МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
Кафедра ИИТ

ЛАБОРАТОРНАЯ РАБОТА №10

По дисциплине: «Современные платформы программирования»

Выполнил:
Студент ФЭИС
3-го курса, группы ПО-5
Прокопчик Е.А.
Проверил:
Крощенко А.А.

Брест 2022

Цель работы: приобрести практические навыки разработки многооконных приложений на JavaFX для работы с базами данных

На основе БД, разработанной в лабораторной работе №9, реализовать многооконное приложениеклиент, позволяющее выполнять основные операции над таблицей в БД (добавление, удаление, модификацию данных).

Основные требования к приложению:

• Для отображения выбирать таблицу с внешними ключами;

• Осуществлять вывод основных данных в табличном представлении;

• При выводе краткого представления записи в таблице (т.е. если выводятся не все поля), по щелчку мышкой на запись осуществлять вывод всех полей в подготовленные компоненты на форме;

• Для всех полей, представленных внешними ключами, выводить их текстовое представление из связанных таблиц (например, таблица-справочник «Времена года» содержит два поля – идентификатор и название сезона, в связанной таблице «Месяц года» есть внешний ключ на таблицу «Времена года»; в этом случае при выводе таблицы «Месяц года» нужно выводить название сезона, а не его идентификатор);

• При выводе предусмотреть упорядочивание по столбцу;

• Реализовать простейший фильтр данных по одному-двум полям;

• При добавлении новых данных в таблицу использовать дополнительное окно для ввода;
• При модификации данных можно использовать ту же форму, что и для добавления, но с внесенными актуальными значениями полей;

• При добавлении/модификации выводить варианты значений полей с внешним ключом с помощью выпадающего списка;

• При удалении данных осуществлять удаление записи, на которой в данных момент находится фокус.

12) База данных «Европейские футбольные чемпионаты»

Код программы:

App.java

```
package com.example;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;
```

```
import java.io.IOException;


public final class App extends Application {
   private static Scene scene;

   @Override
   public final void start(final Stage stage) throws IOException {
      App.scene = new Scene(App.loadFXML("main"), 640, 480);
      stage.setResizable(false);
      stage.setScene(App.scene);
      stage.show();
   }

   public final static Parent loadFXML(final String fxml) throws IOException {
      FXMLLoader fxmlLoader = new FXMLLoader(App.class.getResource(fxml + ".fxml"));
      return fxmlLoader.load();
   }

   public final static void main(final String[] args) {
      launch();
   }
}
```

CreateController.java

```
package com.example;


import java.net.URL;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ResourceBundle;

import java.util.logging.Level;

import java.util.logging.Logger;


import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.fxml.FXML;

import javafx.fxml.Initializable;

import javafx.scene.control.ChoiceBox;


public final class CreateController implements Initializable {

   @FXML
```

```
private ChoiceBox<String> leaguesChoiceBox;
@FXML
private ChoiceBox<String> stadiumsChoiceBox;
@FXML
private ChoiceBox<String> team1ChoiceBox;
@FXML
private ChoiceBox<String> team2ChoiceBox;

private DB db = null;

@Override
public void initialize(URL arg0, ResourceBundle arg1) {
  try {
    this.db = new DB();

    ObservableList<String> leaguesList = FXCollections.observableArrayList();
    ObservableList<String> stadiumsList = FXCollections.observableArrayList();
    ObservableList<String> team1List = FXCollections.observableArrayList();
    ObservableList<String> team2List = FXCollections.observableArrayList();

    ResultSet leaguesSet = this.db.getAll(DB.LEAGUES_TABLE);
    ResultSet stadiumsSet = this.db.getAll(DB.STADIUMS_TABLE);
    ResultSet team1Set = this.db.getAll(DB.TEAM1_TABLE);
    ResultSet team2Set = this.db.getAll(DB.TEAM2_TABLE);

    while (leaguesSet.next() && stadiumsSet.next() && team1Set.next() && team2Set.next()) {
      leaguesList.add(leaguesSet.getString("short_name"));
      stadiumsList.add(stadiumsSet.getString("short_name"));
      team1List.add(team1Set.getString("name"));
      team2List.add(team2Set.getString("name"));
    }

    this.leaguesChoiceBox.setItems(leaguesList);
    this.stadiumsChoiceBox.setItems(stadiumsList);
    this.team1ChoiceBox.setItems(team1List);
```

```
            this.team2ChoiceBox.setItems(team2List);
        } catch (final SQLException exception) {
            Logger.getLogger(CreateController.class.getName()).log(Level.SEVERE, null, exception);
        } catch (final Exception exception) {
            Logger.getLogger(CreateController.class.getName()).log(Level.SEVERE, null, exception);
        }
    }

    @FXML
    private final void create() {
        if (this.leaguesChoiceBox.getSelectionModel().isEmpty()
                || this.stadiumsChoiceBox.getSelectionModel().isEmpty()
                || this.team1ChoiceBox.getSelectionModel().isEmpty()
                || this.team2ChoiceBox.getSelectionModel().isEmpty()) {
            return;
        }

        this.db.addMatch(new Match(null,
                this.leaguesChoiceBox.getValue(),
                this.stadiumsChoiceBox.getValue(),
                null,
                this.team1ChoiceBox.getValue(),
                this.team2ChoiceBox.getValue()));
    }
}
```

