java application code is located. The code follows the model-view-controller (MVC) pattern, meaning the view and the different user input interfaces communicate with the controller on one side, and the database communicates with it on the other side. At the same time, the controller communicates with the main part of the application. All SQL statements, including the connection and commit, will be defined in the database. The controller will call methods from the database and methods to make different user input interfaces visible or invisible for data entry (Figure 1.2).

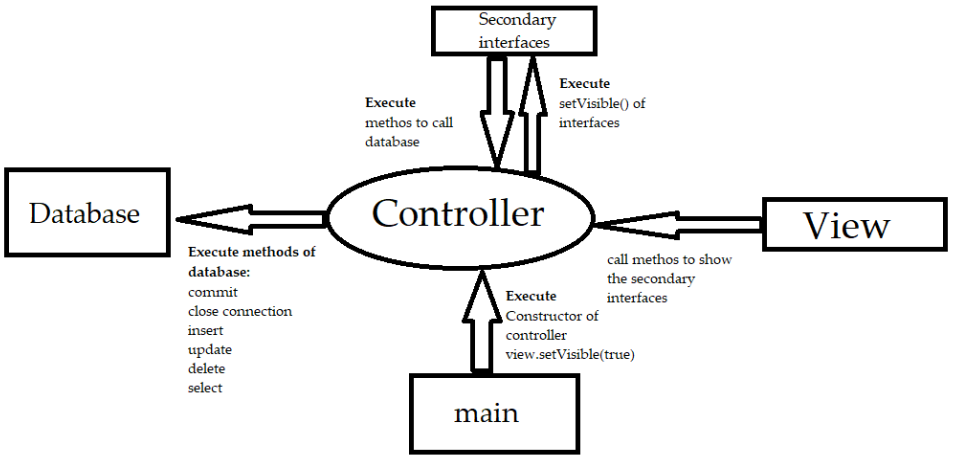

Secondary interfaces
Execute methos to call database
Execute setVisible() of interfaces
Database
Controller
View
Execute methods of database:
commit
close connection
insert
update
delete
select
call methos to show the secondary interfaces
Execute Constructor of controller view.setVisible(true)
main

Figure 1.2

For my project, I specifically used the local Oracle SQL database. I had to download it from the main Oracle website. I also used SQL Developer, which can be found on the internet. On the other hand, I used the Eclipse IDE to develop my Java program with the WindowsBuilder plugin, allowing me to edit interfaces more comfortably. SEE REFERENCE 3

Modules explanation

I will now explain the different modules in both my SQL file and my Java program.

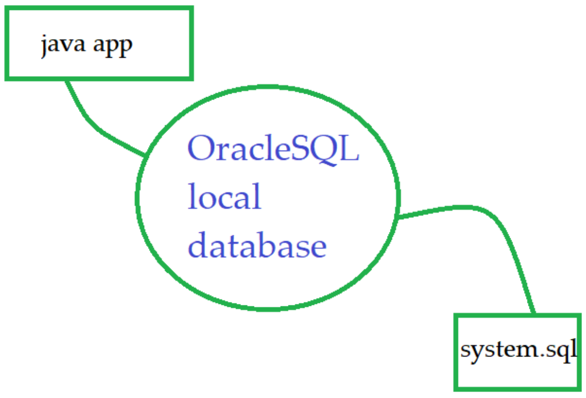


Figure 2.0

SQL file

The tables in the database include: PLAYS, BELONGS, TOGETHER, COACH, PLAYER, WORKS, PARTICIPATES, EMPLOYEE, MATERIAL, MATCH_, OFFER, COURT, REFEREE, COLLABORATES, COMPANY, and EDITION_. These tables collectively form the structure for organizing and managing various aspects related to sports events, including player interactions, team affiliations, coaching relationships, workforce details, participation records, material handling, match and court information, referee assignments, collaborations, company details, and event editions. Each table serves a specific purpose in the relational database model, contributing to the effective and organized storage of data for sports event management. I deleted didn't add some of the tables of E/R because I merged some of them.

I will now list the tables that have been joined. You can see the tables in figure 1.0

Match: The "match" table connects "referee_in" with matches through the foreign keys "DNI_referee" and "Cod_Match". This table also links "courts" using the keys "Cod_Match" and "N_court". Finally, the table joins "played in" too.

Plays: The "Plays" tables joins the "train" table adding "DNI_1", "DNI_2" and "Cod_edition"

Material: The "Material" table establishes the relationship between materials, companies, and editions through the keys "Cod_material", "CIF", and "Cod_Edition" joining the table have.

Offer: The "Offer" table joins "valorate" table adding "DNI_referee" correspond table adding "edition_code".

In this case we will not use offer table because for our example on the database all referees will be in the edition with code "0001". Moreover, I deleted the following tables related to the material: order, are, assigns and contains. So, the material will be added directly without an order from the edition. I simulate that the admin who manage the system will add it directly. I did this to simplify the system since the crucial part of the system is more focused on the team, the coach, the match, etc.

Additionally, as an explanation, I must add that 'DNI' is the ID of the individual, and 'CIF' is the ID of the company.

Finally, I will add some values. You can see them in the end of the sql file.

