HACETTEPE UNIVERSITY
COMPUTER ENGINEERING DEPARTMENT
BBM473 - DATABASE MANAGEMENT SYSTEMS LABORATORY
PROJECT PHASE -3

Subject: Developing Desktop Database Management Module

Group Members: Oktay UĞURLU - 21627725, Tuna Aybar TAŞ - 21627648, Koray KARA – 21803682

Due Date: 09.01.2020 - 23:59

GYM MANAGEMENT SYSTEM

1 Project Definition

In this assignment, we are expected to create database module for desktop to be able to store system tables in the database and obtain statistical data from database by using Data Manipulation Language (DML) phrases like "insert, delete, update" operations with data normalization.

2 Login Screen

Before doing any operation, (insertion, update, delete) the admin should enter his/her username and password. In our project, the admin has fully control over the system and he/she can access all the data stored in and easily manipulate the entire system. That's why we developed this desktop application as an admin panel.

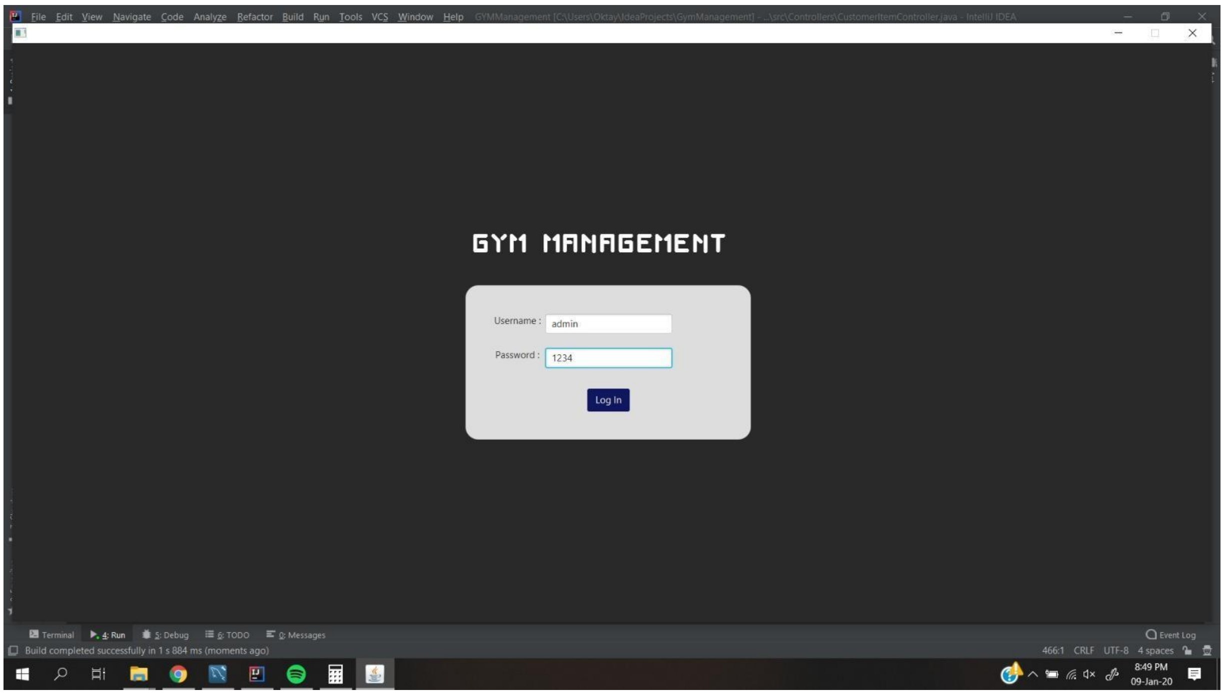


Figure 1: Login Screen

3 Work Done

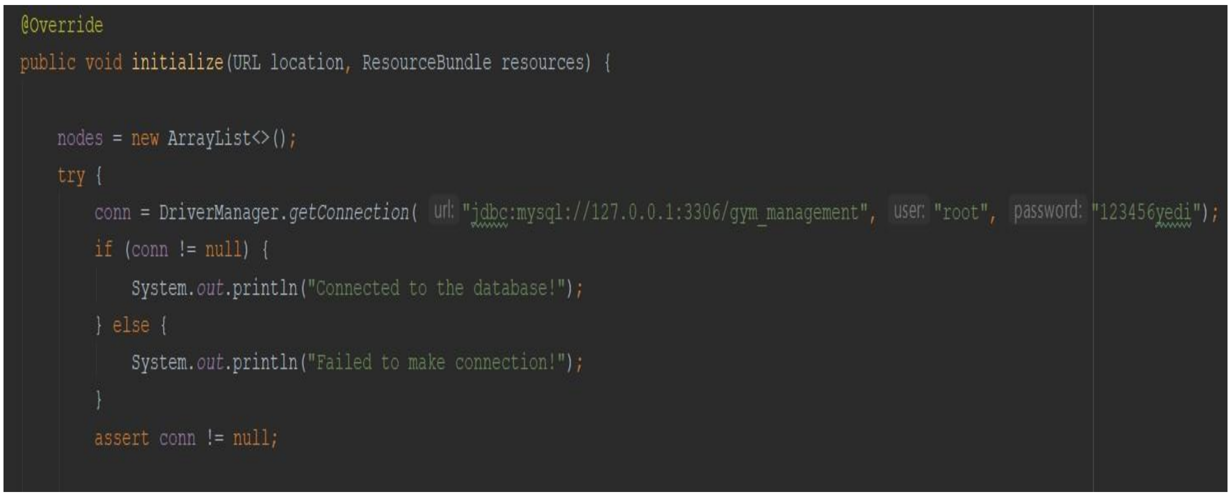
Before starting this assignment, we decided to use JavaFX to create a desktop application in Java8 Runtime Environment. JavaFX is a software platform for creating and delivering rich Internet applications that can run on a wide range of devices as well as desktop applications. It enabled us to design a rich application that operates consistently.

