Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

«Нижегородский государственный архитектурно-строительный университет»

Кафедра прикладной информатики и статистики

Отчет по практике

**Проектирование и разработка интерфейса к базе данных «Спортклуб»**

|  |  |  |
| --- | --- | --- |
| Выполнил студент: |  | Шалявин Д.А., 2 курс |
|  |  | заочное отделение, группа ПРИ-19сз |
| Научный руководитель | оценка, подпись | Старший преподаватель, Родионова С.В. |

Нижний Новгород

2021

**Содержание**

Введение…………………………………………….…………………………..3

Глава 1. Предметная область.….………...………………………………….…4

Глава 2. Разработка приложения………………………………………………7

2.1. Определение требований к приложению.…....……….………….………7

2.2. Выбор средств разработки..…....……….………….……………………8

2.3. Разработка эскизов дизайна.…....……….………….……………………..9

2.4. Программирование.…....……….………….……………………………..11

2.4.1. Создание окна авторизации……………………………………………11

2.4.2. Создание основного окна приложения………………………………..15

2.4.3. Создание модели……………………………………………………….16

2.4.4. Создание Data Access Object (DAO)…………………………………..18

2.4.5. Создание функций добавления информации………………………....21

2.4.6. Создание функций изменения и удаления информации…………….28

2.4.7. Создание функции поиска……………………………………………..30

Заключение…………………………………………………………………….31

Список используемой литературы…………………………..……………….32

Приложение ...…...…………………………………..…………..……………33

**Введение**

Несмотря на то что язык структурированных запросов SQL был создан для того чтобы любой пользователь не обладающий навыками программирования мог работать с базами данных, всё равно остаётся достаточно сложным для освоения. А со временем он обогатился новыми конструкциями, что сделало его ещё сложнее. Ещё одним недостатком SQL является, то что для того чтобы получить какую-либо информацию из базы данных нужно потратить время на составление этого запроса. Также системы управления базами данных часто имеют интерфейс на английском языке.

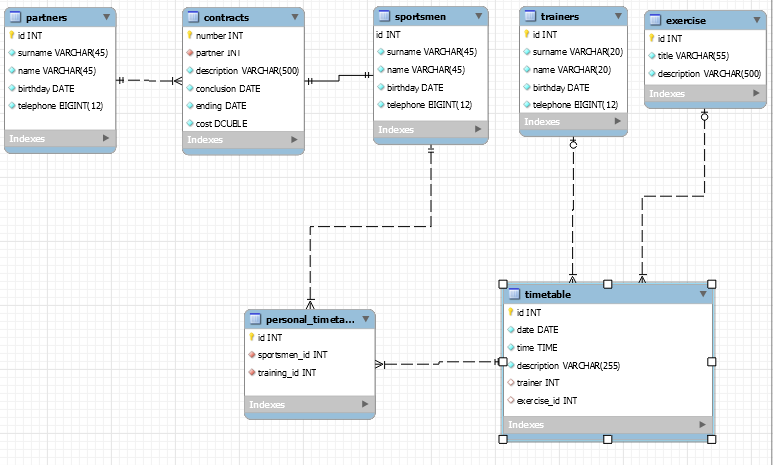
Целью работы является создание приложения для взаимодействия с базой данных.

Задачи работы:

1. Изучение основных этапов разработки приложения;
2. Углубление теоретических знаний и улучшение практических навыков в разработке приложений.
3. Формирование умения применять теоретические знания при решении поставленных профессиональных задач.

**Глава 1. Предметная область.**

Разрабатываемое приложение будет предназначено для работы базой данных «Спортклуб». Данная база данных была спроектирована в СУБД MySQL и имеет следующую структуру.



База данных имеет следующие сущности:

1. Контрагент (идентификационный номер, Ф.И.О., дата рождения, контактные данные) – лицо заключившее договор с клубом (необходим, поскольку договор может быть заключен только с лицом, достигшим 18 лет);

2. Договор (номер договора, идентификационный номер контрагента, описание, дата начала, дата окончания, стоимость);

3. Член клуба (идентификационный номер, Ф.И.О., дата рождения, контактные данные) – это непосредственно человек, который будет заниматься спортом;

4. Тренер (идентификационный номер, Ф.И.О., дата рождения, контактные данные) – руководитель спортивных занятий;

5. Занятия (идентификационный номер, название занятия, описание) – виды занятий проводимых в клубе;

6. Расписание (идентификационный номер занятия, дата и время проведения занятия, идентификационный номер тренера, описание);

7. Личное расписание члена клуба (идентификационный номер записи, номер спортсмена, идентификационный номер тренировки из расписания).

На указанной выше схеме «Контрагенты» обозначены таблицей «partners», «Договоры» – «contracts», «Члены клуба» – «sportsmens», «Тренеры» – «trainers», «Занятия» – «exercise», «Расписание» – «timetable», «Личное расписание» – «personal\_timetable».

Таблица «partners» - на стадии предварительного проектирования базы данных она была выделена из таблицы «contracts» для не загромождения её и с целью нормализации базы. Она предназначена для хранения сведений о контрагентах.

Таблица «contracts» - предназначена для хранения сведений о заключенных договорах, имеет первичный ключ – номер договора (number) и внешний ключ id контрагента.

Таблица «sportsmens» - предназначена для хранения сведений о членах клуба, имеет внешний ключ id (номер договора) который является и первичным ключом данной таблицы.

Таблица «trainers» - предназначена для хранения сведений о тренерах, имеет первичный ключ id, каждый представленный в базе тренер имеет обширную специализацию и может проводить любое занятие, проводимое в клубе, поэтому не имеет связи с таблицей «exercise».

Таблица «exercise» - предназначена для хранения сведений о занятиях, проводимых в клубе, имеет первичный ключ id. Содержит описание занятия.

Таблица «timetable» - предназначена для представления расписания занятий в клубе, первичным ключом является id, имеет два внешних ключа id тренера и id занятия, также содержит краткое описание предстоящей тренировки, особенности её проведения.

Таблица «personal\_timetable» - служит для записи сведений о посещенных тренировках членами клуба имеет первичный ключ id записи и два внешних ключа id спортсмена и id тренировки из расписания клуба.

**Глава 2. Разработка приложения.**

**2.1. Определение требований к приложению.**

Целевой аудиторией приложения будут работники спортклуба, занимающиеся учетом клиентов, составлением расписания и записью на занятия.

Приложение должно иметь основные функции работы с базами данных, а именно: добавление, удаление и изменение существующей информации.