SEE REFERENCE 4 For SQL usage

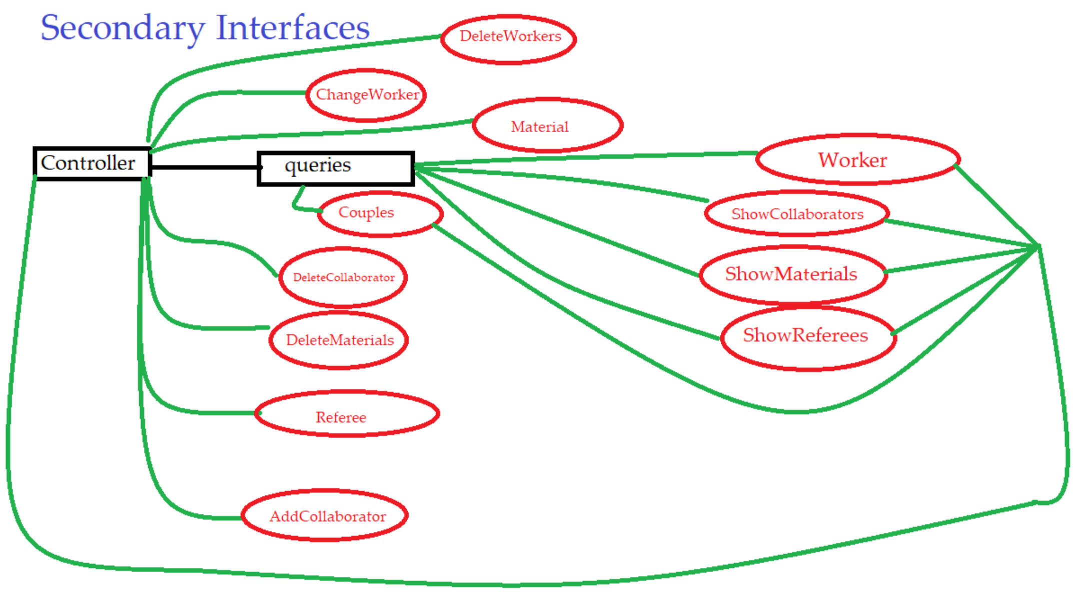
Java application

I will now explain each of the following modules one by one, showing screenshots of the application, etc.

Modules

I'm going to divide de module's explanation according to Figure 1.2.

Secondary Interfaces



Firgure 2.1

For the secondary interfaces, we have two parts: the first part is for displaying data from the database, which will be in the 'queries' section, and the second part is for modifying the database, which will be the rest of the classes.

The 'queries' class will be an auxiliary class to display the four different tables. It's simply a 'JFrame' to which I add access to these four interfaces.

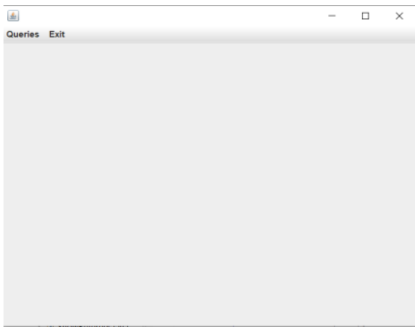
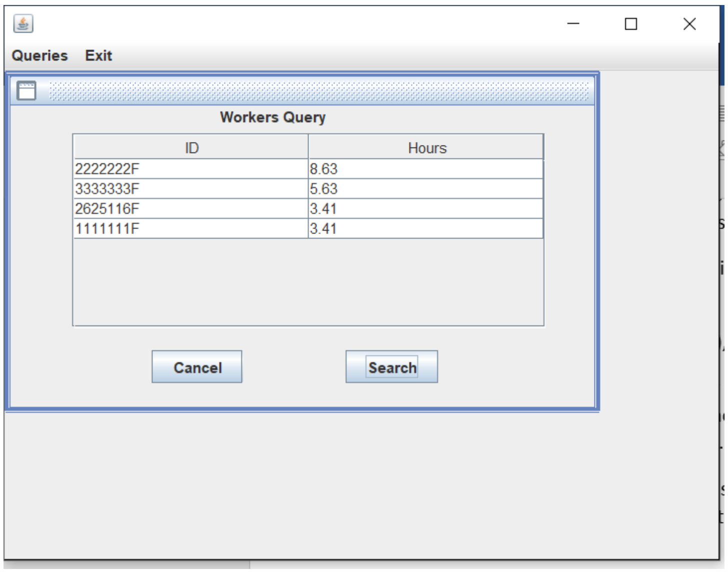


Figure 2.2

In the data display part (queries), I have 5 classes.

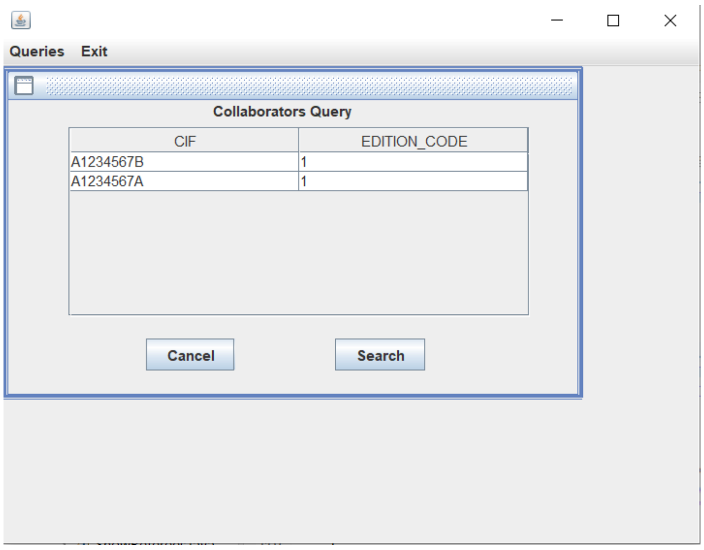
The 'worker.java' class will show us all the workers and how many total hours each one works. They are ordered according to the hors worked. 'Worker.java' calls the 'showWorkers' method of the 'controller.java' class, which in turn calls 'searchWorkerTable' from the 'database.java' class (Figure 2.3).