DB.java

```
package com.example;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.logging.Level;
import java.util.logging.Logger;

public final class DB {
    private final static String DATABASE_NAME = "lab10spp";
    public final static String LEAGUES_TABLE = "leagues";
```

```
    public final static String STADIUMS_TABLE = "stadiums";
    public final static String TEAM1_TABLE = "team1";
    public final static String TEAM2_TABLE = "team2";
    public final static String MATCHES_TABLE = "matches";

    private Connection connection = null;

    public DB() {
        try {
            final String HOST = "localhost";
            final String PORT = "3306";
            final String USERNAME = "root";
            final String PASSWORD = "root";
            final String URL = "jdbc:mysql://" + HOST + ':' + PORT;

            //Class.forName("com.mysql.cj.jdbc.Driver");
            this.connection = DriverManager.getConnection(URL + "/?user=" + USERNAME + "&password=" +
PASSWORD);
            this.prepare();
            //this.fill();
        } /*catch (final ClassNotFoundException exception) {
            Logger.getLogger(DB.class.getName()).log(Level.SEVERE, null, exception);
        }*/ catch (final SQLException exception) {
            Logger.getLogger(DB.class.getName()).log(Level.SEVERE, null, exception);
        } /*catch (final Exception exception) {
            Logger.getLogger(DB.class.getName()).log(Level.SEVERE, null, exception);
        } */
    }

    public final void close() {
        try {
            this.connection.close();
        } catch (final SQLException exception) {
            Logger.getLogger(DB.class.getName()).log(Level.SEVERE, null, exception);
        }
    }

    private final void prepare() {
        try {
            final String[] preparation = {
                new String("CREATE DATABASE IF NOT EXISTS `" + DB.DATABASE_NAME + "`;"),
                new String("CREATE TABLE IF NOT EXISTS `" + DB.DATABASE_NAME + "`.`"
                    + DB.LEAGUES_TABLE
                    + "` ( `id` INT UNSIGNED NOT NULL AUTO_INCREMENT , `name` VARCHAR(64) NOT NULL , "
                    + "`country` VARCHAR(64) NOT NULL , `short_name` VARCHAR(64) NOT NULL , PRIMARY KEY (`id`) ,
"
                    + "UNIQUE (`short_name`) ) ENGINE = InnoDB;"),
                new String("CREATE TABLE IF NOT EXISTS `" + DB.DATABASE_NAME + "`.`"
                    + DB.STADIUMS_TABLE
                    + "` ( `id` INT UNSIGNED NOT NULL AUTO_INCREMENT , `name` VARCHAR(64) NOT NULL , "
                    + "`country` VARCHAR(64) NOT NULL , `short_name` VARCHAR(64) NOT NULL , "
                    + "PRIMARY KEY (`id`) , UNIQUE (`short_name`) ) ENGINE = InnoDB;"),
                new String("CREATE TABLE IF NOT EXISTS `" + DB.DATABASE_NAME + "`.`" +
                    DB.TEAM1_TABLE
                    + "` ( `id` INT UNSIGNED NOT NULL AUTO_INCREMENT , `name` VARCHAR(64) NOT NULL , "
                    + "`description` TEXT NULL , PRIMARY KEY (`id`) , UNIQUE (`name`) ) ENGINE = InnoDB;"),
```

```
            new String("CREATE TABLE IF NOT EXISTS `" + DB.DATABASE_NAME + "`.`" +
                DB.TEAM2_TABLE
                + "` ( `id` INT UNSIGNED NOT NULL AUTO_INCREMENT , `name` VARCHAR(64) NOT NULL , "
                + "`description` TEXT NULL , PRIMARY KEY (`id`) , UNIQUE (`name`) ) ENGINE = InnoDB;"),
            new String("CREATE TABLE IF NOT EXISTS `" + DB.DATABASE_NAME + "`.`" +
                DB.MATCHES_TABLE
                + "` ( `id` INT UNSIGNED NOT NULL AUTO_INCREMENT , `league_id` INT UNSIGNED NOT NULL , "
                + "`stadium_id` INT UNSIGNED NOT NULL , `date` DATETIME NOT NULL DEFAULT
CURRENT_TIMESTAMP , "
                + "`team1_id` INT UNSIGNED NOT NULL , `team2_id` INT UNSIGNED NOT NULL , "
                + "PRIMARY KEY (`id`) , INDEX `league_id_index` (`league_id`) , INDEX `stadium_id_index`
(`stadium_id`) , "
                + "INDEX `team1_id_index` (`team1_id`) , INDEX `team2_id_index` (`team2_id`) ) ENGINE =
InnoDB;"),
            new String("ALTER TABLE `" + DB.DATABASE_NAME + "`.`" + DB.MATCHES_TABLE
                + "` ADD FOREIGN KEY ( `league_id` ) REFERENCES `"
                + DB.LEAGUES_TABLE
                + "` (`id`) ON DELETE CASCADE ON UPDATE RESTRICT ;"),
            new String("ALTER TABLE `" + DB.DATABASE_NAME + "`.`" + DB.MATCHES_TABLE
                + "` ADD FOREIGN KEY ( `stadium_id` ) REFERENCES `"
                + DB.STADIUMS_TABLE
                + "` (`id`) ON DELETE CASCADE ON UPDATE RESTRICT ;"),
            new String("ALTER TABLE `" + DB.DATABASE_NAME + "`.`" + DB.MATCHES_TABLE
                + "` ADD FOREIGN KEY ( `team1_id` ) REFERENCES `" + DB.TEAM1_TABLE