We took advantage of some of the key features available in JavaFX such as FXML and Scene Builder. FXML allowed us to write the user interface separate from the application logic, thereby making the code easier to maintain. Scene Builder generates FXML markup that can be ported to an IDE and it helped us to interactively design the graphical user interface.

The execution of the program begins with the Main class, which call the FXML loader and it parses the FXML document and builds the scene graph. After building the scene graph, the FXML loader instantiates the controller class and finally calls the controller's initialize() method.

In the beginning, in order to connect to MySQL in Java, we had different options, but we preferred to connect to MySQL using Java JDBC because it ensures us some abilities to request connections to the database. Some of them are sending queries to database using SQL statements, and receive results for processing.

We used the DriverManager class by creating Connection interface object. As part of its initialization, this class will attempt to load the driver class. This allowed us to customize the JDBC Drivers used by our application. At the beginning of the initialize() function, we made connection with the help of one of the DriverManager methods called getConnection and counted the tables in our database.

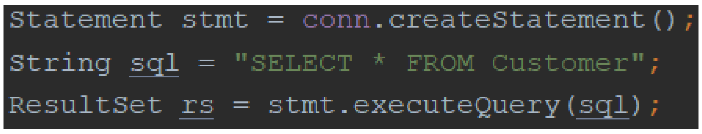


```
@Override
public void initialize(URL location, ResourceBundle resources) {


    nodes = new ArrayList<>();
    try {
        conn = DriverManager.getConnection( url: "jdbc:mysql://127.0.0.1:3306/gym_management", user: "root", password: "123456yedi");
        if (conn != null) {
            System.out.println("Connected to the database!");
        } else {
            System.out.println("Failed to make connection!");
        }
        assert conn != null;
```

Figure 2: initialize function

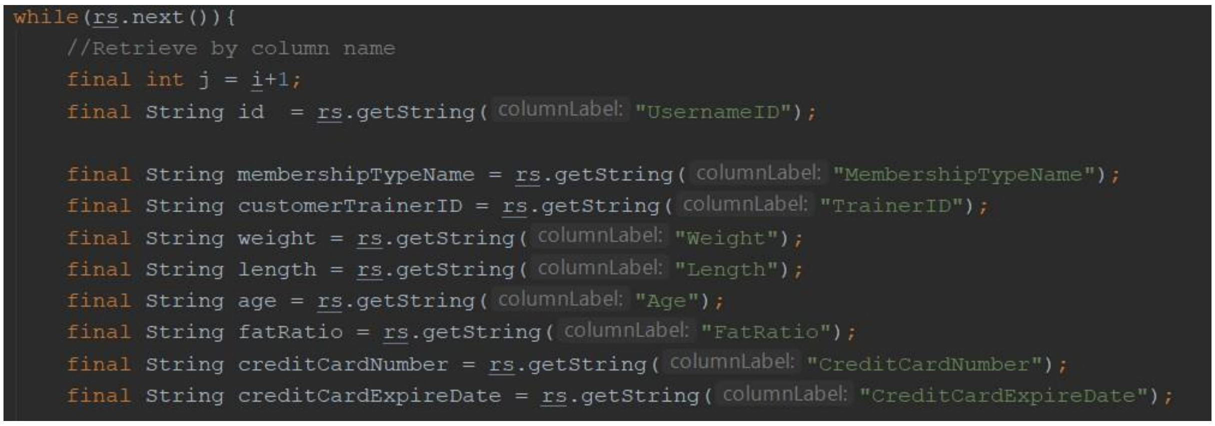
Then, in order to interact with the SQL statements, we used Statement interface. This interface represents a SQL statement and we need a Connection object to create a Statement object. Statement object generate ResultSet objects, which is a table of data representing a database result set.



```
Statement stmt = conn.createStatement();
String sql = "SELECT * FROM Customer";
ResultSet rs = stmt.executeQuery(sql);
```

Figure 3: Creating Statement object

We access the data in a ResultSet object through a cursor. This cursor points to one row of data in the ResultSet object. In order to move the cursor, we repeatedly called the method ResultSet.next and the method outputs the data in the row where the cursor is positioned currently.