ID	Hours
2222222F	8.63
3333333F	5.63
2625116F	3.41
1111111F	3.41

Figure 2.3

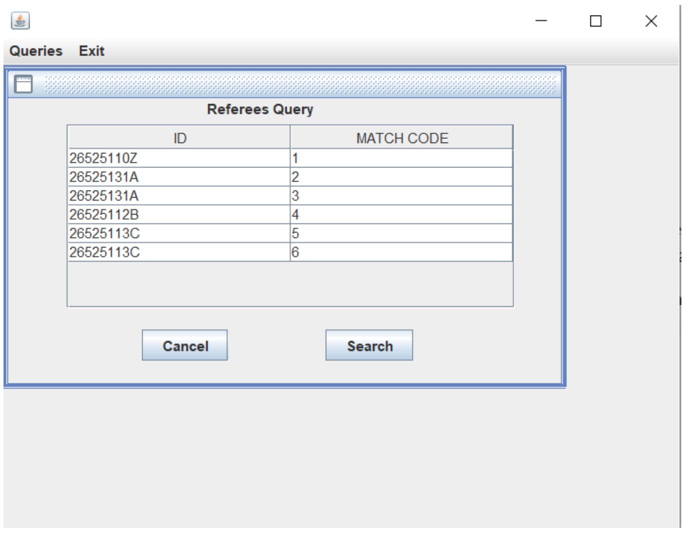
The 'ShowCollaborators.java' class will display all the companies collaborating in any edition and will tell us in which edition each company collaborates. This class calls the corresponding method in the controller, and the controller in 'Database.java' (Figure 2.4).



CIF	EDITION_CODE
A1234567B	1
A1234567A	1

Figure 2.4

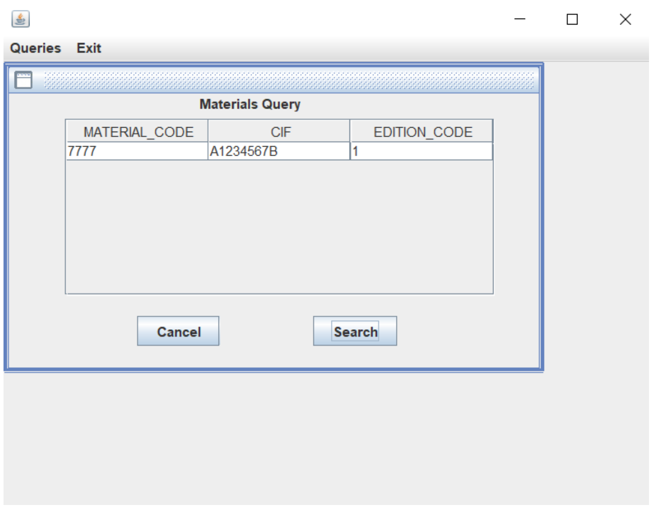
The 'ShowReferees.java' class will display all the referees arbitrating in a match with a specified match code. This class calls the corresponding method in the controller, and the controller in 'Database.java' (Figure 2.5).



ID	MATCH CODE
26525110Z	1
26525131A	2
26525131A	3
26525112B	4
26525113C	5
26525113C	6

Figure 2.5

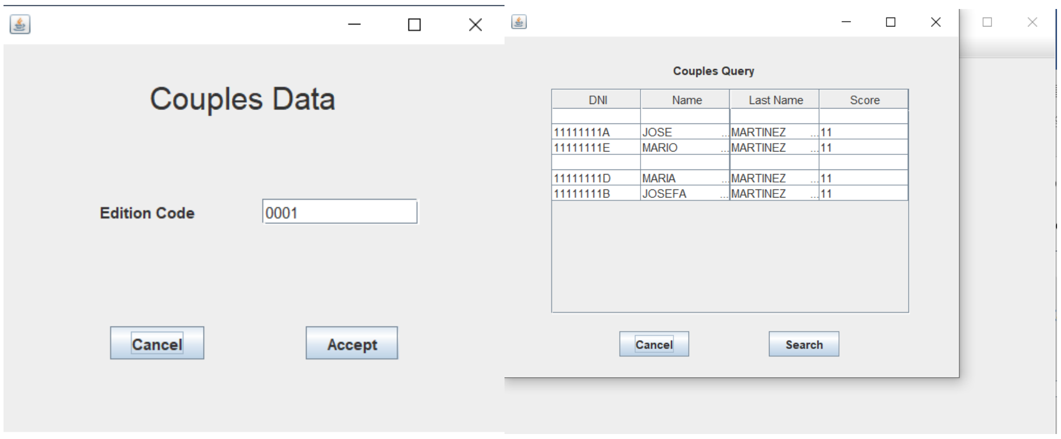
The 'ShowMaterials.java' class will display all the materials, along with the code (CIF) of the company that provided it and the edition code. This class calls the corresponding method in the controller, and the controller in 'Database.java' (Figure 2.6)."