Теперь перейдём к формированию требований. Начнем с функциональных:

система должна принимать: персональные данные клиентов спортклуба (контрагентов, спортсменов), тренеров клуба, данные о проводимых занятиях и заносить их в базу данных;

система должна проводить проверку вводимых данных на корректность (например, имя не может иметь цифр);

система должна предоставлять возможность поиска клиента в базе данных по фамилии;

система должна предоставлять для сотрудников компании-заказчика возможность просмотра, редактирования и удаления информации в базе данных;

система должна быть работоспособна на устройствах под управлением операционной системы Windows 7 и выше.

Из не функциональных требований отметим:

система должна предлагать каждому вошедшему пользователю авторизоваться или зарегистрироваться в системе;

система должна записывать произведенные пользователем действия над базой данных в файл (сохранять логи).

**2.2. Выбор средств разработки.**

Для разработки приложения я буду использовать язык программирования Java. Это строго типизированный объектно-ориентированный язык программирования общего назначения. Для создания графического представления я буду использовать JavaFX (платформа на основе Java для создания приложений с насыщенным графическим интерфейсом) и приложение для упрощения создания графической части и дизайна SceneBuilder. Также из программных средств я буду использовать среду разработки IntelliJ IDEA.

Для создания приложения я буду использовать концепцию Model-View-Controller. Model-View-Controller (MVC, «Модель-Представление-Контроллер», «Модель-Вид-Контроллер») – схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер – таким образом, что модификация каждого компонента может осуществляться независимо.

**2.3. Разработка эскизов дизайна.**

В качестве основных цветов приложения я возьму следующий набор:

код цвета - #307672

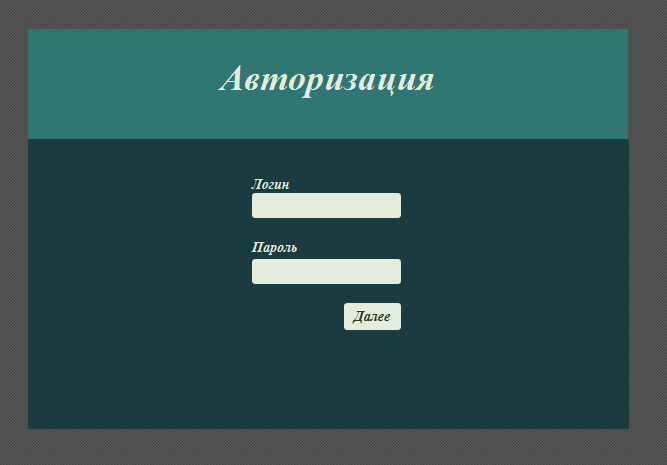
код цвета - #144d53

Для текста будут использованы черный и белый цвета шрифт Times New Roman.

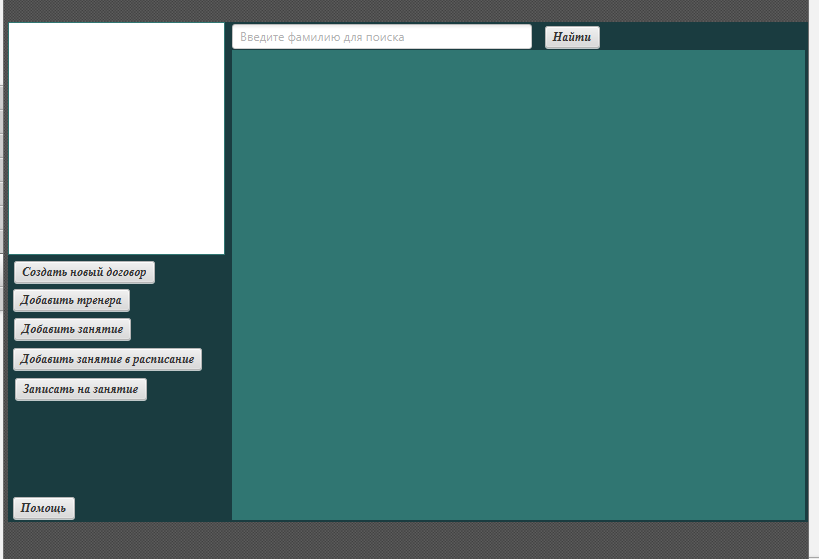
Приступим к созданию эскиза дизайна страницы авторизации.

Страница авторизации должна иметь поле для ввода имени пользователя (логина) и поле для ввода пароля. Должна быть кнопка подтверждения ввода.

Теперь переведем текст в графику с помощью SceneBuilder.



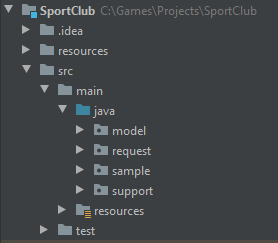
Теперь приступим к созданию дизайна основной страницы приложения. Она должна содержать панель навигации по базе данных, панель поиска, поле для вывода информации из базы данных и набор кнопок для начала поиска, создания новых договоров, внесения изменения в расписание и др. Получим следующее представление:



Также потребуется создать отдельные страницы для: создания нового договора, добавления нового тренера, добавления нового занятия, изменения расписания и записи на занятие. Они будут выполнены в том же стиле, и иметь набор полей по количеству параметров сущностей содержащихся в базе данных и кнопок подтверждения и отмены изменений.

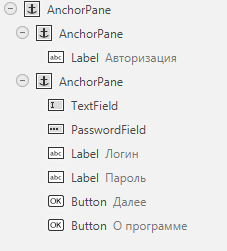
**2.4. Программирование.**

Приступим к написанию программного кода приложения. Начнем с создания предварительной структуры проекта. Создадим несколько пакетов (папок). Папка **model** будет содержать классы, представляющие сущности содержащиеся в базе данных, папка **request** будет содержать Data Access Object – это прослойка между БД и системой, папка **sample** будет содержать классы контроллеры, в папке **support** будут находиться классы поддержки, в папке **resources** будут содержаться файлы представления (View).



**2.4.1. Создание окна авторизации.**

Дизайн окна авторизации уже разработан, SceneBuilder автоматически переводит собранную страницу в формат fxml, остается только сохранить страницу, в папке с ресурсами проекта. Это будет файл sample.fxml. Окно авторизации будет иметь следующую структуру.



Также в SceneBuilder сразу присвоим id для элементов страницы, к которым будем обращаться из кода.