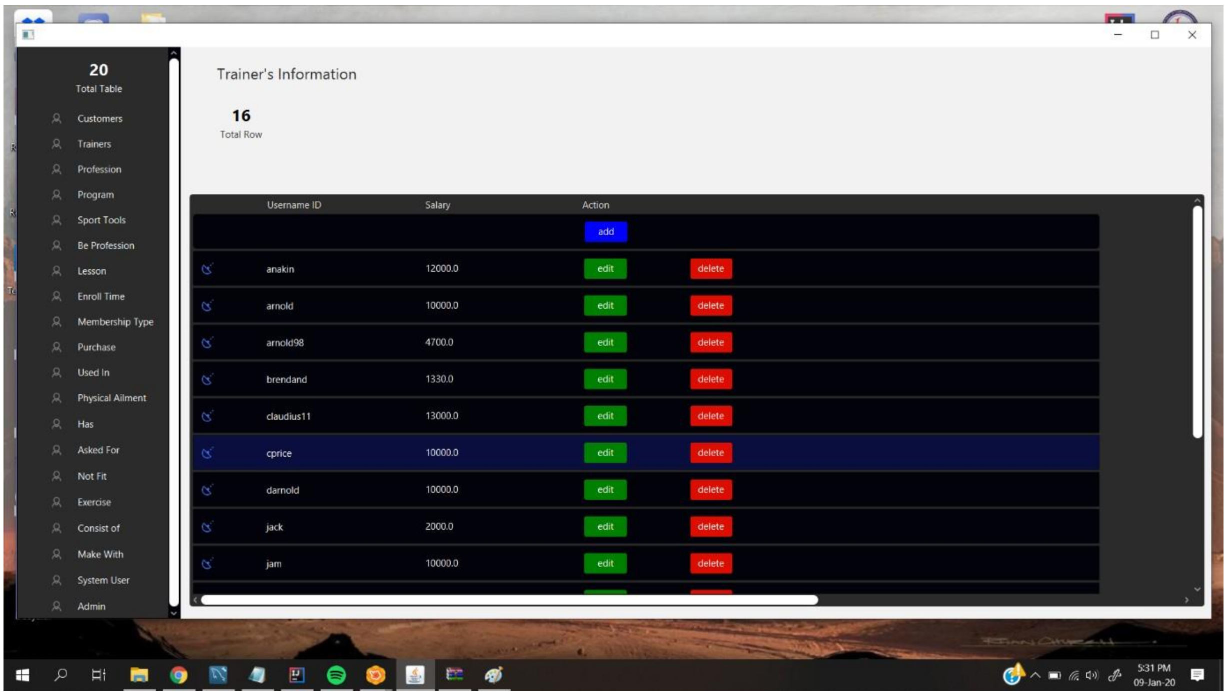


```
while(rs.next()){
    //Retrieve by column name
    final int j = i+1;
    final String id  = rs.getString( columnLabel: "UsernameID");

    final String membershipTypeName = rs.getString( columnLabel: "MembershipTypeName");
    final String customerTrainerID = rs.getString( columnLabel: "TrainerID");
    final String weight = rs.getString( columnLabel: "Weight");
    final String length = rs.getString( columnLabel: "Length");
    final String age = rs.getString( columnLabel: "Age");
    final String fatRatio = rs.getString( columnLabel: "FatRatio");
    final String creditCardNumber = rs.getString( columnLabel: "CreditCardNumber");
    final String creditCardExpireDate = rs.getString( columnLabel: "CreditCardExpireDate");
```

Figure 4: Processing ResultSet Objects

We created controller files for all tables that we created in MySQL and we store some information that are related to FXML files such as Label, TextField, Button, Hbox and Pane objects. We also use handleClicks function that is triggered when an action event happens. We process delete, update and insert operations in this function. An example image of the user interface for trainer table is shown below.



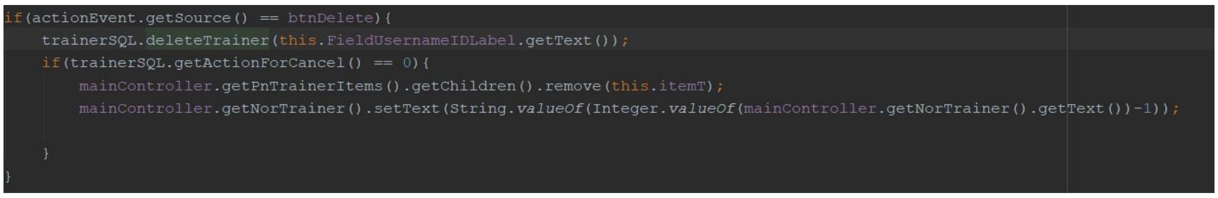
Trainer's Information

16
Total Row

Username ID	Salary	Action	
		add	
anakin	12000.0	edit	delete
arnold	10000.0	edit	delete
arnold98	4700.0	edit	delete
brendand	1330.0	edit	delete
claudius11	13000.0	edit	delete
cprice	10000.0	edit	delete
darnold	10000.0	edit	delete
jack	2000.0	edit	delete
jam	10000.0	edit	delete

Figure 5: User interface for trainer table

When the delete button is pressed, the method of getSource() of ActionEvent object will be equal to btnDelete button and the related row will be deleted.

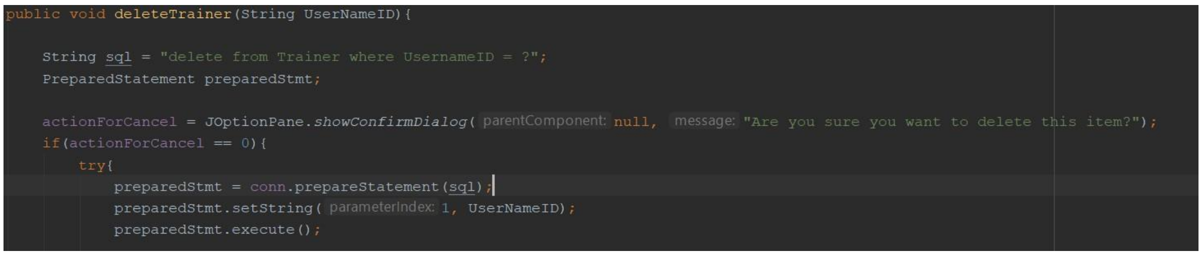


```
if(actionEvent.getSource() == btnDelete){
    trainerSQL.deleteTrainer(this.FieldUsernameIDLabel.getText());
    if(trainerSQL.getActionForCancel() == 0){
        mainController.getPnTrainerItems().getChildren().remove(this.itemT);
        mainController.getNorTrainer().setText(String.valueOf(Integer.valueOf(mainController.getNorTrainer().getText())-1));

    }
}
```

Figure 6: Process of pressing the delete button

We call the deleteTrainer() function to delete a spesific trainer. In this function, we use PreparedStatement interface that is subinterface of Statement. We use it to execute parameterized query and it improved the performance of the application because query is compiled only once.