                + "` (`id`) ON DELETE CASCADE ON UPDATE RESTRICT ;"),
            new String("ALTER TABLE `" + DB.DATABASE_NAME + "`.`" + DB.MATCHES_TABLE
                + "` ADD FOREIGN KEY ( `team2_id` ) REFERENCES `" + DB.TEAM2_TABLE
                + "` (`id`) ON DELETE CASCADE ON UPDATE RESTRICT ;"),
            new String("USE `" + DB.DATABASE_NAME + "`;")
        };

        Statement statement = this.connection.createStatement();
        statement.closeOnCompletion();

        for (final String sql : preparation) {
          statement.executeUpdate(sql);
        }
      } catch (final Exception exception) {
        Logger.getLogger(DB.class.getName()).log(Level.SEVERE, null, exception);
      }
    }

    public final ResultSet getAll(final String table) {
      try {
        Statement statement = this.connection.createStatement();
        statement.closeOnCompletion();

        return statement
            .executeQuery(new String(
                "SELECT * FROM `" + DB.DATABASE_NAME + "`.`" + table + "` ORDER BY `id`;"));
      } catch (final SQLException exception) {
        Logger.getLogger(DB.class.getName()).log(Level.SEVERE, null, exception);
      }

      return null;
    }
```

```
  public final ResultSet getMatches(final String leagueFilter, final String stadiumFilter, final Integer id) {
    try {
      Statement statement = this.connection.createStatement();
      statement.closeOnCompletion();

      String sql = new String("SELECT "
          + "`matches`.`id` , "
          + "`leagues`.`short_name` AS `league` , "
          + "`stadiums`.`short_name` AS `stadium` , "
          + "`matches`.`date` , "
          + "`team1`.`name` AS `team1` , "
          + "`team2`.`name` AS `team2` "
          + "FROM `matches` "
          + "INNER JOIN `leagues` ON `matches`.`league_id` = `leagues`.`id` "
          + "INNER JOIN `stadiums` ON `matches`.`stadium_id` = `stadiums`.`id` "
          + "INNER JOIN `team1` ON `matches`.`team1_id` = `team1`.`id` "
          + "INNER JOIN `team2` ON `matches`.`team2_id` = `team2`.`id`");

      if (id != null) {
        sql += " WHERE `matches`.`id` = '" + Integer.toString(id) + "';";
        return statement.executeQuery(sql);
      }

      if (leagueFilter != null && stadiumFilter != null) {
        sql += " WHERE `leagues`.`short_name` = '" + leagueFilter + "' AND `stadiums`.`short_name` = '" +
stadiumFilter
            + "';";
        return statement.executeQuery(sql);
      }

      if (leagueFilter != null) {
        sql += " WHERE `leagues`.`short_name` = '" + leagueFilter + "';";
        return statement.executeQuery(sql);
      }

      if (stadiumFilter != null) {
        sql += " WHERE `stadiums`.`short_name` = '" + stadiumFilter + "';";
        return statement.executeQuery(sql);
      }

      return statement.executeQuery(sql + ';');
    } catch (final SQLException exception) {
      Logger.getLogger(DB.class.getName()).log(Level.SEVERE, null, exception);
    }

    return null;
  }

  public final ResultSet getMatchDetails(final Integer id) {
    try {
      Statement statement = this.connection.createStatement();
      statement.closeOnCompletion();

      return statement
          .executeQuery(new String("SELECT "
```

```
                    + "`leagues`.`name` AS `league_name` , "
                    + "`leagues`.`country` AS `league_country` , "
                    + "`stadiums`.`name` AS `stadium_name` , "
                    + "`stadiums`.`country` AS `stadium_country` , "
                    + "`team1`.`description` AS `team1_description` , "
                    + "`team2`.`description` AS `team2_description` "
                    + "FROM `matches` "
                    + "INNER JOIN `leagues` ON `matches`.`league_id` = `leagues`.`id` "
                    + "INNER JOIN `stadiums` ON `matches`.`stadium_id` = `stadiums`.`id` "
                    + "INNER JOIN `team1` ON `matches`.`team1_id` = `team1`.`id` "
                    + "INNER JOIN `team2` ON `matches`.`team2_id` = `team2`.`id` "
                    + "WHERE `matches`.`id` = '" + Integer.toString(id) + "';"));
        } catch (final SQLException exception) {
            Logger.getLogger(DB.class.getName()).log(Level.SEVERE, null, exception);
        }

        return null;
    }

    public final void deleteByID(final String table, final Integer id) {
        try {
            Statement statement = this.connection.createStatement();
            statement.closeOnCompletion();
            statement.executeUpdate(
                    "DELETE FROM `" + DB.DATABASE_NAME + "`.`" + table + "` WHERE `id` = '" + id + "\';");
        } catch (final SQLException exception) {
            Logger.getLogger(DB.class.getName()).log(Level.SEVERE, null, exception);
        }