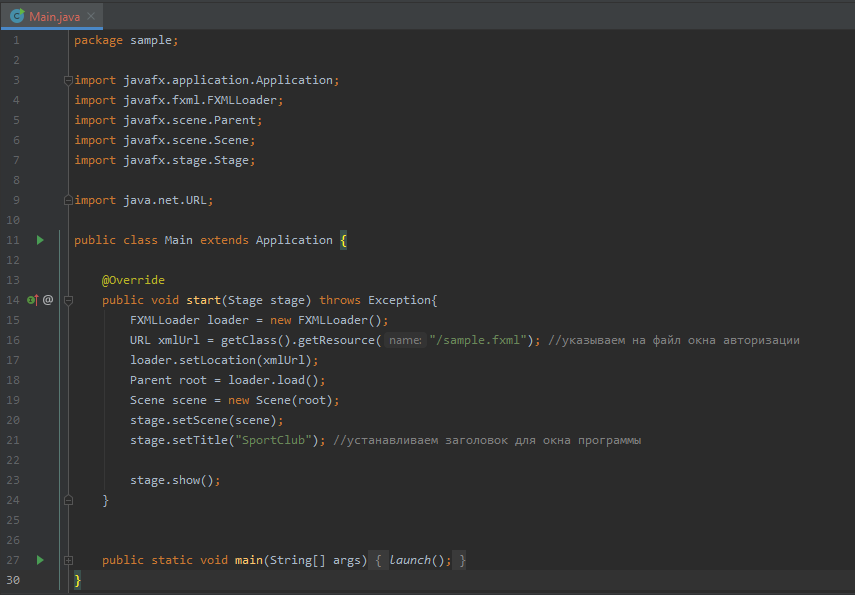
Для текстового поля «логин» присвоим id – inputLogin.

Для текстового поля «пароль» присвоим id – inputPassword.

Для кнопки «далее» присвоим id – nextBtn.

Для кнопки «о программе» присвоим id – helpBtn.

Теперь перейдем к созданию класса запуска приложения, это будет класс Main. После запуска этого класса будет открываться окно авторизации. При создании проекта JavaFX среда разработки автоматически создает класс Main, остается только внести в него небольшие изменения (указаны комментариями на рисунке ниже).

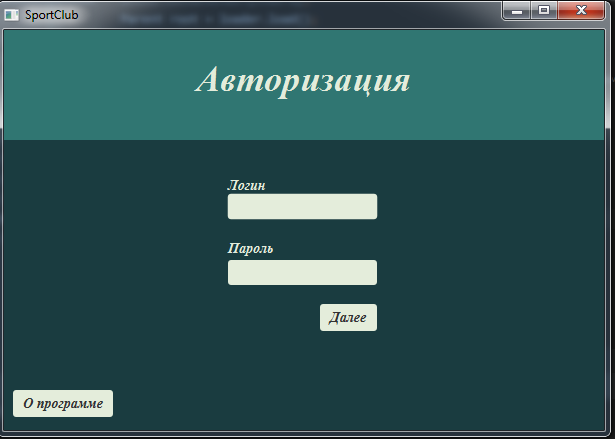


Для работы с окном авторизации потребуется создать класс контроллер. SceneBuilder предлагает сформировать класс контроллера за нас, этим и воспользуемся. Получим следующий код.



Остается только создать файл Controller.java в пакете sample и поместить этот код туда. Для работы с элементами страницы потребуется дополнительно создать метод initialize(). Он будет считывать данные введенные пользователем в поля логин и пароль, затем устанавливать соединение с базой данных, а в случае ввода неверных данных выводить всплывающее информационное окно. Если соединение с БД успешно установлено будет открываться основное окно программы. В дальнейшем дополнительно будет добавлено логгирование информации о пользователе, вошедшем в систему.

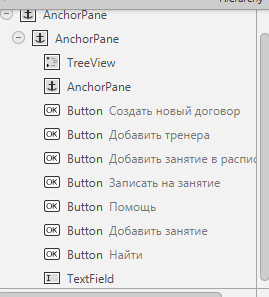
Попробуем запустить приложение. После запуска увидим окно авторизации.



**2.4.2. Создание основного окна приложения.**

Для создания основной страницы приложения необходимо выполнить те же действия, как и для окна авторизации: создать файл представления → создать класс контроллер → создать в контроллере метод инициализации. После этих действий структура проекта пополнится двумя файлами: mainScene.fxml и MainSceneController.java.

Окно будет иметь следующую иерархию элементов:



Также как и для окна авторизации, я присвою id для элементов страницы.

Метод инициализации этого класса-контроллера будет наполнять элементами компонент TreeView, присваивать обработчики событий для кнопок, считывать информацию из строки поиска и др.

Для того чтобы наполнить данную страницу этим функционалом необходимо сначала создать модель и создать классы которые будут делать запросы к базе данных.

**2.4.3. Создание модели.**

Как было сказано выше база данных имеет следующие сущности:

1. Контрагент; (Partner)

2. Договор;(Contract)

3. Член клуба; (Sportsman)

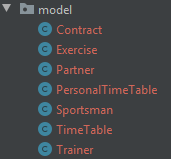
4. Тренер; (Trainer)

5. Занятия; (Exercise)

6. Расписание; (TimeTable)

7. Личное расписание члена клуба. (PersonalTimeTable)

Для работы с этими сущностями нам потребуется создать 7 классов моделей в пакете model:



Каждый из этих классов будет иметь такие же поля как в базе данных. Приведу код класса Sportsman для примера.

package model;  
  
import java.math.BigDecimal;  
import java.sql.Date;  
  
public class Sportsman {  
 private int id;  
 private String surname;  
 private String name;  
 private Date birthday;  
 private BigDecimal telephone;  
  
 //конструктор класса  
 public Sportsman(int id, String surname, String name, Date birthday, BigDecimal telephone) {  
 this.id = id;  
 this.surname = surname;  
 this.name = name;  
 this.birthday = birthday;  
 this.telephone = telephone;  
 }  
  
 //методы получения и редактирования полей объекта  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public String getSurname() {  
 return surname;  
 }  
  
 public void setSurname(String surname) {  
 this.surname = surname;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public Date getBirthday() {  
 return birthday;  
 }  
  
 public void setBirthday(Date birthday) {  
 this.birthday = birthday;  
 }  
  
 public BigDecimal getTelephone() {  
 return telephone;  
 }  
  
 public void setTelephone(BigDecimal telephone) {  
 this.telephone = telephone;  
 }  
}

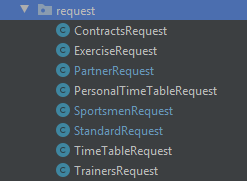
Остальные классы сущностей будут иметь схожую структуру.

**2.4.4. Создание Data Access Object** **(DAO).**

Как уже было сказано выше этот набор классов будет создаваться в пакете request и их задачей будет получение информации из базы данных.

Потребуется создать 7 классов по одному для каждой модели и 1 класс дополнительно для удобства работы с DAO из контроллера. Дополнительный класс (StandardRequest) будет собирать данные из остальных классов, и предоставлять их по запросу контроллеру.