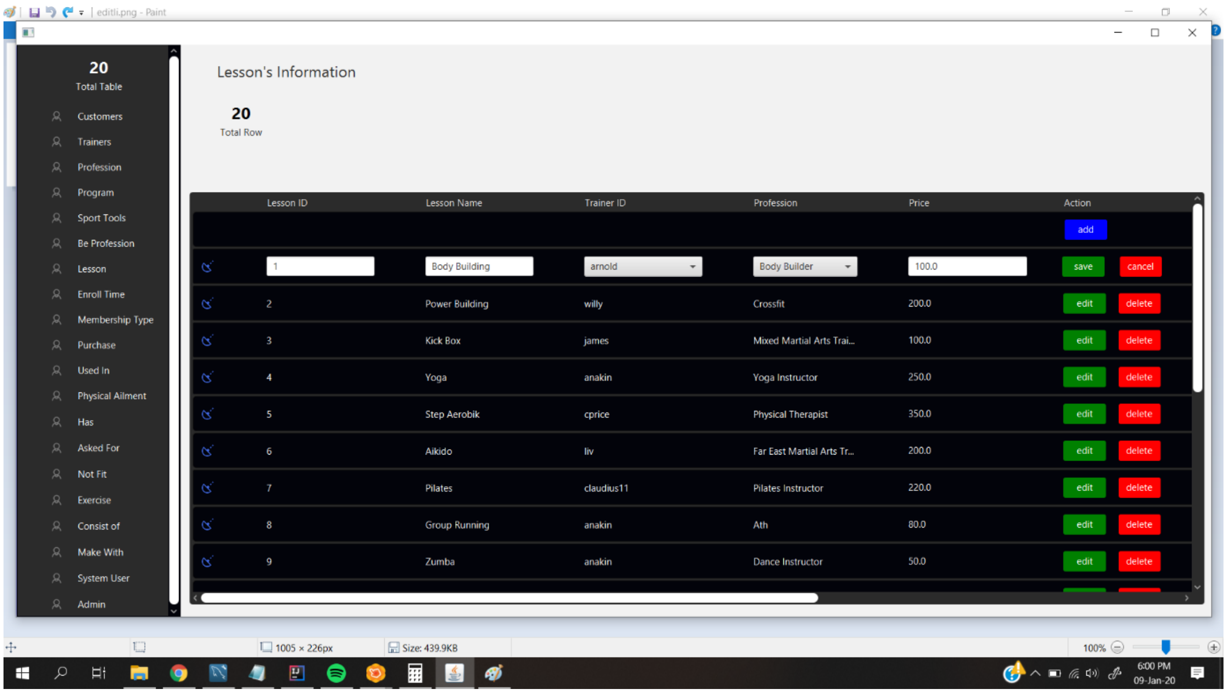
```
public void deleteTrainer(String UserNameID){

    String sql = "delete from Trainer where UsernameID = ?";
    PreparedStatement preparedStmt;

    actionForCancel = JOptionPane.showConfirmDialog( parentComponent: null,  message: "Are you sure you want to delete this item?");
    if(actionForCancel == 0){
        try{
            preparedStmt = conn.prepareStatement(sql);
            preparedStmt.setString( parameterIndex: 1, UserNameID);
            preparedStmt.execute();
```

Figure 7: deleteTrainer() function using PreparedStatement interface

When the edit button is pressed, the method of getSource() of ActionEvent object will be equal to btnEdit and the new TextField object appears in the interface and the admin can update the data according to choice box. This choice box contains the available foreign keys of the row that is attempted to be updated.



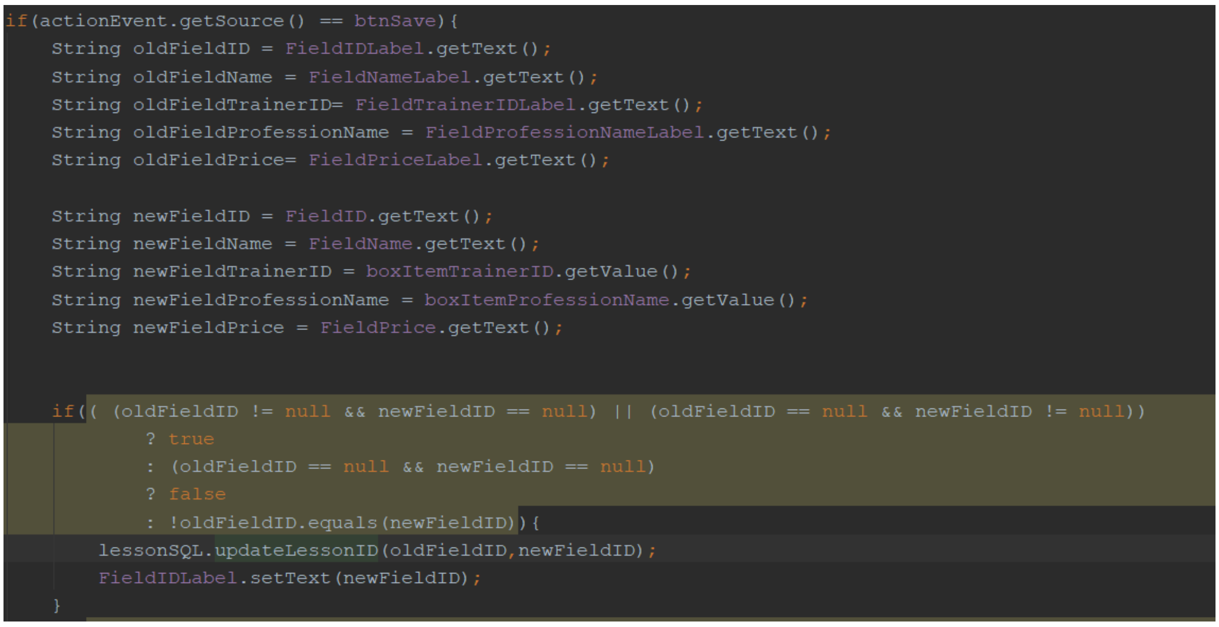
Lesson's Information

20
Total Row

Lesson ID	Lesson Name	Trainer ID	Profession	Price	Action
					add
1	Body Building	arnold	Body Builder	100.0	save cancel
2	Power Building	willy	Crossfit	200.0	edit delete
3	Kick Box	james	Mixed Martial Arts Trai...	100.0	edit delete
4	Yoga	anakin	Yoga Instructor	250.0	edit delete
5	Step Aerobik	cprice	Physical Therapist	350.0	edit delete
6	Aikido	liv	Far East Martial Arts Tr...	200.0	edit delete
7	Pilates	claudius11	Pilates Instructor	220.0	edit delete
8	Group Running	anakin	Ath	80.0	edit delete
9	Zumba	anakin	Dance Instructor	50.0	edit delete

Figure 8: User interface when the edit button is pressed

If the save button is pressed after updating row, the updated data is replaced with the old values and the Label field appears again with the updated values in the interface. The related code for this purpose is shown below.