MATERIAL_CODE	CIF	EDITION_CODE
7777	A1234567B	1

Figure 2.6

"Couples.java" displays the couples of a specific edition. First, a frame is opened to enter the edition, and then you can query the couples. They will be displayed with a score in descending order. This class calls the corresponding method in the controller, and the controller interacts with "DataBase.java" (Figure 2.7).


Couples Data

Edition Code	0001

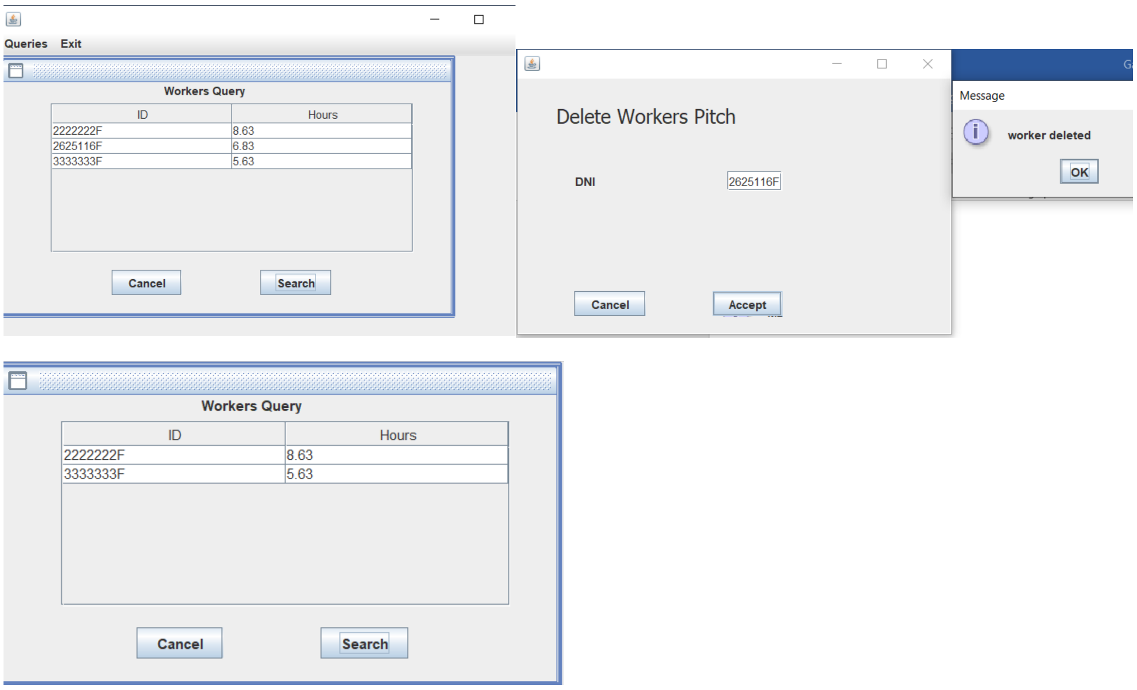
Couples Query

DNI	Name	Last Name	Score
11111111A	JOSE	MARTINEZ	11
11111111E	MARIO	MARTINEZ	11
11111111D	MARIA	MARTINEZ	11
11111111B	JOSEFA	MARTINEZ	11

Figure 2.7

On the other hand, I have the remaining classes that are used to add, update, or delete items from the database.

"DeleteWorkers.java" is responsible for removing workers from any sports event. I have considered that if a worker is dismissed from an event, they cannot work in more events. Therefore, you only need to insert the worker's ID (DNI). This class calls the corresponding method in the controller, and the controller interacts with "DataBase.java" (Figure 2.8).