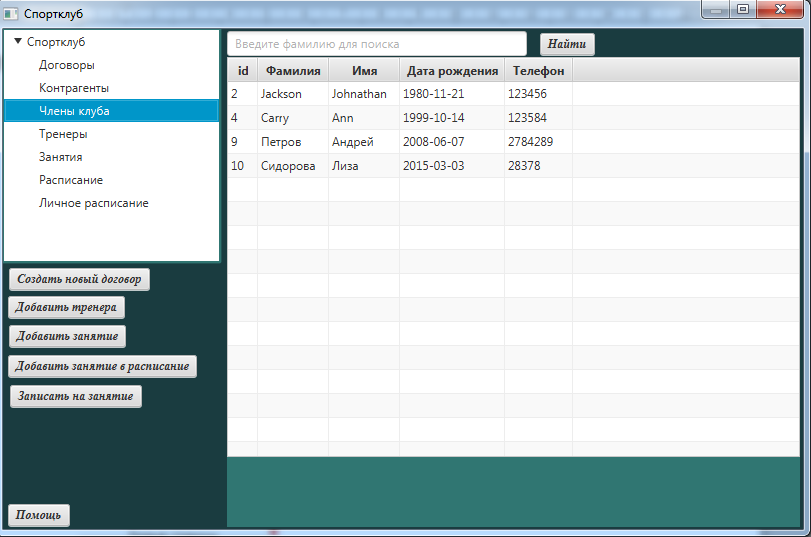
Получим следующую структуру пакета request.



Также как и модели, все эти классы будут иметь схожее строение (за исключением класса StandardRequest), и для примера приведу код класса PartnerRequest.

package request;  
  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.event.EventHandler;  
import javafx.scene.control.TableColumn;  
import javafx.scene.control.TableView;  
import javafx.scene.control.cell.PropertyValueFactory;  
import javafx.scene.input.MouseEvent;  
import model.Partner;  
import sample.Controller;  
import sample.edit\_controller.EditPartnersController;  
import support.MyContextMenu;  
  
import java.sql.Connection;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.sql.Date;  
  
public class PartnerRequest {  
 //получаем соединение с базой данных от класса контроллера окна авторизации  
 private static Connection *connection* = Controller.*getConnection*();  
 //инициализация списка контрагентов  
 private ObservableList<Partner> partners = FXCollections.*observableArrayList*();  
 //получение данных из БД и заполнение списка  
 private ObservableList<Partner> getPartners(){  
 try {  
 PreparedStatement preparedStatement = *connection*.prepareStatement("SELECT \* FROM partners ORDER BY id");  
  
 ResultSet resultSet = preparedStatement.executeQuery();  
  
 while (resultSet.next()){  
 partners.add(new Partner(  
 resultSet.getInt("id"),  
 resultSet.getString("surname"),  
 resultSet.getString("name"),  
 resultSet.getDate("birthday"),  
 resultSet.getBigDecimal("telephone")  
 ));  
  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return partners;  
 }  
 //создание табличного представления из списка  
 TableView<Partner> createTBPartners(){  
 TableView<Partner> tableView = new TableView<>(getPartners());  
  
  
 TableColumn<Partner, Integer> id = new TableColumn<>("id");  
 id.setCellValueFactory(new PropertyValueFactory<>("id"));  
 tableView.getColumns().add(id);  
  
 TableColumn<Partner, String> surname = new TableColumn<>("Фамилия");  
 surname.setCellValueFactory(new PropertyValueFactory<>("surname"));  
 tableView.getColumns().add(surname);  
  
 TableColumn<Partner, String> name = new TableColumn<>("Имя");  
 name.setCellValueFactory(new PropertyValueFactory<>("name"));  
 tableView.getColumns().add(name);  
  
 TableColumn<Partner, Date> birthday = new TableColumn<>("Дата рождения");  
 birthday.setCellValueFactory(new PropertyValueFactory<>("birthday"));  
 tableView.getColumns().add(birthday);  
  
 TableColumn<Partner, Integer> telephone = new TableColumn<>("Телефон");  
 telephone.setCellValueFactory(new PropertyValueFactory<>("telephone"));  
 tableView.getColumns().add(telephone);  
  
 tableView.setPrefWidth(573);  
  
 tableView.setOnMouseClicked(new EventHandler<MouseEvent>() {  
 @Override  
 public void handle(MouseEvent mouseEvent) {  
 if (tableView.getSelectionModel().getSelectedItem() != null) {  
 tableView.setContextMenu(MyContextMenu.*partnerContext*());  
 int lineNumber = tableView.getSelectionModel().getSelectedItem().getId();  
 EditPartnersController.*setContragentID*(lineNumber);  
 }  
 }  
 });  
 return tableView;  
 }  
}

Теперь можно проверить работу приложения. Откроем основную страницу программы. И выберем пункт меню «Члены клуба».



Наше приложение пока не умеет добавлять данные в БД поэтому таблицы были заполнены через MySQL Workbench.

**2.4.5. Создание функций добавления информации.**

Для создания функций добавления потребуется создать 5 представлений и соответственно 5 классов-контроллеров:

1. Для создания нового договора (здесь будут добавляться: договор, контрагент и спортсмен);

2. Для добавления тренера;

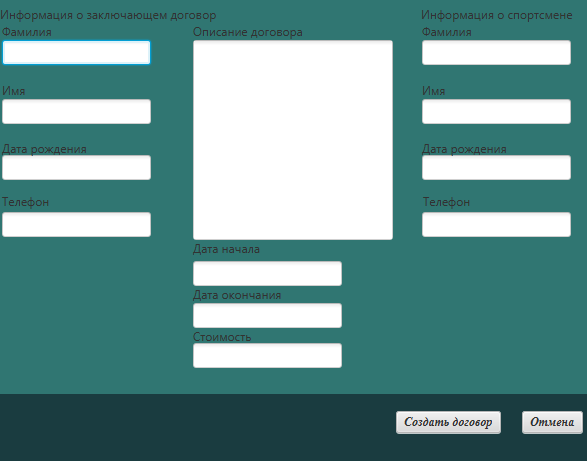
3. Для добавления занятия в список занятий;

4. Для добавления занятия в расписание занятий;

5. Для записи на занятие.

Для создания представлений и «скелета» контроллеров будем использовать SceneBuilder. Начнем с представления для создания нового договора. В этом представлении нам потребуется запросить данные контрагента, данные о договоре и данные, о спортсмене, затем если они корректны, занести эти данные в базу данных.