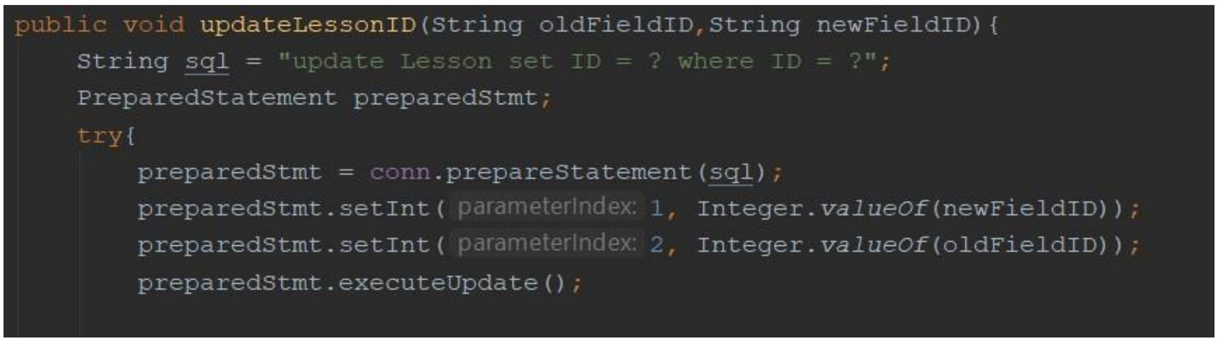
```
if(actionEvent.getSource() == btnSave){
    String oldFieldID = FieldIDLabel.getText();
    String oldFieldName = FieldNameLabel.getText();
    String oldFieldTrainerID= FieldTrainerIDLabel.getText();
    String oldFieldProfessionName = FieldProfessionNameLabel.getText();
    String oldFieldPrice= FieldPriceLabel.getText();

    String newFieldID = FieldID.getText();
    String newFieldName = FieldName.getText();
    String newFieldTrainerID = boxItemTrainerID.getValue();
    String newFieldProfessionName = boxItemProfessionName.getValue();
    String newFieldPrice = FieldPrice.getText();


    if(( (oldFieldID != null && newFieldID == null) || (oldFieldID == null && newFieldID != null))
            ? true
            : (oldFieldID == null && newFieldID == null)
            ? false
            : !oldFieldID.equals(newFieldID)){
        lessonSQL.updateLessonID(oldFieldID,newFieldID);
        FieldIDLabel.setText(newFieldID);
    }
```

Figure 9: The events when the save button is pressed

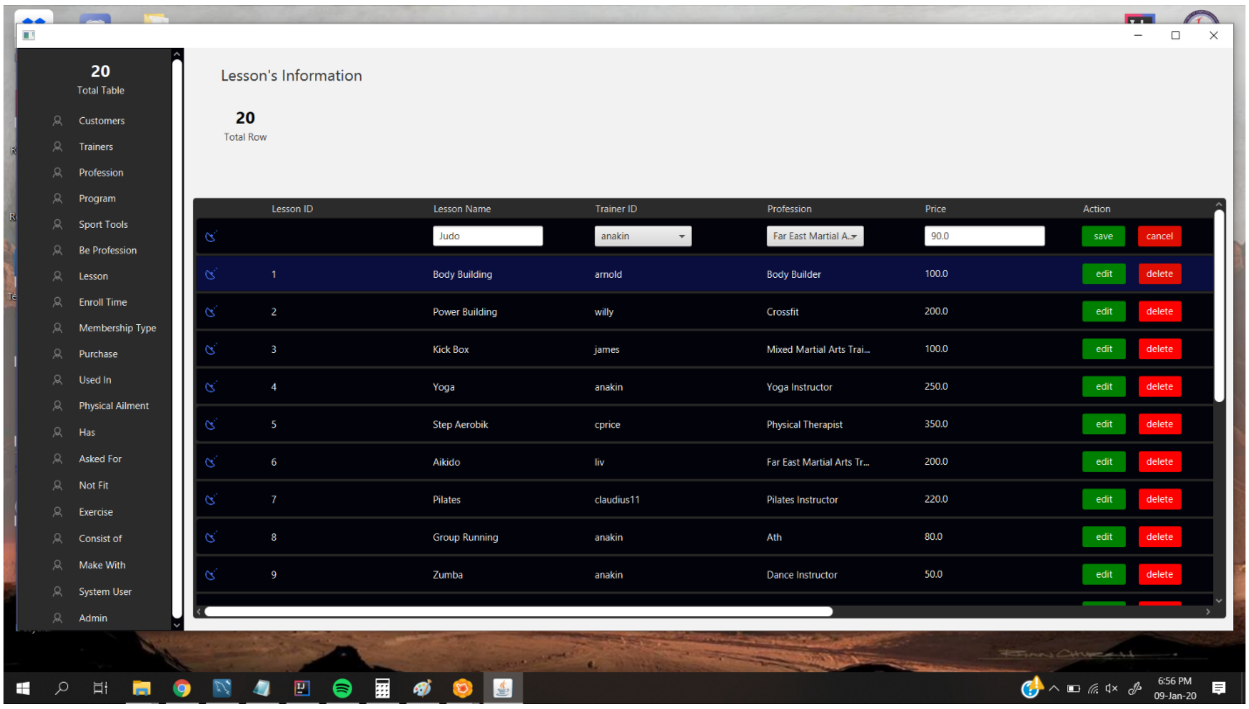
We created update functions for each attribute of each row. For example, in the updateLessonID() function we delete the lesson according to its ID. Some part of the related code for updateLessonID() function is shown in below.



```
public void updateLessonID(String oldFieldID,String newFieldID){
    String sql = "update Lesson set ID = ? where ID = ?";
    PreparedStatement preparedStmt;
    try{
        preparedStmt = conn.prepareStatement(sql);
        preparedStmt.setInt( parameterIndex: 1, Integer.valueOf(newFieldID));
        preparedStmt.setInt( parameterIndex: 2, Integer.valueOf(oldFieldID));
        preparedStmt.executeUpdate();
```