Workers Query

ID	Hours
2222222F	8.63
2625116F	6.83
3333333F	5.63

Delete Workers Pitch

DNI	2625116F

Message

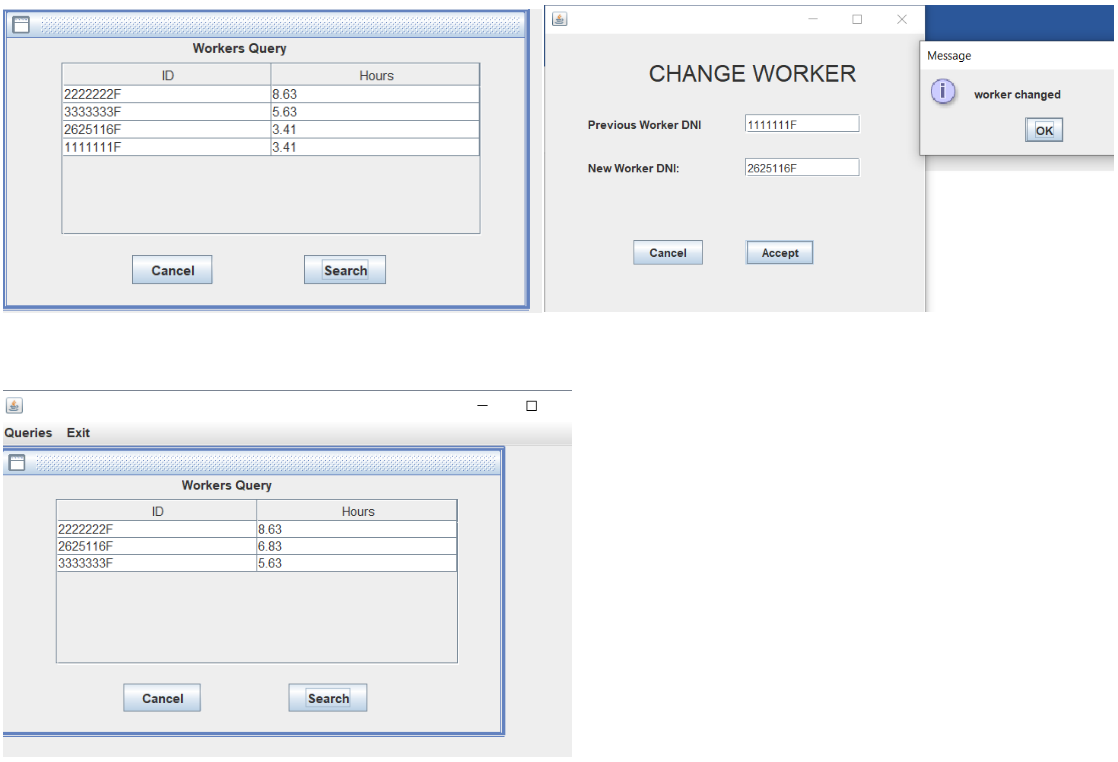
worker deleted

Workers Query

ID	Hours
2222222F	8.63
3333333F	5.63

Figure 2.8

"ChangeWorker.java" is designed to replace one worker with another. In other words, if a worker is not performing well, we can replace them with another existing worker already in the database. This class calls the corresponding method in the controller, and the controller interacts with "DataBase.java" (Figure 2.9).


Workers Query

ID	Hours
2222222F	8.63
3333333F	5.63
2625116F	3.41
1111111F	3.41

CHANGE WORKER

Previous Worker DNI: 1111111F

New Worker DNI: 2625116F

Message

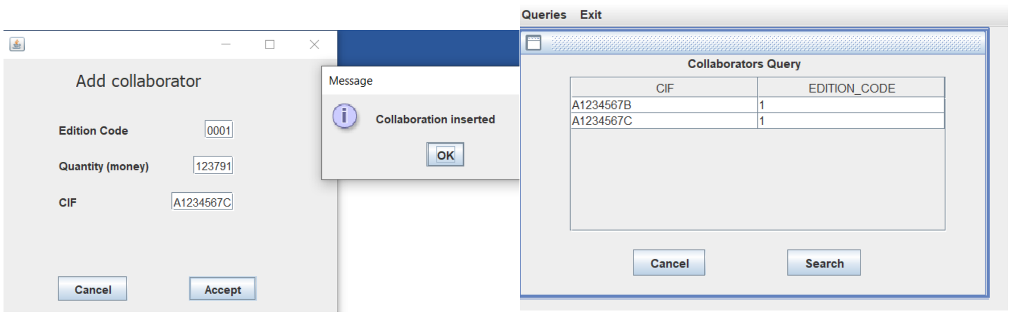
worker changed

Workers Query

ID	Hours
2222222F	8.63
2625116F	6.83
3333333F	5.63

Figure 2.9

"AddCollaborator.java" adds a new collaborating company to an edition. This company will contribute a specific amount of money represented by the 'quantity' variable. This class calls the corresponding method in the controller, and the controller interacts with "DataBase.java" (Figure 2.10).


Add collaborator

Edition Code: 0001

Quantity (money): 123791

CIF: A1234567C

Message

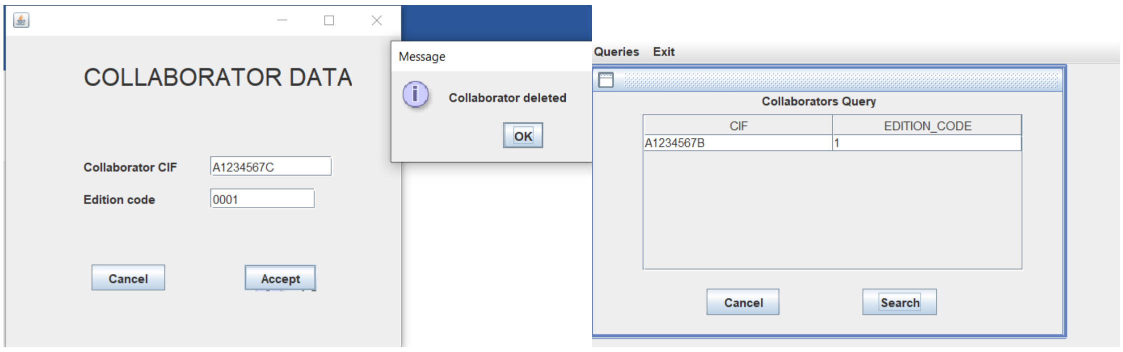
Collaboration inserted

Collaborators Query

CIF	EDITION_CODE
A1234567B	1
A1234567C	1

Figure 2.10

"DeleteCollaborator.java" removes a collaboration of a company (with its corresponding CIF) in a specific edition. This class calls the corresponding method in the controller, and the controller interacts with "DataBase.java" (Figure 2.11).