    }

    public final void addMatch(final Match match) {
        try {
            Statement statement = this.connection.createStatement();
            statement.closeOnCompletion();
            ResultSet resultSet = null;

            resultSet = statement.executeQuery(
                    "SELECT `leagues`.`id` FROM `leagues` WHERE `short_name` = '" + match.getLeague() + "';");
            resultSet.next();
            Integer leagueId = resultSet.getInt("id");

            resultSet = statement
                    .executeQuery("SELECT `stadiums`.`id` FROM `stadiums` WHERE `short_name` = '" +
match.getStadium() + "';");
            resultSet.next();
            Integer stadiumId = resultSet.getInt("id");

            resultSet = statement.executeQuery("SELECT `team1`.`id` FROM `team1` WHERE `name` = '" +
match.getTeam1() + "';");
            resultSet.next();
            Integer team1Id = resultSet.getInt("id");

            resultSet = statement.executeQuery("SELECT `team2`.`id` FROM `team2` WHERE `name` = '" +
match.getTeam2() + "';");
            resultSet.next();
            Integer team2Id = resultSet.getInt("id");
```

```
        String query = null;

        if (match.getId() == null) {
           query = new String("INSERT INTO `" + DB.MATCHES_TABLE
                 + "` (`id`, `league_id`, `stadium_id`, `date`, `team1_id`, `team2_id`) "
                 + "VALUES (NULL, '" + leagueId + "', '" + stadiumId + "', current_timestamp(), "
                 +Integer.toString(team1Id) + "', '" + Integer.toString(team2Id) + "');");

           statement.executeUpdate(query);
           return;
        }

         query = new String("UPDATE `" + DB.MATCHES_TABLE + "` SET "
              + "`league_id` = '" + Integer.toString(leagueId)
              + "' , `stadium_id` = '" + Integer.toString(stadiumId)
              + "' , `date` = current_timestamp() , "
              + "' , `team1_id` = '" + Integer.toString(team1Id)
              + "' , `team2_id` = '" + Integer.toString(team2Id)
              + "' WHERE `matches`.`id` = '" + Integer.toString(match.getId()) + "';");


        statement.executeUpdate(query);
     } catch (final SQLException exception) {
        Logger.getLogger(DB.class.getName()).log(Level.SEVERE, null, exception);
     }
  }

  private final void fill() {
     this.fillLeagues();
     this.fillStadiums();
     this.fillTeam1();
     this.fillTeam2();
     this.fillMatches();
  }

  private final void fillLeagues() {
     try {
        String[] leagues = {
              new String("INSERT INTO `" + DB.LEAGUES_TABLE
                    + "` (`id`, `name`, `country`, `short_name`) VALUES (NULL, 'Belarusian Premier League', 'Belarus',
'BPL');"),
              new String("INSERT INTO `" + DB.LEAGUES_TABLE
                    + "` (`id`, `name`, `country`, `short_name`) VALUES (NULL, 'Bundesliga', 'Germany', 'BL');"),
              new String("INSERT INTO `" + DB.LEAGUES_TABLE
                    + "` (`id`, `name`, `country`, `short_name`) VALUES (NULL, 'UEFA', 'Europe', 'UEFA');")
        };

        Statement statement = this.connection.createStatement();
        statement.closeOnCompletion();

        for (final String sql : leagues) {
           statement.executeUpdate(sql);
        }
     } catch (final SQLException exception) {
        Logger.getLogger(DB.class.getName()).log(Level.SEVERE, null, exception);
```

```
    }
}

private final void fillStadiums() {
    try {
        String[] stadiums = {
                new String("INSERT INTO `" + DB.STADIUMS_TABLE
                        + "` (`id`, `name`, `country`, `short_name`) VALUES (NULL, 'Barysaŭ-Arena', 'Belarus', 'BA');"),
                new String("INSERT INTO `" + DB.STADIUMS_TABLE
                        + "` (`id`, `name`, `country`, `short_name`) VALUES (NULL, 'Westfalenstadion', 'Germany', 'WS');"),
                new String("INSERT INTO `" + DB.STADIUMS_TABLE
                        + "` (`id`, `name`, `country`, `short_name`) VALUES (NULL, 'Allianz Arena', 'Germany', 'AA');"),
                new String("INSERT INTO `" + DB.STADIUMS_TABLE
                        + "` (`id`, `name`, `country`, `short_name`) VALUES (NULL, 'Enfield', 'Enland', 'EF');"),
                new String("INSERT INTO `" + DB.STADIUMS_TABLE
                        + "` (`id`, `name`, `country`, `short_name`) VALUES (NULL, 'Santiago-Bernabéu', 'Spain', 'SB');")
        };

        Statement statement = this.connection.createStatement();
        statement.closeOnCompletion();

        for (final String sql : stadiums) {
            statement.executeUpdate(sql);
        }
    } catch (final SQLException exception) {
        Logger.getLogger(DB.class.getName()).log(Level.SEVERE, null, exception);
    }
}