Figure 10: updateLessonID() function

When the add button is pressed, the method of getSource() of ActionEvent object will be equal to btnAddInsert button and again the TextField appears for the related row and the user can add a new lesson when s/he is pressed the save button. When the new data is wanted to be added in its table, if the id of the data is auto-incremented, id of that data is not entered. Figure provided below.



Lesson's Information

20
Total Row

Lesson ID	Lesson Name	Trainer ID	Profession	Price	Action
	Judo	anakin	Far East Martial A...	90.0	save cancel
1	Body Building	arnold	Body Builder	100.0	edit delete
2	Power Building	willy	Crossfit	200.0	edit delete
3	Kick Box	james	Mixed Martial Arts Trai...	100.0	edit delete
4	Yoga	anakin	Yoga Instructor	250.0	edit delete
5	Step Aerobik	cprice	Physical Therapist	350.0	edit delete
6	Aikido	liv	Far East Martial Arts Tr...	200.0	edit delete
7	Pilates	claudius11	Pilates Instructor	220.0	edit delete
8	Group Running	anakin	Ath	80.0	edit delete
9	Zumba	anakin	Dance Instructor	50.0	edit delete

Figure 11: The event occurs when the add button is pressed

We insert a new data according to the primary key of the table except the tables that have auto-incremented primary key id. In order to achieve this, we created insert function for each table. For example, if a new lesson is intended to be inserted, we simply make insertion operation in the figure above.

If we want to insert new Customer, Admin or Trainer, the system made an insertion operation for System User first by using triggers.

We obtain the count of the table and count of the records in our tables and they can be seen in the upper left corner of the interface.

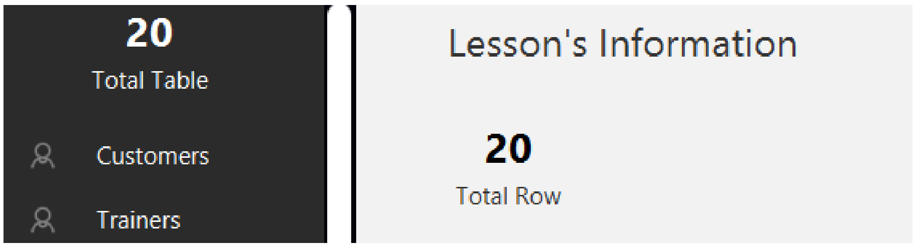
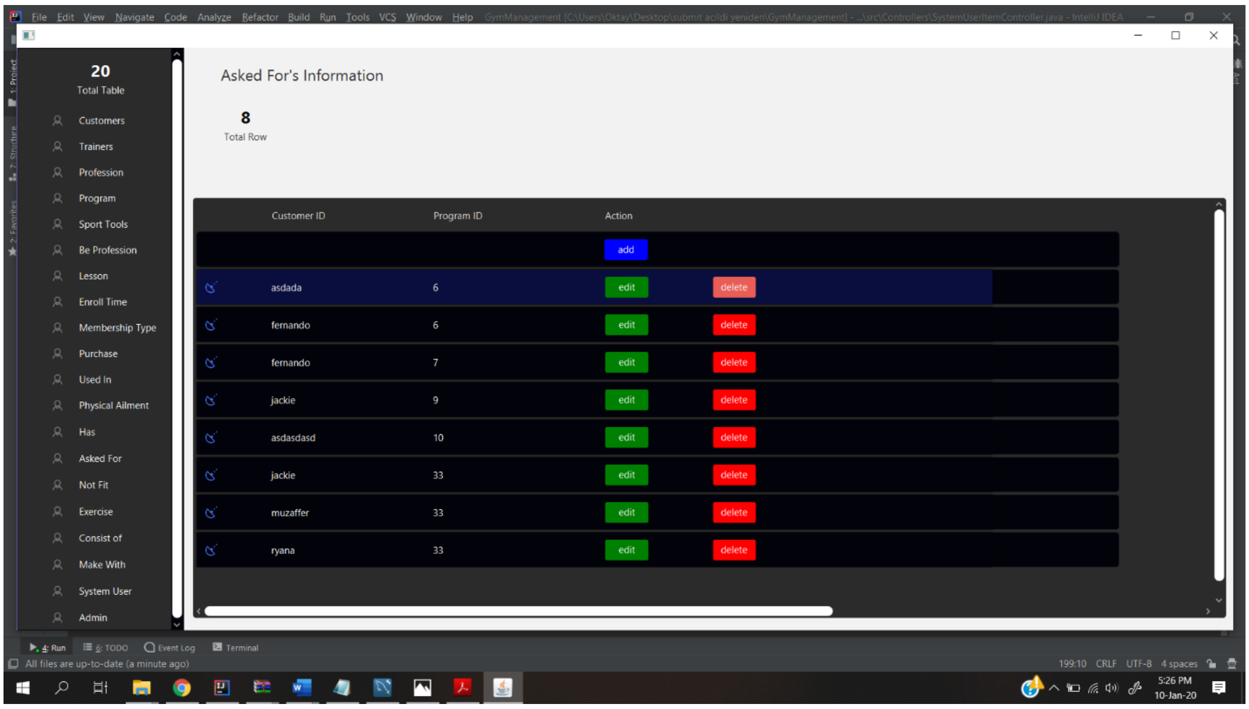


Figure 12: Statistical data from database

When users want to delete some items from table, they can easily press “delete” button as you can see in below Figure 13. After pressing, the message info shows to confirm this deletion and you can press “Yes” or “No”, in below Figure 14. The table re-render and if this deletion cause changes in other tables in the database, also the other tables are re-render after this deletion and the “Total Row” information decrease by 1 as you can see in Figure 15.