COLLABORATOR DATA

Collaborator CIF: A1234567C

Edition code: 0001

Message

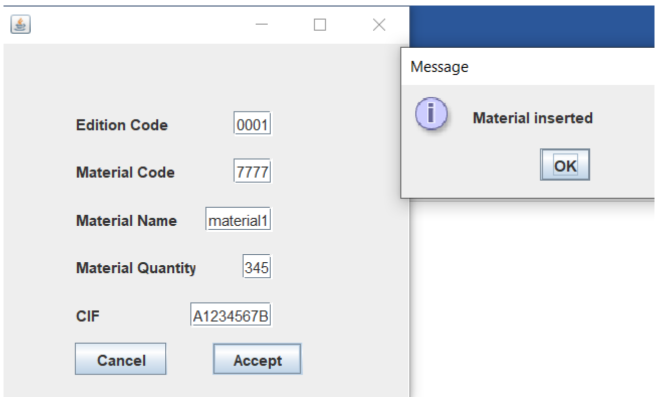
Collaborator deleted

Collaborators Query

CIF	EDITION_CODE
A1234567B	1

Figure 2.11

"Material.java" adds a new material to an edition. This material should contain the data for the material code, material name, quantity, CIF of the providing company, and the edition. This class calls the corresponding method in the controller, and the controller interacts with "DataBase.java" (Figure 2.12).

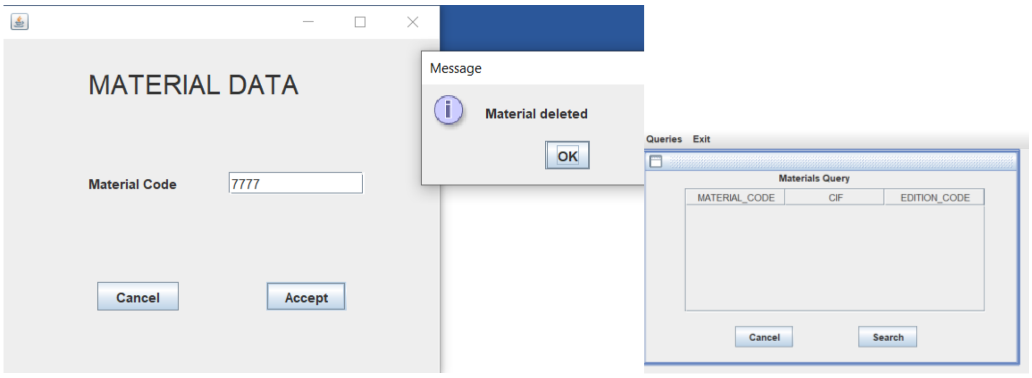

Edition Code	0001
Material Code	7777
Material Name	material1
Material Quantity	345
CIF	A1234567B

Message

Material inserted

Figure 2.12

"DeleteMaterial.java" removes a material based on its code. This class calls the corresponding method in the controller, and the controller interacts with "DataBase.java" (Figure 2.13).


MATERIAL DATA

Material Code: 7777

Message

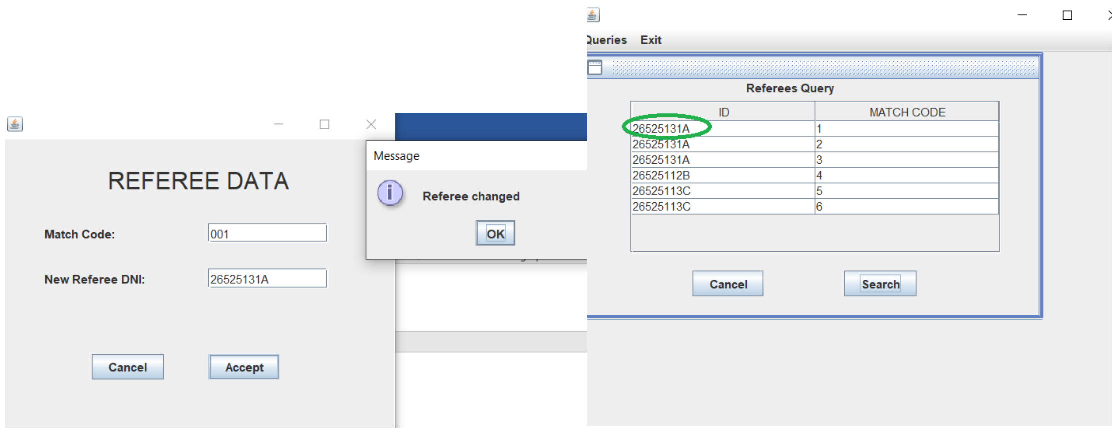
Material deleted

Materials Query

MATERIAL_CODE	CIF	EDITION_CODE

Figure 2.13

"Referee.java" changes the referee arbitrating in a specific match. You need to enter the match code and the referee's ID (DNI). This class calls the corresponding method in the controller, and the controller interacts with "DataBase.java" (Figure 2.14).