private final void fillTeam1() {
    try {
        String[] team1 = {
                new String("INSERT INTO `" + DB.TEAM1_TABLE
                        + "` (`id`, `name`, `description`) VALUES (NULL, 'Dynamo Minsk', "
                        + "'Football team from Minsk, Belarus.');"),
                new String("INSERT INTO `" + DB.TEAM1_TABLE
                        + "` (`id`, `name`, `description`) VALUES (NULL, 'Borussia', "
                        + "'Football team from Dortmund, Germany.');"),
                new String("INSERT INTO `" + DB.TEAM1_TABLE
                        + "` (`id`, `name`, `description`) VALUES (NULL, 'VfB Stuttgart', "
                        + "'Football team from Stuttgart, Germany.');"),
                new String("INSERT INTO `" + DB.TEAM1_TABLE
                        + "` (`id`, `name`, `description`) VALUES (NULL, 'Villarreal', "
                        + "'Football team from Villarreal, Spain.');"),
                new String("INSERT INTO `" + DB.TEAM1_TABLE
                        + "` (`id`, `name`, `description`) VALUES (NULL, 'Real Madrid', "
                        + "'Football team from Madrid, Spain.');")
        };

        Statement statement = this.connection.createStatement();
        statement.closeOnCompletion();

        for (final String sql : team1) {
            statement.executeUpdate(sql);
        }
    } catch (final SQLException exception) {
```

```
            Logger.getLogger(DB.class.getName()).log(Level.SEVERE, null, exception);
        }
    }

    private final void fillTeam2() {
        try {
            String[] team2 = {
                new String("INSERT INTO `" + DB.TEAM2_TABLE
                    + "` (`id`, `name`, `description`) VALUES (NULL, 'BATE', "
                    + "'Football team from Barysau, Belarus.');"),
                new String("INSERT INTO `" + DB.TEAM2_TABLE
                    + "` (`id`, `name`, `description`) VALUES (NULL, 'FC Bayern München', "
                    + "'Football team from München, Germany.');"),
                new String("INSERT INTO `" + DB.TEAM2_TABLE
                    + "` (`id`, `name`, `description`) VALUES (NULL, 'VfL Wolfsburg', "
                    + "'Football team from Wolfsburg, Germany.');"),
                new String("INSERT INTO `" + DB.TEAM2_TABLE
                    + "` (`id`, `name`, `description`) VALUES (NULL, 'Liverpool FC', "
                    + "'Football team from Liverpool, England.');"),
                new String("INSERT INTO `" + DB.TEAM2_TABLE
                    + "` (`id`, `name`, `description`) VALUES (NULL, 'Manchester City FC', "
                    + "'Football team from Manchester, England.');")
            };

            Statement statement = this.connection.createStatement();
            statement.closeOnCompletion();

            for (final String sql : team2) {
                statement.executeUpdate(sql);
            }
        } catch (final SQLException exception) {
            Logger.getLogger(DB.class.getName()).log(Level.SEVERE, null, exception);
        }
    }

    private final void fillMatches() {
        try {
            String[] matches = {
                new String("INSERT INTO `" + DB.MATCHES_TABLE
                    + "` (`id`, `league_id`, `stadium_id`, `date`, `team1_id`, `team2_id`) "
                    + "VALUES (NULL, '1', '1', current_timestamp(), '1', '1');"),
                new String("INSERT INTO `" + DB.MATCHES_TABLE
                    + "` (`id`, `league_id`, `stadium_id`, `date`, `team1_id`, `team2_id`) "
                    + "VALUES (NULL, '2', '2', current_timestamp(), '2', '2');"),
                new String("INSERT INTO `" + DB.MATCHES_TABLE
                    + "` (`id`, `league_id`, `stadium_id`, `date`, `team1_id`, `team2_id`) "
                    + "VALUES (NULL, '2', '3', current_timestamp(), '3', '3');"),
                new String("INSERT INTO `" + DB.MATCHES_TABLE
                    + "` (`id`, `league_id`, `stadium_id`, `date`, `team1_id`, `team2_id`) "
                    + "VALUES (NULL, '3', '4', current_timestamp(), '4', '4');"),
                new String("INSERT INTO `" + DB.MATCHES_TABLE
                    + "` (`id`, `league_id`, `stadium_id`, `date`, `team1_id`, `team2_id`) "
                    + "VALUES (NULL, '3', '5', current_timestamp(), '5', '5');")
            };

            Statement statement = this.connection.createStatement();
```

```
            statement.closeOnCompletion();

            for (final String sql : matches) {
                statement.executeUpdate(sql);
            }
        } catch (final SQLException exception) {
            Logger.getLogger(DB.class.getName()).log(Level.SEVERE, null, exception);
        }
    }
}
```