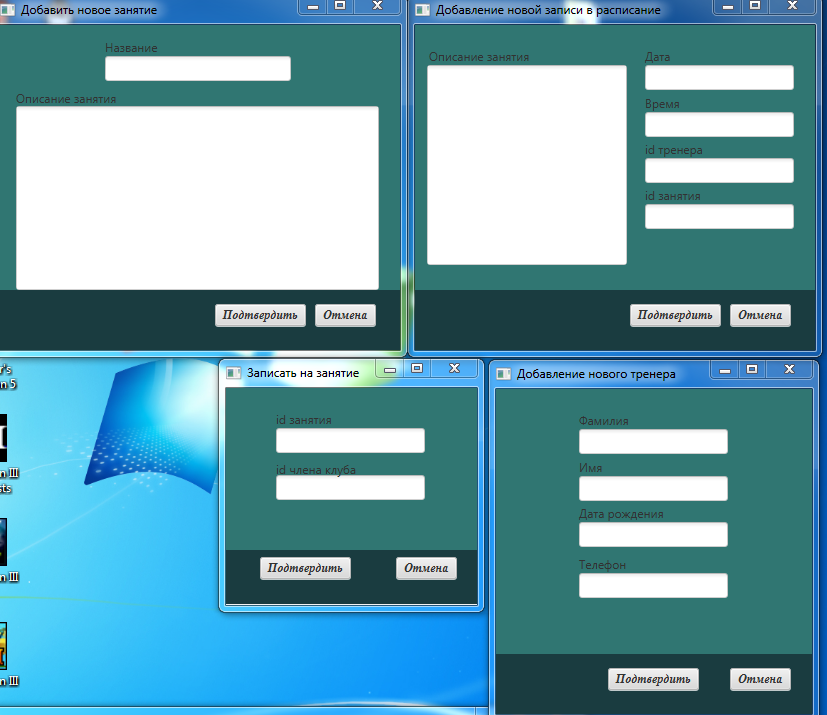
Приступим к разработке дизайна этого представления. Для каждого из полей сущностей нужно создать текстовое поле для ввода информации, за исключением поля id, т.к. оно заполняется автоматически при добавлении данных в базу.



Теперь нужно создать класс контроллера, для удобства контроллеры, отвечающие за добавление информации, выделим в отдельный пакет add\_controller в составе пакета sample. Назовем первый контроллер NewContractController.java. Приведу полный код этого класса:

package sample.add\_controller;  
  
import javafx.fxml.FXML;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.control.TextArea;  
import javafx.scene.control.TextField;  
import javafx.stage.Stage;  
import org.apache.log4j.Logger;  
import support.AlertWindow;  
import sample.Controller;  
import sample.MainSceneController;  
  
import java.io.IOException;  
import java.net.URL;  
import java.sql.Connection;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.util.ResourceBundle;  
  
public class NewContractController {  
 private static Connection *connection* = Controller.*getConnection*();  
 private static final Logger *logger* = Logger.*getLogger*(NewContractController.class);  
  
 @FXML // ResourceBundle that was given to the FXMLLoader  
 private ResourceBundle resources;  
  
 @FXML // URL location of the FXML file that was given to the FXMLLoader  
 private URL location;  
  
 @FXML // fx:id="contragentSurname"  
 private TextField contragentSurname; // Value injected by FXMLLoader  
  
 @FXML // fx:id="contragentName"  
 private TextField contragentName; // Value injected by FXMLLoader  
  
 @FXML // fx:id="contragentBirthday"  
 private TextField contragentBirthday; // Value injected by FXMLLoader  
  
 @FXML // fx:id="contagentTelephone"  
 private TextField contagentTelephone; // Value injected by FXMLLoader  
  
 @FXML // fx:id="contractDescription"  
 private TextArea contractDescription; // Value injected by FXMLLoader  
  
 @FXML // fx:id="contractConclusion"  
 private TextField contractConclusion; // Value injected by FXMLLoader  
  
 @FXML // fx:id="contractEnding"  
 private TextField contractEnding; // Value injected by FXMLLoader  
  
 @FXML // fx:id="contractCost"  
 private TextField contractCost; // Value injected by FXMLLoader  
  
 @FXML // fx:id="sportsmanSurname"  
 private TextField sportsmanSurname; // Value injected by FXMLLoader  
  
 @FXML // fx:id="sportsmanName"  
 private TextField sportsmanName; // Value injected by FXMLLoader  
  
 @FXML // fx:id="sportsmanBirthday"  
 private TextField sportsmanBirthday; // Value injected by FXMLLoader  
  
 @FXML // fx:id="sportsmanTelephone"  
 private TextField sportsmanTelephone; // Value injected by FXMLLoader  
  
 @FXML // fx:id="createBtn"  
 private Button createBtn; // Value injected by FXMLLoader  
  
 @FXML // fx:id="cancelBtn"  
 private Button cancelBtn; // Value injected by FXMLLoader  
  
 @FXML // This method is called by the FXMLLoader when initialization is complete  
 void initialize() {  
 createBtn.setOnAction(e -> {  
 createContract();  
 });  
 cancelBtn.setOnAction(e -> {  
 returnToPrevStage();  
 });  
 }  
  
 //добавляем новый договор  
 private void createContract(){  
 //проверяем ввод, если все нормально, то продолжаем  
 if(checkingInput()) {  
 Stage stage = (Stage) createBtn.getScene().getWindow();  
 try {  
 PreparedStatement preparedStatement;  
 //создаем первый запрос для добавления сведений о контрагенте  
 preparedStatement = *connection*.prepareStatement(  
 "INSERT INTO partners(surname, name, birthday, telephone) VALUES (?, ?, ?, ?)");  
  
 preparedStatement.setString(1, contragentSurname.getText().trim());  
 preparedStatement.setString(2, contragentName.getText().trim());  
 preparedStatement.setString(3, contragentBirthday.getText().trim());  
 preparedStatement.setString(4, contagentTelephone.getText().trim());  
  
 preparedStatement.executeUpdate();  
  
 //второй запрос для добавления сведения о договоре  
 //предварительно получаем номер последней записи в таблице контрагенты  
 preparedStatement = *connection*.prepareStatement("SELECT MAX(id) as id FROM partners");  
 ResultSet rs = preparedStatement.executeQuery();  
 rs.next();  
 int number = rs.getInt("id");  
 rs.close();  
  
 preparedStatement = *connection*.prepareStatement(  
 "INSERT INTO contracts(partner, description, conclusion, ending, cost) VALUES (?, ?, ?, ?, ?)");  
  
 preparedStatement.setString(1, String.*valueOf*(number));  
 preparedStatement.setString(2, contractDescription.getText().trim());  
 preparedStatement.setString(3, contractConclusion.getText().trim());  
 preparedStatement.setString(4, contractEnding.getText().trim());  
 preparedStatement.setString(5, contractCost.getText().trim());  
  
 preparedStatement.executeUpdate();  
  