Match Code: 001

New Referee DNI: 26525131A

Referee changed

ID	MATCH CODE
26525131A	1
26525131A	2
26525131A	3
26525112B	4
26525113C	5
26525113C	6

Figure 2.14

View

The view is the main design of the application that opens in the controller's constructor. The view contains all the buttons that lead us to the secondary interfaces. It is connected to the controller class, and for each click on each button, it calls a method of the controller that opens a different secondary view. Figure 2.15

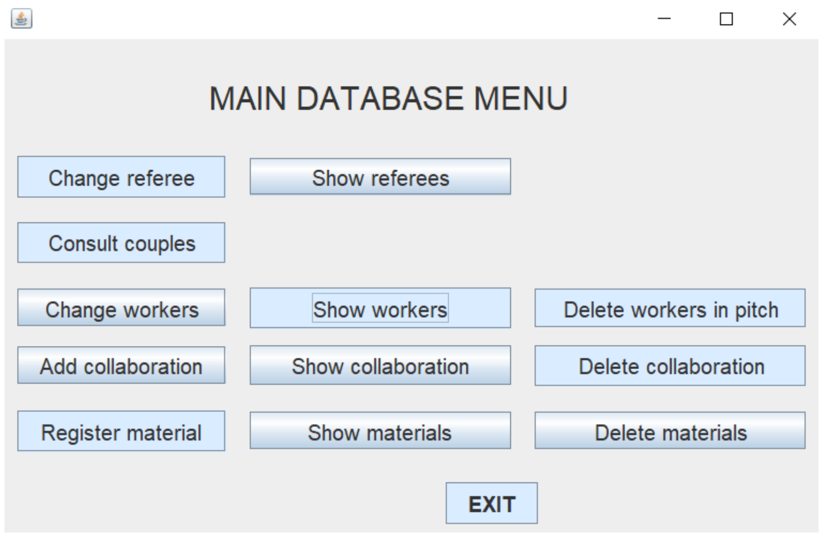
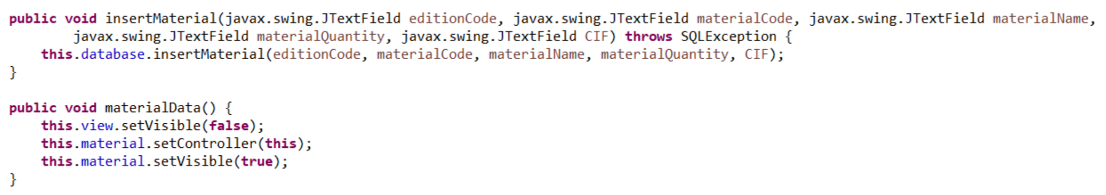


Figure 2.15

Controller

The controller acts as the central nervous system of the Java application, bridging database interactions and interface visualizations. It seamlessly handles calls to both database methods and display methods, enabling secondary interfaces to communicate with the database through it. Figure 2.16 showcases this interplay, where the 'material.java' interface interacts with 'insertMaterial()' in 'database.java' for database operations, while the 'materialData()' method leverages 'setVisible' to orchestrate interface visibility, all orchestrated by the controller.



```java
public void insertMaterial(javax.swing.JTextField editionCode, javax.swing.JTextField materialCode, javax.swing.JTextField materialName,
        javax.swing.JTextField materialQuantity, javax.swing.JTextField CIF) throws SQLException {
    this.database.insertMaterial(editionCode, materialCode, materialName, materialQuantity, CIF);
}

public void materialData() {
    this.view.setVisible(false);
    this.material.setController(this);
    this.material.setVisible(true);
}
```

Figure 2.16

Main

Object Instantiation:

The method begins by creating instances of various classes, each representing specific functionalities within the application. These include classes for handling referees, collaborators, materials, queries, worker management, couples lists, collaborations, and more.

It also creates instances of core classes like DataBase, View, and Controler (likely a typo for Controller), which form the foundation of the application's structure.

Database Connection:

The method attempts to establish a connection to the database using the establishConnection method of the DataBase class. This step is crucial for enabling the application to interact with and retrieve data from the database.

If an error occurs during the connection attempt, an error message is printed to alert the user or developer of the issue.

User Interface Setup:

The method creates an instance of the View class, which is responsible for setting up the user interface (UI) of the application. This includes elements like windows, menus, buttons, and other visual components that allow users to interact with the application.

Controller Initialization:

The method creates an instance of the Controler class, following the Model-View-Controller (MVC) design pattern. This pattern separates the application's concerns into three distinct parts:

Model: Represents the data and business logic of the application.

View: Handles the user interface and its interactions.

Controller: Mediates between the model and view, managing application behavior and user actions.

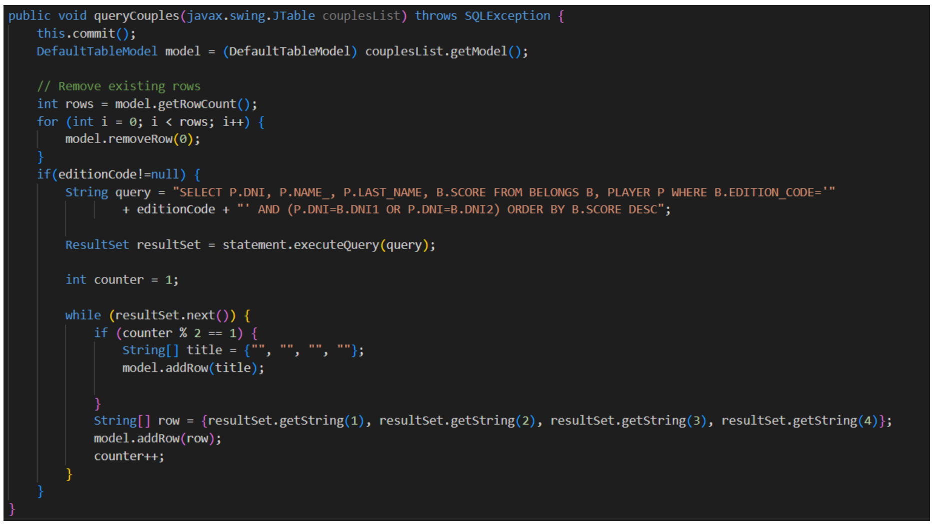
Database

This Java code establishes a connection with an Oracle database and provides methods to perform database operations related to sports events, such as changing referees, displaying pairs in an edition, and managing materials and collaborators. As an example, I'm going to explain "queryCouples" method (figure 2.17). SEE REFERENCE 5 for transactions like commit, SEE REFERENCE 6 and SEE REFERENCE 7 for other commands in java sql.