MainController.java

```
package com.example;

import java.io.IOException;
import java.net.URL;
import java.sql.Date;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ResourceBundle;
import java.util.logging.Level;
import java.util.logging.Logger;

import javafx.beans.value.ChangeListener;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.ChoiceBox;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.TextArea;
import javafx.scene.control.TextField;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.stage.Stage;
import javafx.stage.StageStyle;

public final class MainController implements Initializable {
    @FXML
    private TableView<Match> matchesTableView;
    @FXML
    private TableColumn<Match, String> matchesTableViewId;
    @FXML
    private TableColumn<Match, String> matchesTableViewLeague;
    @FXML
    private TableColumn<Match, String> matchesTableViewStadium;
    @FXML
    private TableColumn<Match, Date> matchesTableViewDate;
    @FXML
    private TableColumn<Match, String> matchesTableViewTeam1;
```

```
@FXML
private TableColumn<Match, String> matchesTableViewTeam2;
@FXML
private TableColumn<Match, String> matchesTableViewEdit;

@FXML
private TextField leagueNameTextField;
@FXML
private TextField leagueCountryTextField;
@FXML
private TextField stadiumNameTextField;
@FXML
private TextField stadiumCountryTextField;
@FXML
private TextArea team1DescriptionTextArea;
@FXML
private TextArea team2DescriptionTextArea;

@FXML
private ChoiceBox<String> leaguesChoiceBox;
@FXML
private ChoiceBox<String> stadiumsChoiceBox;

private ObservableList<Match> matchesList = null;
private DB db = null;

String leagueFilter = null;
String stadiumFilter = null;

@Override
public void initialize(URL arg0, ResourceBundle arg1) {
  this.matchesList = FXCollections.observableArrayList();
  this.db = new DB();

  this.matchesTableViewId.setCellValueFactory(new PropertyValueFactory<>("id"));
  this.matchesTableViewLeague.setCellValueFactory(new PropertyValueFactory<>("league"));
  this.matchesTableViewStadium.setCellValueFactory(new PropertyValueFactory<>("stadium"));
  this.matchesTableViewDate.setCellValueFactory(new PropertyValueFactory<>("date"));
  this.matchesTableViewTeam1.setCellValueFactory(new PropertyValueFactory<>("team1"));
  this.matchesTableViewTeam2.setCellValueFactory(new PropertyValueFactory<>("team2"));

  ChangeListener<Object> listener = (obs, oldValue, newValue) -> {
    try {
      Match match = this.matchesTableView.getSelectionModel().getSelectedItem();

      if (match == null) {
        return;
      }

      ResultSet matchDetailsSet = this.db.getMatchDetails(match.getId());
      matchDetailsSet.next();

      this.leagueNameTextField.setText(matchDetailsSet.getString("league_name"));
      this.leagueCountryTextField.setText(matchDetailsSet.getString("league_country"));
      this.stadiumNameTextField.setText(matchDetailsSet.getString("stadium_name"));
      this.stadiumCountryTextField.setText(matchDetailsSet.getString("stadium_country"));
```

```
            this.team1DescriptionTextArea.setText(matchDetailsSet.getString("team1_description"));
            this.team2DescriptionTextArea.setText(matchDetailsSet.getString("team2_description"));
        } catch (final SQLException exception) {
            Logger.getLogger(MainController.class.getName()).log(Level.SEVERE, null, exception);
        } catch (final Exception exception) {
            Logger.getLogger(MainController.class.getName()).log(Level.SEVERE, null, exception);
        }
    };

    this.matchesTableView.focusedProperty().addListener(listener);
    this.matchesTableView.getSelectionModel().selectedItemProperty().addListener(listener);
}

@FXML
private final void create() {
    try {
        Parent parent = App.loadFXML("create");
        Stage stage = new Stage();
        stage.setScene(new Scene(parent));
        stage.initStyle(StageStyle.UTILITY);
        stage.show();
    } catch (final IOException exception) {
        Logger.getLogger(MainController.class.getName()).log(Level.SEVERE, null, exception);
    }
}

@FXML
private final void read() {
    try {
        this.clearTextFields();
        this.matchesList.clear();

        ResultSet matchesSet = this.db.getMatches(this.leagueFilter, this.stadiumFilter, null);

        while (matchesSet.next()) {
            this.matchesList.add(new Match(
                    matchesSet.getInt("id"),
                    matchesSet.getString("league"),
                    matchesSet.getString("stadium"),
                    matchesSet.getDate("date"),
                    matchesSet.getString("team1"),
                    matchesSet.getString("team2")));
            this.matchesTableView.setItems(this.matchesList);
        }

        ObservableList<String> leaguesList = FXCollections.observableArrayList();
        ObservableList<String> stadiumsList = FXCollections.observableArrayList();

        ResultSet leaguesSet = this.db.getAll(DB.LEAGUES_TABLE);
        ResultSet stadiumsSet = this.db.getAll(DB.STADIUMS_TABLE);

        while (leaguesSet.next() && stadiumsSet.next()) {
            leaguesList.add(leaguesSet.getString("short_name"));
            stadiumsList.add(stadiumsSet.getString("short_name"));
        }
```

```
      this.leaguesChoiceBox.setItems(leaguesList);
      this.stadiumsChoiceBox.setItems(stadiumsList);
    } catch (final SQLException exception) {
      Logger.getLogger(MainController.class.getName()).log(Level.SEVERE, null, exception);
    } catch (final Exception exception) {
      Logger.getLogger(MainController.class.getName()).log(Level.SEVERE, null, exception);
    }
  }

  @FXML
  private final void update() {
    try {
      Match match = this.matchesTableView.getSelectionModel().getSelectedItem();

      if (match == null) {
        return;
      }

      FXMLLoader loader = new FXMLLoader(getClass().getResource("update.fxml"));
      Parent parent = loader.load();

      UpdateController updateController = loader.getController();
      updateController.setUpdatingId(match.getId());

      Stage stage = new Stage();
      stage.setScene(new Scene(parent));
      stage.initStyle(StageStyle.UTILITY);
      stage.show();
    } catch (final IOException exception) {
      Logger.getLogger(MainController.class.getName()).log(Level.SEVERE, null, exception);
    } catch (final Exception exception) {
      Logger.getLogger(MainController.class.getName()).log(Level.SEVERE, null, exception);
    }
  }

  @FXML
  private final void delete() {
    Match match = this.matchesTableView.getSelectionModel().getSelectedItem();

    if (match == null) {
      return;
    }

    this.db.deleteByID(DB.MATCHES_TABLE, match.getId());
  }

  @FXML
  private final void search() {
    this.leagueFilter = this.leaguesChoiceBox.getValue();
    this.stadiumFilter = this.stadiumsChoiceBox.getValue();
    this.read();
  }

  private final void clearTextFields() {
    this.leagueNameTextField.clear();
    this.leagueCountryTextField.clear();
```

```
        this.stadiumNameTextField.clear();
        this.stadiumCountryTextField.clear();
        this.team1DescriptionTextArea.clear();
        this.team2DescriptionTextArea.clear();
    }
}
```