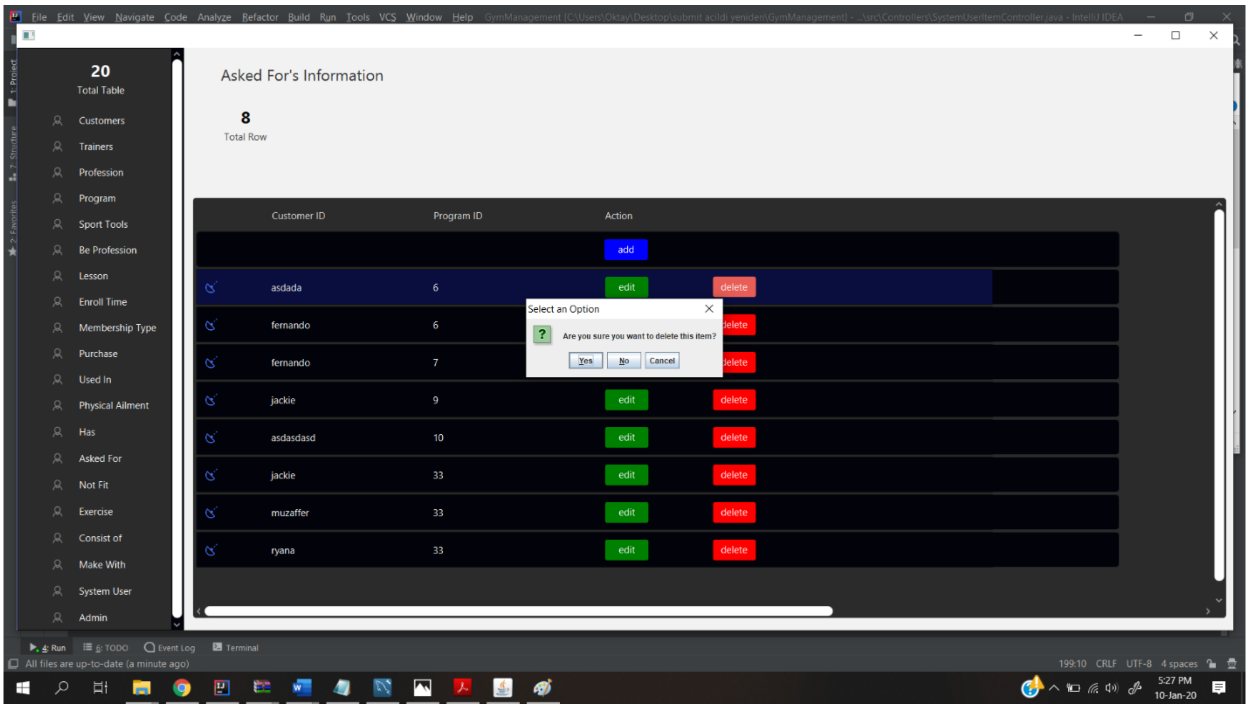


Asked For's Information

8
Total Row

Customer ID	Program ID	Action	
		add	
asdada	6	edit	delete
fernando	6	edit	delete
fernando	7	edit	delete
jackie	9	edit	delete
asdasdasd	10	edit	delete
jackie	33	edit	delete
muzaffer	33	edit	delete
ryana	33	edit	delete

Figure 13: Before deletion of first row in Asked_For table.



Asked For's Information

8
Total Row

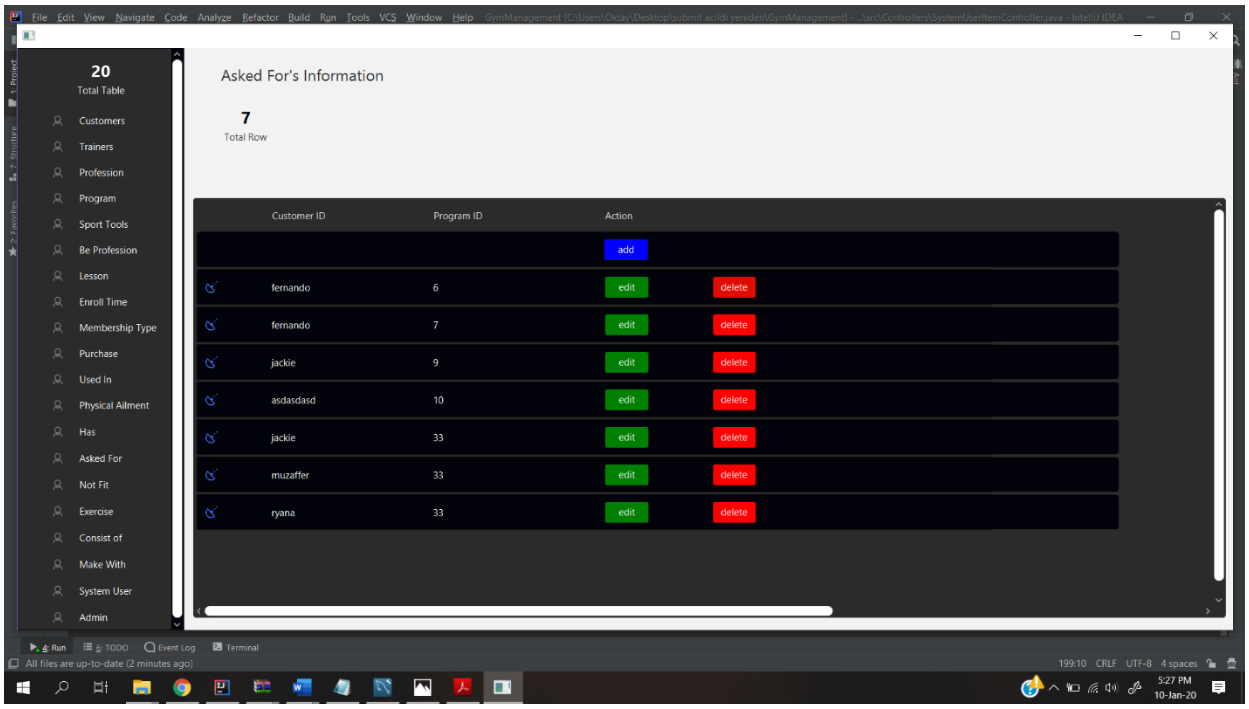
Customer ID	Program ID	Action	
		add	
asdada	6	edit	delete
fernando	6		delete
fernando	7		delete
jackie	9	edit	delete
asdasdasd	10	edit	delete
jackie	33	edit	delete
muzaffer	33	edit	delete
ryana	33	edit	delete