This method is responsible for querying the database to retrieve and display information about pairs in a specific edition in a graphical interface table (**couplesList**). Here's a detailed explanation of the method:

Detailed Explanation:

- **commit()** is invoked to commit any pending changes to the database.
- The table model (**DefaultTableModel**) is obtained to manipulate its data.
- Existing rows in the table are removed.
- It checks if **editionCode** is not null.
- An SQL query is constructed to retrieve information about pairs in the specified edition. Ordering them by the score.
- The query is executed, and a result set (**resultSet**) is obtained.
- Results are iterated, and rows are added to the table (**couplesList**).
- If **counter** is odd, an empty title row is added for better readability.
- **counter** is incremented for handling alternate rows.



```java
public void queryCouples(javax.swing.JTable couplesList) throws SQLException {
    this.commit();
    DefaultTableModel model = (DefaultTableModel) couplesList.getModel();

    // Remove existing rows
    int rows = model.getRowCount();
    for (int i = 0; i < rows; i++) {
        model.removeRow(0);
    }
    if(editionCode!=null) {
        String query = "SELECT P.DNI, P.NAME_, P.LAST_NAME, B.SCORE FROM BELONGS B, PLAYER P WHERE B.EDITION_CODE='"
                + editionCode + "' AND (P.DNI=B.DNI1 OR P.DNI=B.DNI2) ORDER BY B.SCORE DESC";

        ResultSet resultSet = statement.executeQuery(query);

        int counter = 1;

        while (resultSet.next()) {
            if (counter % 2 == 1) {
                String[] title = {"", "", "", ""};
                model.addRow(title);

            }
            String[] row = {resultSet.getString(1), resultSet.getString(2), resultSet.getString(3), resultSet.getString(4)};
            model.addRow(row);
            counter++;
        }
    }
}
```

Figure 2.17

Conclusions and Future Work

In conclusion, the implemented Java application successfully connects to an Oracle database and manages various aspects related to sports events, including referees, collaborators, materials, and worker assignments. It provides a graphical interface for users to interact with the database, displaying relevant information and allowing the execution of operations such as adding, updating, or deleting records.

The system consists of multiple modules, each handling specific functionalities, such as displaying information about workers, collaborators, referees, and materials. The graphical interface facilitates user interaction, making it intuitive and user-friendly.

Future Work and Enhancements:

While the current system fulfills its primary objectives, there are opportunities for further enhancements and improvements.

Security Measures:

Implementing secure authentication and authorization mechanisms to ensure that only authorized users can access and modify the database. Additionally, it would be beneficial to incorporate encryption for sensitive data stored in the database.

Error Handling and Logging:

Enhancing error handling mechanisms to provide more informative error messages to users. Furthermore, implementing a logging system to record events, errors, and user activities for troubleshooting and auditing purposes.

User Experience and Interface:

Improving the overall user interface design for better aesthetics and usability. Additionally, adding user prompts, tooltips, or guidance to assist users in understanding how to use the application effectively.

Database Optimization:

Implementing database optimizations, such as indexing, to enhance query performance, especially for large datasets. Considering database partitioning or sharding strategies to manage scalability.

Extended Functionality:

Adding additional modules or functionalities based on user feedback or evolving requirements. Incorporating features like advanced search capabilities, data visualization, or reporting tools. For example, the possibility of add new workers.

Compatibility and Portability:

Ensuring compatibility with different database systems, not limited to Oracle. Enhancing portability to make the application deployable on various operating systems.

Internationalization and Localization:

Supporting multiple languages to make the application accessible to a broader audience. Allowing users to customize date formats, number formats, and other regional settings.

Performance Monitoring:

Implementing performance monitoring tools to track and analyze application performance over time. Introducing caching mechanisms for frequently accessed data to reduce response times.

By addressing these future work areas, the system can be further refined to meet higher standards of security, usability, and efficiency, ensuring a robust and versatile solution for managing sports event-related data.

References

1. https://www.youtube.com/watch?v=sMjrnJUIgwk&t=69s
2. https://www.oracle.com/database/technologies/oracle21c-windows-downloads.html
3. https://help.eclipse.org/latest/index.jsp?topic=%2Forg.eclipse.wb.doc.user%2Fhtml%2Findex.html
4. https://docs.oracle.com/cd/B19188_01/doc/B15917/squse.htm
5. https://docs.oracle.com/javase/tutorial/jdbc/basics/transactions.html
6. https://stackoverflow.com/questions/34136173/cannot-find-symbol-getmodel
7. https://learn.microsoft.com/en-us/sql/connect/jdbc/reference/executeupdate-method-java-lang-string-sqlserverstatement?view=sql-server-ver16