Match.java

```
package com.example;

import java.sql.Date;

public final class Match {
    private Integer id;
    private String league;
    private String stadium;
    private Date date;
    private String team1;
    private String team2;

    public Match(final Integer id, final String league, final String stadium, final Date date,
            final String team1, final String team2) {
        this.id = id;
        this.league = league;
        this.stadium = stadium;
        this.date = date;
        this.team1 = team1;
        this.team2 = team2;
    }

    public final Integer getId() {
        return this.id;
    }

    public final void setId(final Integer id) {
        this.id = id;
    }

    public final String getLeague() {
        return this.league;
    }

    public final void setLeague(final String league) {
        this.league = league;
    }

    public final String getStadium() {
        return this.stadium;
    }

    public final void setStadium(final String stadium) {
        this.stadium = stadium;
    }
```

```
    public final Date getDate() {
        return this.date;
    }

    public final void setDate(final Date date) {
        this.date = date;
    }

    public final String getTeam1() {
        return this.team1;
    }

    public final void setTeam1(final String team1) {
        this.team1 = team1;
    }

    public final String getTeam2() {
        return this.team2;
    }

    public final void setTeam2(final String team2) {
        this.team2 = team2;
    }
}
```

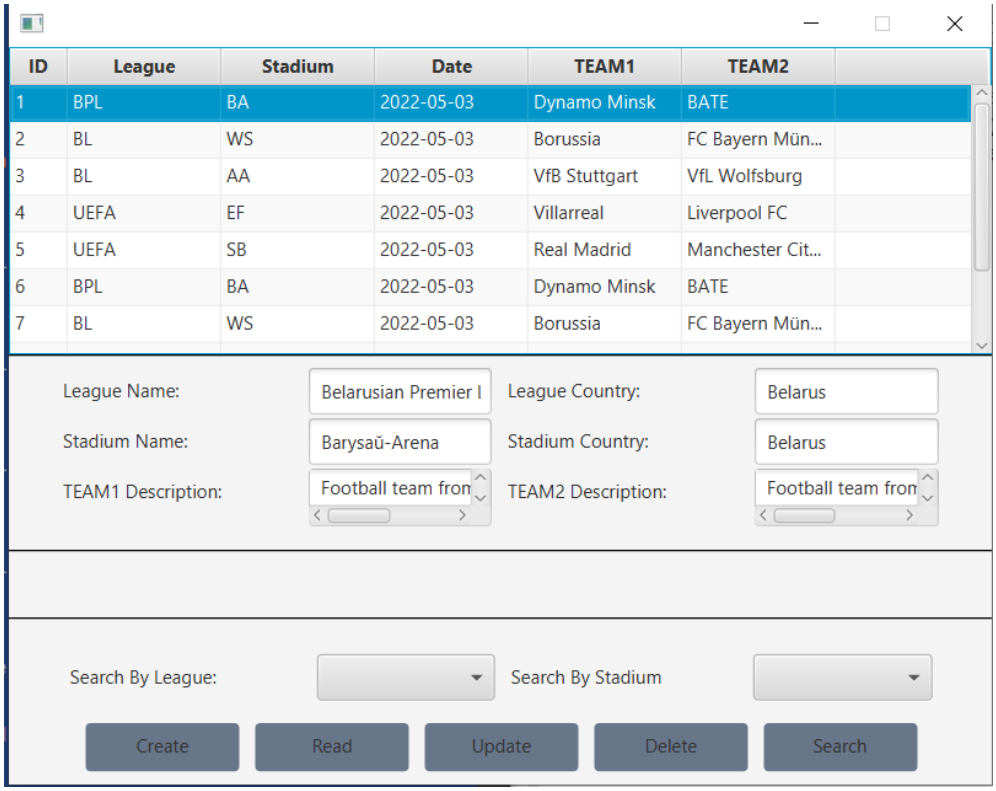
UpdateController.java

```
package com.example;

import java.net.URL;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ResourceBundle;
import java.util.logging.Level;
import java.util.logging.Logger;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.ChoiceBox;

public final class UpdateController implements Initializable {
    @FXML
    private ChoiceBox<String> leaguesChoiceBox;
    @FXML
    private ChoiceBox<String> stadiumsChoiceBox;
    @FXML
    private ChoiceBox<String> team1ChoiceBox;
    @FXML
    private ChoiceBox<String> team2ChoiceBox;

    private Integer updatingId = null;
```

```
private DB db = null;

@Override
public void initialize(URL arg0, ResourceBundle arg1) {
  try {
    this.db = new DB();

    ObservableList<String> leaguesList = FXCollections.observableArrayList();
    ObservableList<String> stadiumsList = FXCollections.observableArrayList();
    ObservableList<String> team1List = FXCollections.observableArrayList();