 //третий запрос - для добавления записей в таблицу Спортсмены  
 //предварительно получаем сведения о последнем номере договора - он будет id спортсмена  
 preparedStatement = *connection*.prepareStatement("SELECT MAX(number) as number FROM contracts");  
 rs = preparedStatement.executeQuery();  
 rs.next();  
 number = rs.getInt("number");  
 rs.close();  
  
 preparedStatement = *connection*.prepareStatement(  
 "INSERT INTO sportsmen(id, surname, name, birthday, telephone) VALUES (?, ?, ?, ?, ?)");  
  
 preparedStatement.setString(1, String.*valueOf*(number));  
 preparedStatement.setString(2, sportsmanSurname.getText().trim());  
 preparedStatement.setString(3, sportsmanName.getText().trim());  
 preparedStatement.setString(4, sportsmanBirthday.getText().trim());  
 preparedStatement.setString(5, sportsmanTelephone.getText().trim());  
  
 preparedStatement.executeUpdate();  
  
 preparedStatement.close();  
  
 AlertWindow.*showAlertWithoutHeaderText*("Данные успешно добавлены!");  
  
 *logger*.info("Добавлен новый договор. Спортсмен: " + sportsmanSurname.getText().trim() + " " + sportsmanName.getText().trim());  
 MainSceneController.*standardRequest*.refreshAll();  
 //возвращаемся на предыдущую страницу  
 returnToPrevStage();  
 } catch (SQLException e) {  
 *logger*.error(e.getMessage());  
 AlertWindow.*showAlertWithoutHeaderText*("Если вы видете это сообщение, то обратитесь к администратору!");  
 }  
 }  
 }  
  
 //вовращает на предыдущую страницу  
 private void returnToPrevStage(){  
 Stage stage = (Stage) createBtn.getScene().getWindow();  
 stage.close();  
 FXMLLoader fxmlLoader = new FXMLLoader(getClass().getResource("/mainScene.fxml"));  
 Parent root1;  
 try {  
 root1 = fxmlLoader.load();  
 stage.setScene(new Scene(root1));  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 //проверяем вводимые данные на соответствие шаблонам  
 //и выводим сообщение в случае ошибки во введенных данных  
 public boolean checkingInput(){  
 //проверка данных контрагента  
 String partnerSurname = contragentSurname.getText().trim();  
 if(!partnerSurname.matches("^\\D+\\b")){  
 AlertWindow.*showAlertWithoutHeaderText*(  
 "Проверьте поле 'Фамилия' контрагента!\nПоле не должно содержать цифр!");  
 return false;  
 }  
 String partnerName = contragentName.getText();  
 if(!partnerName.matches("^\\D+\\b")){  
 AlertWindow.*showAlertWithoutHeaderText*(  
 "Проверьте поле 'Имя' контрагента!\nПоле не должно содержать цифр!");  
 return false;  
 }  
 String partnerBirthday = contragentBirthday.getText();  
 if(!partnerBirthday.matches("^\\d\\d\\d\\d-\\d\\d-\\d\\d\\b")){  
 AlertWindow.*showAlertWithoutHeaderText*(  
 "Проверьте поле 'Дата рождения' контрагента!\nДата должна быть в формате - гггг-мм-дд!");  
 return false;  
 }  
 String partnerTelephone = contagentTelephone.getText();  
 if(!partnerTelephone.matches("^\\d{11}$")){  
 AlertWindow.*showAlertWithoutHeaderText*(  
 "Проверьте поле 'Телефон' контрагента!\nНомер должен содержать 11 цифр и не может содержать спецсимволов!");  
 return false;  
 }  
  
 //проверка данных о договоре  
 String description = contractDescription.getText();  
 if(description.isEmpty()){  
 AlertWindow.*showAlertWithoutHeaderText*(  
 "Поле 'Описание' не должно быть пустым!");  
 return false;  
 }  
 String conclusion = contractConclusion.getText();  
 if(!conclusion.matches("^\\d\\d\\d\\d-\\d\\d-\\d\\d\\b")){  
 AlertWindow.*showAlertWithoutHeaderText*(  
 "Проверьте поле 'Дата начала'!\nДата должна быть в формате - гггг-мм-дд!");  
 return false;  
 }  
 String ending = contractEnding.getText();  
 if(!ending.matches("^\\d\\d\\d\\d-\\d\\d-\\d\\d\\b")){  
 AlertWindow.*showAlertWithoutHeaderText*(  
 "Проверьте поле 'Дата окончания'!\nДата должна быть в формате - гггг-мм-дд!");  
 return false;  
 }  
 String cost = contractCost.getText();  
 if(!cost.matches("^\\d+\\.?\\d\*$")){  
 AlertWindow.*showAlertWithoutHeaderText*(  
 "Проверьте поле 'Стоимость'!\nПоле не должно содержать букв и спецсимволов, кроме '.'!");  
 return false;  
 }  
  
 //проверка данных о спортсмене  
 String sportsmanLastName = sportsmanSurname.getText();  
 if(!sportsmanLastName.matches("^\\D+\\b")){  
 AlertWindow.*showAlertWithoutHeaderText*(  
 "Проверьте поле 'Фамилия' спортсмена!\nПоле не должно содержать цифр!");  
 return false;  
 }  
 String sportsmanFirstName = sportsmanName.getText();  
 if(!sportsmanFirstName.matches("^\\D+\\b")){  
 AlertWindow.*showAlertWithoutHeaderText*(  
 "Проверьте поле 'Имя' спортсмена!\nПоле не должно содержать цифр!");  
 return false;  
 }  
 String sportsmanBD = sportsmanBirthday.getText();  
 if(!sportsmanBD.matches("^\\d\\d\\d\\d-\\d\\d-\\d\\d\\b")){  
 AlertWindow.*showAlertWithoutHeaderText*(  
 "Проверьте поле 'Дата рождения' спортсмена!\nДата должна быть в формате - гггг-мм-дд!");  
 return false;  
 }  
 String sportsmanTel = sportsmanTelephone.getText();  
 if(!sportsmanTel.matches("^\\d{11}$")){  
 AlertWindow.*showAlertWithoutHeaderText*(  
 "Проверьте поле 'Телефон' спортсмена!\nНомер должен содержать 11 цифр и не может содержать спецсимволов!");  
 return false;  
 }  
 return true;  
 }  
}

Создание остальных представлений будет похоже на это, поэтому просто добавлю изображение этих представлений.



**2.4.6. Создание функций изменения и удаления информации.**

Для создания функции изменения потребуется создать представление для каждой сущности, для облегчения работы буду использовать представления, созданные для добавления информации. Классы-контроллеры также будут схожими, за исключением того что при вызове функции изменения поля формы будут заполняться текущими данными автоматически.