Select an Option

Are you sure you want to delete this item?

Yes No Cancel

Figure 14: Information message to confirm of deletion



Asked For's Information

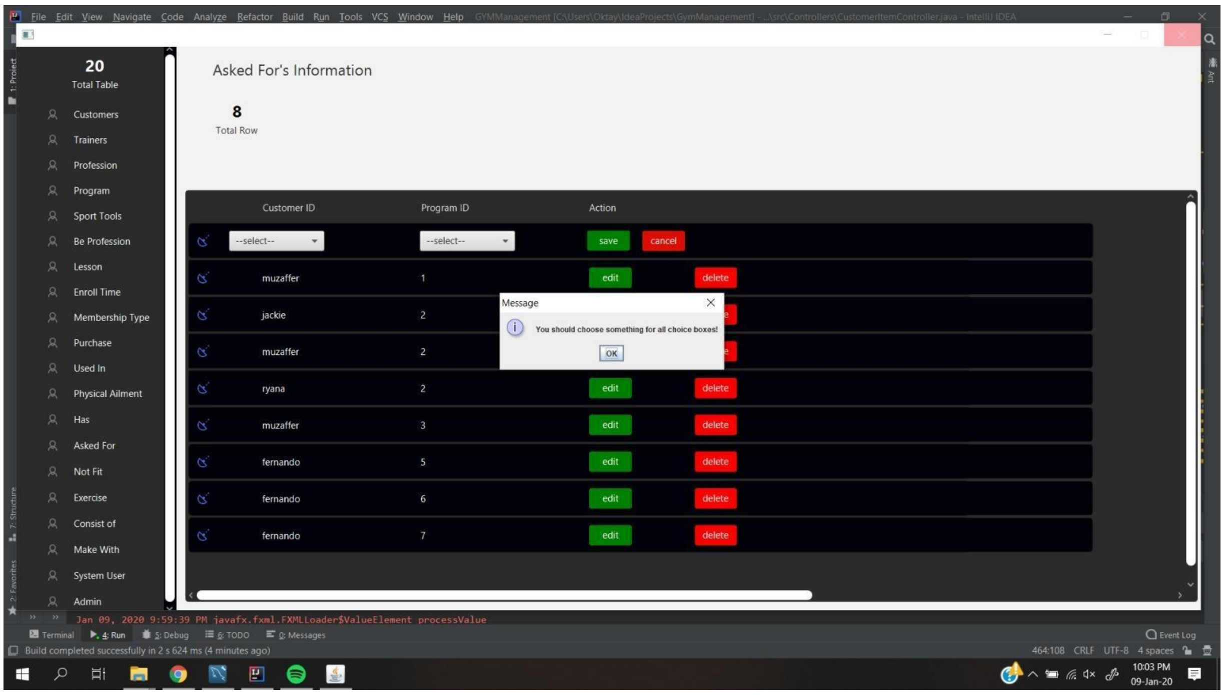
7
Total Row

Customer ID	Program ID	Action	
		add	
fernando	6	edit	delete
fernando	7	edit	delete
jackie	9	edit	delete
asdasdasd	10	edit	delete
jackie	33	edit	delete
muzaffer	33	edit	delete
ryana	33	edit	delete

All files are up-to-date (2 minutes ago)

Figure 15: After deletion of first item in this table.

Choice boxes allow us to select foreign keys in the system. If the primary key of a table is these foreign keys, the admin cannot leave this choice box blank when she/he wants to do any operation. If it is done by user, the error massage appears in the screen.



Asked For's Information

8
Total Row

Customer ID	Program ID	Action	
--select--	--select--	save	cancel
muzaffer	1	edit	delete
jackie	2		
muzaffer	2		
ryana	2	edit	delete
muzaffer	3	edit	delete
fernando	5	edit	delete
fernando	6	edit	delete
fernando	7	edit	delete

Message

You should choose something for all choice boxes!

OK

```
Jan 09, 2020 9:59:39 PM javafx.fxml.FXMLLoader$ValueElement processValue
```

Build completed successfully in 2 s 624 ms (4 minutes ago)

Figure 16: Error Message

4 About Data Normalization

When we design the project, we have taken care to optimize the progress of the project. All the information of the tables are designed to be consistent and not to create redundancy. So, all tables satisfy 1NF, 2NF, 3NF and BCNF conditions and we do not need to make data normalization in this phase. In addition to the previous database design, an admin user has been added and other data types have been modified. The diagram that we shared with the name diagram.png in the zip file shows those changes.