    ObservableList<String> team2List = FXCollections.observableArrayList();

    ResultSet leaguesSet = this.db.getAll(DB.LEAGUES_TABLE);
    ResultSet stadiumsSet = this.db.getAll(DB.STADIUMS_TABLE);
    ResultSet team1Set = this.db.getAll(DB.TEAM1_TABLE);
    ResultSet team2Set = this.db.getAll(DB.TEAM2_TABLE);

    while (leaguesSet.next() && stadiumsSet.next() && team1Set.next() && team2Set.next()) {
      leaguesList.add(leaguesSet.getString("short_name"));
      stadiumsList.add(stadiumsSet.getString("short_name"));
      team1List.add(team1Set.getString("name"));
      team2List.add(team2Set.getString("name"));
    }

    this.leaguesChoiceBox.setItems(leaguesList);
    this.stadiumsChoiceBox.setItems(stadiumsList);
    this.team1ChoiceBox.setItems(team1List);
    this.team2ChoiceBox.setItems(team2List);
  } catch (final SQLException exception) {
    Logger.getLogger(CreateController.class.getName()).log(Level.SEVERE, null, exception);
  } catch (final Exception exception) {
    Logger.getLogger(CreateController.class.getName()).log(Level.SEVERE, null, exception);
  }
}

public final void setUpdatingId(final Integer updatingId) {
  try {
    this.updatingId = updatingId;

    ResultSet match = this.db.getMatches(null, null, this.updatingId);
    match.next();

    this.leaguesChoiceBox.setValue(match.getString("league"));
    this.stadiumsChoiceBox.setValue(match.getString("stadium"));
    this.team1ChoiceBox.setValue(match.getString("team1"));
    this.team2ChoiceBox.setValue(match.getString("team2"));
  } catch (final SQLException exception) {
    Logger.getLogger(CreateController.class.getName()).log(Level.SEVERE, null, exception);
  } catch (final Exception exception) {
    Logger.getLogger(CreateController.class.getName()).log(Level.SEVERE, null, exception);
  }
}

@FXML
private final void update() {
  if (this.leaguesChoiceBox.getSelectionModel().isEmpty()
```

```
            || this.stadiumsChoiceBox.getSelectionModel().isEmpty()
            || this.team1ChoiceBox.getSelectionModel().isEmpty()
            || this.team2ChoiceBox.getSelectionModel().isEmpty()
            || this.updatingId == null) {
        return;
    }

    this.db.addMatch(new Match(this.updatingId,
            this.leaguesChoiceBox.getValue(),
            this.stadiumsChoiceBox.getValue(),
            null,
            this.team1ChoiceBox.getValue(),
            this.team2ChoiceBox.getValue()));
  }
}
```

Результат:



Вывод: приобрести практические навыки разработки многооконных приложений на JavaFX для работы с базами данных