Чтобы не загромождать интерфейс приложения кнопками изменения, добавлю функцию изменения в контекстное меню при нажатии на запись в таблицах.

Для создания контекстного меню создам отдельный класс с именем MyContextMenu в пакете support. В этом классе будут содержаться статические методы, создающие контекстное меню для каждой таблицы и содержащие SQL запросы для изменения данных.

Также в контекстном меню будет доступна функция удаления записи из таблицы.

Приведу код для создания контекстного меню для таблицы «Договоры»:

public static ContextMenu partnerContext() {  
 ContextMenu contextMenu = new ContextMenu();  
 MenuItem info = new MenuItem("Дополнительно");  
 MenuItem change = new MenuItem("Изменить");  
 MenuItem delete = new MenuItem("Удалить");  
 contextMenu.getItems().addAll(change, delete, info);  
  
 change.setOnAction(e ->{  
 *fxmlLoader* = new FXMLLoader(MyContextMenu.class.getResource(  
 "/additionalScenes/edit\_scenes/editPartners.fxml"));  
 Parent root1;  
 try {  
 root1 = *fxmlLoader*.load();  
 Stage addStage = new Stage();  
 addStage.initModality(Modality.*WINDOW\_MODAL*);  
 addStage.setTitle("Изменение данных контрагента");  
 addStage.setScene(new Scene(root1));  
 addStage.show();  
 } catch (IOException ex) {  
 ex.printStackTrace();  
 }  
 });  
  
 delete.setOnAction(e ->{  
 Alert alert = new Alert(  
 Alert.AlertType.*CONFIRMATION*,  
 "Вместе с этой записью вы также удалите сведения о спортсмене и договоре.\n" +  
 "Вы действительно хотите удалить выбранную запись?",  
 ButtonType.*YES*, ButtonType.*NO*);  
 alert.setHeaderText(null);  
 alert.showAndWait();  
  
 if (alert.getResult() == ButtonType.*YES*) {  
 try {  
 PreparedStatement preparedStatement = *connection*.prepareStatement(  
 "delete from partners where id = ?;");  
  
 preparedStatement.setInt(1, EditPartnersController.*getContragentID*());  
  
 preparedStatement.executeUpdate();  
 MainSceneController.*standardRequest*.refreshAll();  
 } catch (SQLException exception) {  
 exception.printStackTrace();  
 }  
 }  
 });  
  
 return contextMenu;  
}

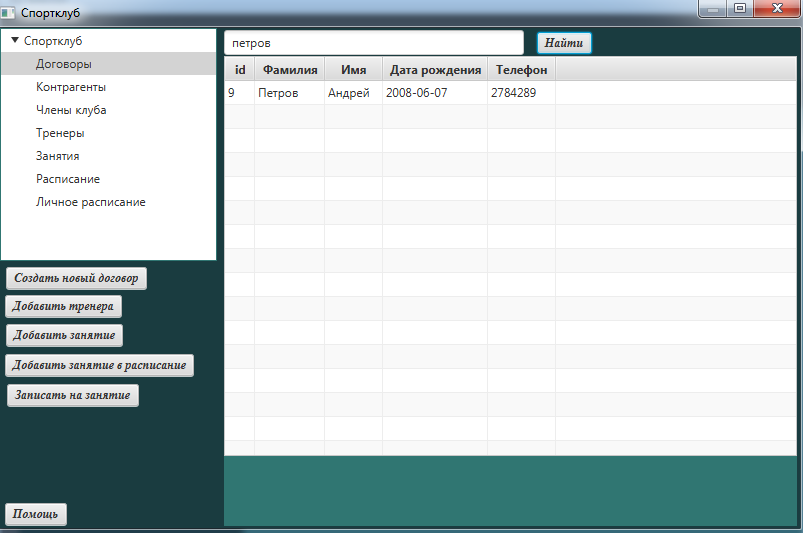
**2.4.7. Создание функции поиска.**

Для реализации этой функции потребуется в контроллере главной страницы получить текст из строки поиска, потом по нажатию кнопки «Найти» создать запрос к базе данных и выдать ответ в виде записи из таблицы. Так будет выглядеть код метода запроса к базе данных на примере поиска спортсмена по фамилии:

TableView<Sportsman> searchSportsman(String sportsmanSurname) throws SQLException {  
 PreparedStatement preparedStatement;  
  
 //запрос для получения сведений о спортсмене  
 preparedStatement = *connection*.prepareStatement(  
 "SELECT \* FROM sportsmen WHERE surname = ?");  
  
 preparedStatement.setString(1, sportsmanSurname);  
  
 ResultSet resultSet = preparedStatement.executeQuery();

Далее в этом методе идёт построение табличного представления.

Результат работы:



**Заключение**

Результатом данной работы стало создание приложения для работы с базой данных позволяющее создавать, изменять и удалять данные из базы. На мой взгляд, приложение является простым в освоении и облегчит работу с базой данных «Спортклуб». Скриншоты работы программы приведены в приложении к данной работе.

В ходе выполнения данной работы были изучены основные этапы разработки приложений. Углублены теоретические знания и улучшены практические навыки в разработке приложений. Сформировано умение применять теоретические знания при решении поставленных профессиональных задач.

Выполненная работа позволила углубить знания в области информатики и использования современных информационных систем.

**Список литературы**

1. Сайт о программировании Metanit.com – Режим доступа: https://metanit.com/, свободный – (21.06.2021);

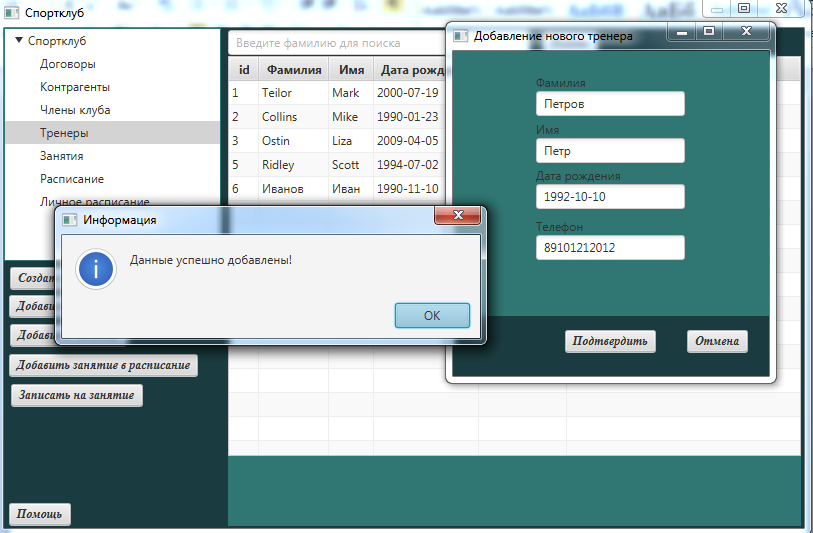
2. Сайт о программировании StackOverFlow – Режим доступа: https://stackoverflow.com/, свободный – (21.06.2021);

3. Шилдт, Герберт Java. Полное руководство, 10-е изд. : Пер. с англ. -СПб. ООО «Альфакнига»;2018. – 1488 с.

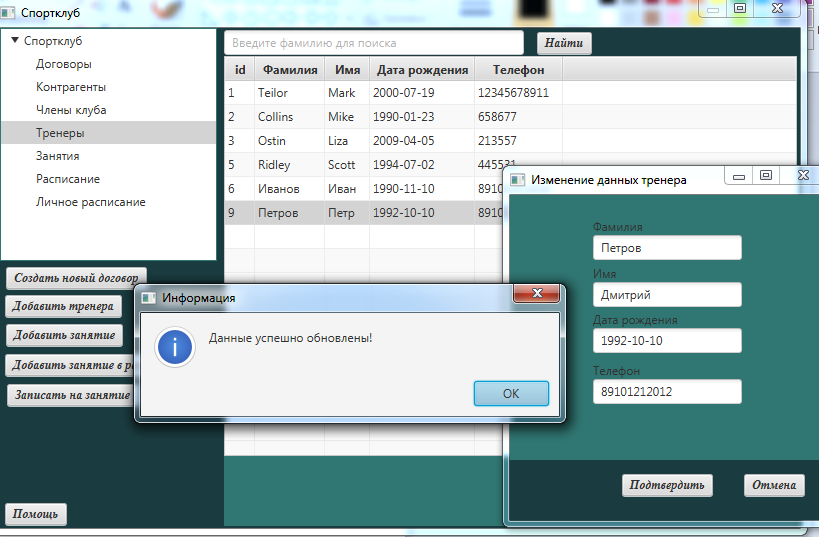
Приложение к проекту

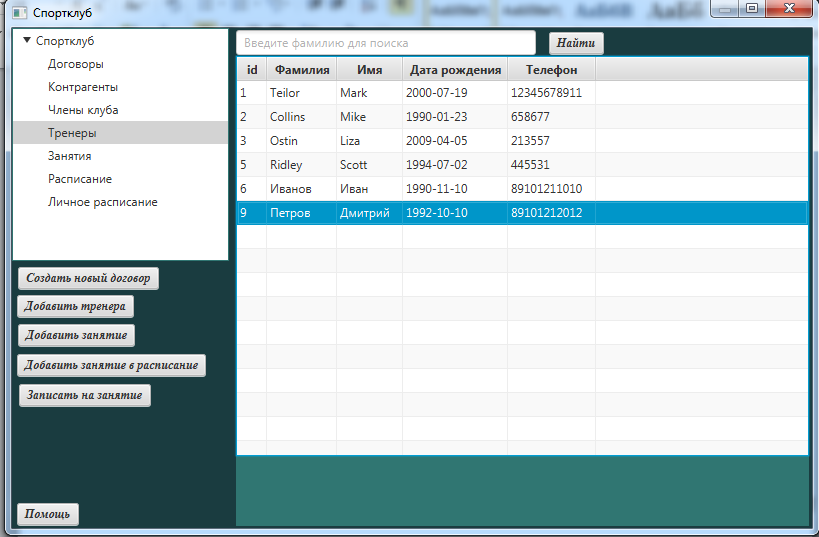
**Скриншоты работы программы**

Добавление записи в базу:

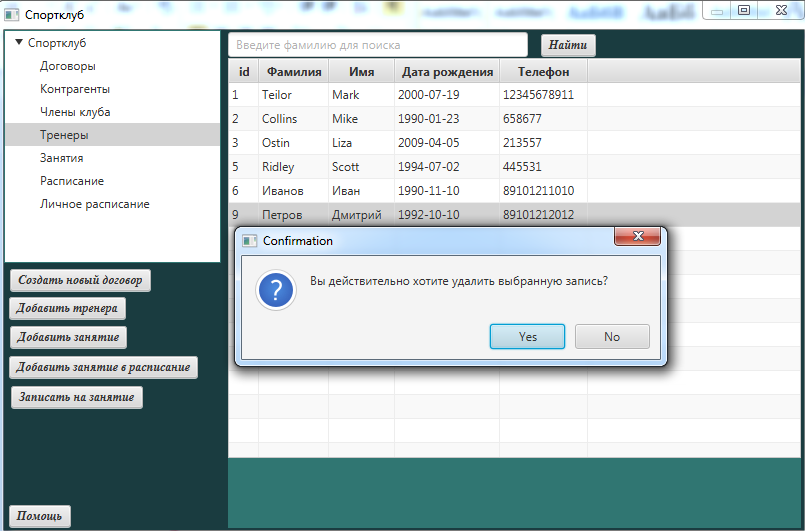


Изменение данных:

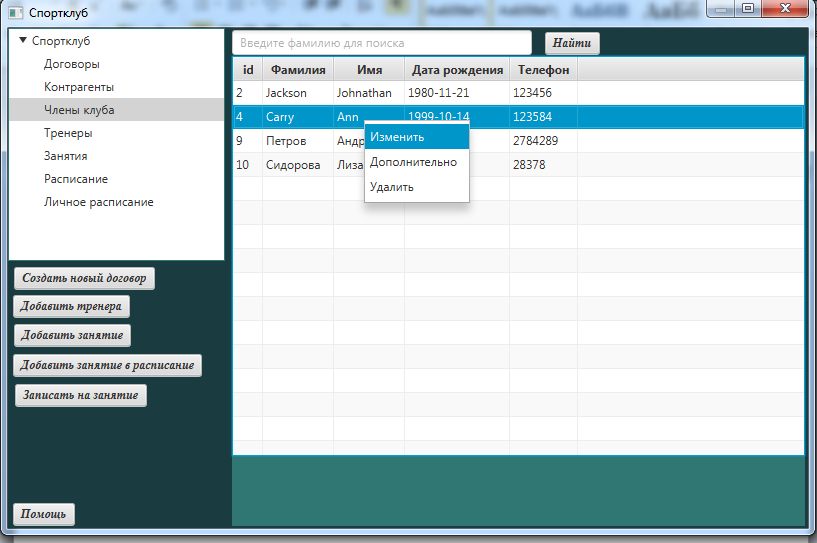




Удаление записи:



Вызов контекстного меню:



Нажатие на кнопку «Помощь»:

