**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Федеральное государственное автономное образовательное учреждение  
высшего образования**

**«Санкт-Петербургский государственный университет  
аэрокосмического приборостроения»**

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

КУРСОВОЙ ПРОЕКТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| старший преподаватель |  |  |  | Шумова Е. О. |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ |
| **Разработка приложения для организации взаимодействия объектов при заданных критериях** |
| по курсу: ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ |
|  |

РАБОТУ ВЫПОЛНИЛА

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТКА ГР. № | 4131 |  | 20.12.2023 |  | Кресик Е.А. |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург, 2023

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Федеральное государственное автономное образовательное учреждение  
высшего образования**

**«Санкт-Петербургский государственный университет  
аэрокосмического приборостроения»**

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

**Задание**

**на курсовой проект по дисциплине «Объектно-ориентированное программирование»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| СТУДЕНТУ ГР.№ | 4131 |  |  | Кресик Е.А. |
|  |  |  |  | ФИО |

Тема «Разработка приложения для организации взаимодействия объектов при заданных критериях»

Исходные данные: Турагенство

Проект должен содержать:

• анализ предметной области

• разработку классов

• разработку тестового приложения

• оформление пояснительной записки по результатам выполнения проекта

• создание презентации к проекту

Срок сдачи законченного проекта \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Руководитель проекта \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ст.преп. Е.О.Шумова

Дата выдачи задания 01.09.2023 г.

Оглавление

[1. Постановка задачи: 4](#_Toc153401843)

[1.1 Анализ предметной области: 4](#_Toc153401844)

[1.2 Формулировка технического задания: 4](#_Toc153401845)

[2. Разработка классов: 5](#_Toc153401846)

[2.1 Классы сущностей: 5](#_Toc153401847)

[2.2 Интерфейсные классы: 5](#_Toc153401848)

[2.3 Управляющие классы: 6](#_Toc153401849)

[2.4 Классы-хранилища: 6](#_Toc153401850)

[2.5 Используемые паттерны и их UML-диаграммы: 6](#_Toc153401851)

[2.6 Полная диаграмма классов: 7](#_Toc153401852)

[3. Разработка приложения: 8](#_Toc153401853)

[3.1 Разработка интерфейса: 8](#_Toc153401854)

[3.2 Реализация методов класса: 18](#_Toc153401855)

[4. Тестирование приложения 22](#_Toc153401856)

[Список использованной литературы: 40](#_Toc153401857)

[Приложение 1 41](#_Toc153401858)

# Постановка задачи:

## Анализ предметной области:

Объектом исследования представленной работы является Турагенство.

Для повышения качества обслуживания и ускорения обслуживания необходимо иметь программу для работы с клиентами при продаже билетов.

Необходимо разработать систему классов, которая поможет хранить взаимосвязанные сущности и осуществлять с ними различные операции.

Сущности:

«Билет»

* ID тура
* ID туриста
* Дату покупки
* ID билета

«Тур»

* ID тура
* Место отдыха
* Период отдыха
* Стоимость

«Турист»

* ID туриста
* ФИО
* Дата рождения

## Формулировка технического задания:

Необходимо разработать и реализовать систему классов «Продажа билетов»

Система должна быть способна реализовывать следующие задачи:

1. Работа с билетами
   1. Добавление билета
   2. Удаление билета
   3. Просмотр билетов
2. Работа с турами
   1. Добавление тура
   2. Удаление тура
   3. Просмотр туров
3. Работа с туристами
   1. Добавление туриста
   2. Удаление туриста
   3. Просмотр туристов

# Разработка классов:

## Классы сущностей:

Представлено три класса сущностей:

Сущность билет представляет собой класс Ticket, который хранит данные о билетах:

* tour\_ID (ID тура)
* tourist\_ID (ID туриста)
* date\_purchase (Дату покупки)
* ticket\_ID (ID билета)

Сущность туров представляет собой класс Tour, который хранит данные о турах:

* tour\_ID (ID тура)
* tour\_location (Место отдыха)
* tour\_time (Период отдыха)
* tour\_cost (Стоимость)

Сущность туристов представляет собой класс Tourist, который, хранит данные выдаче туристах:

* tourist\_ID (ID туриста)
* tourist\_name (ФИО)
* tourist\_birthday (Дата рождения)

## Интерфейсные классы:

Tourists, Tours, Tickets, Ticket\_add\_form, Tour\_add\_form, Tourist\_add\_form, Ticket\_del\_form, Tour\_del\_form, Tourist\_del\_form – классы для возможности удаления, добавления, просмотра данных

## Управляющие классы:

MainWindow — класс отвечает за глобальное взаимодействие сущностей.

## Классы-хранилища:

Database — класс, в котором хранятся классы сущностей.

## Используемые паттерны и их UML-диаграммы:

* Паттерн Наблюдатель:

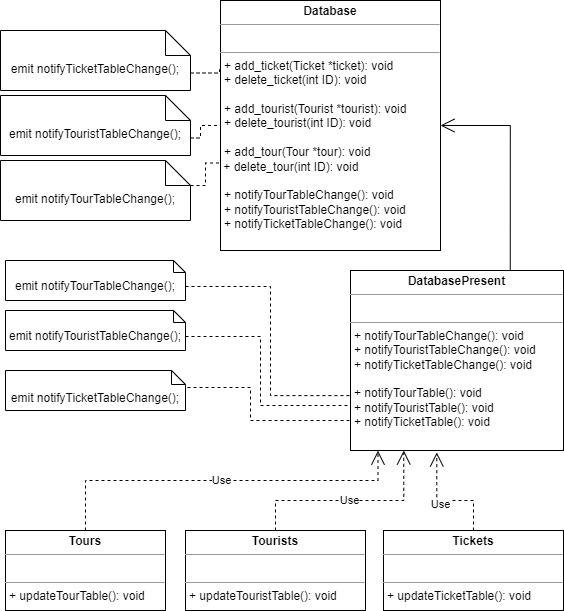
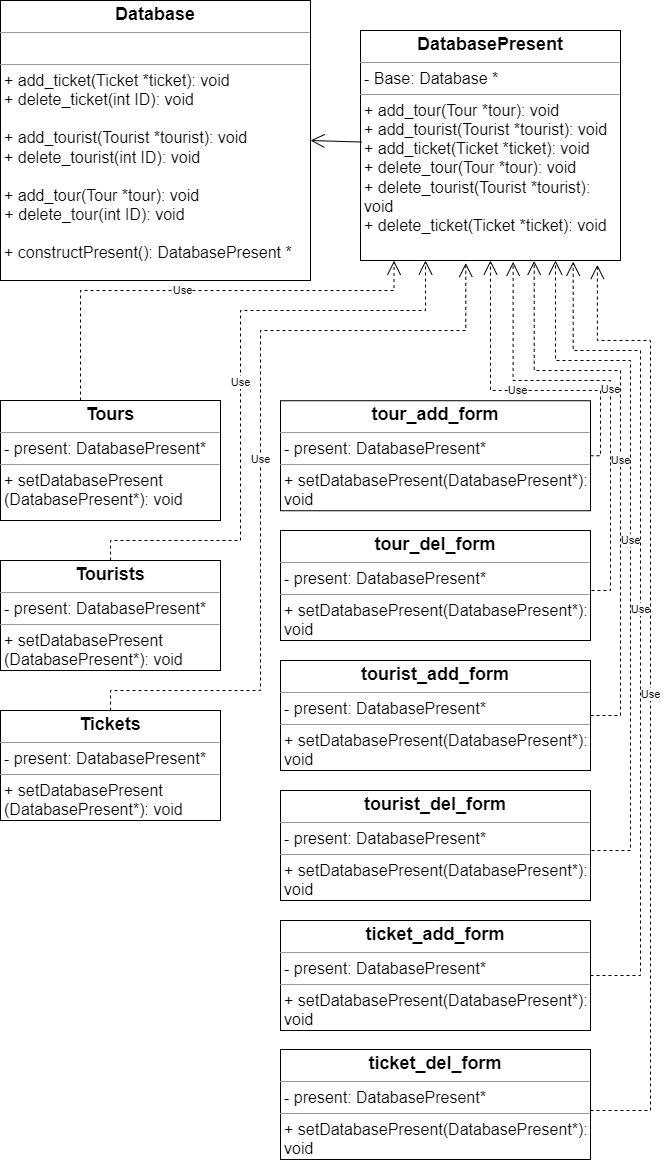


Рисунок 1 – Диаграмма паттерна Наблюдатель

* Паттерн Посредник



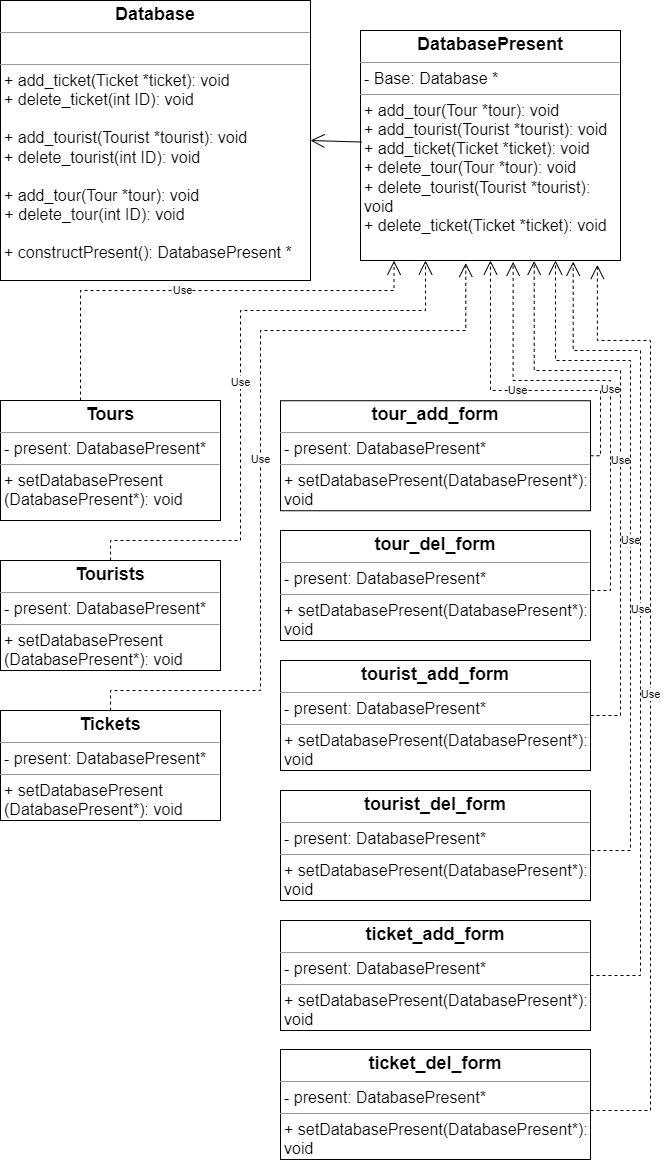


Рисунок 2 – Диаграмма паттерна Посредник

## Полная диаграмма классов:





Рисунок 3 – Общая диаграмма классов

# Разработка приложения:

## Разработка интерфейса:

Для просмотра всех билетов есть Форма для просмотра всех билетов, где можно увидеть список билетов, добавить новый билет, удалить существующий билет. При нажатии кнопки «Добавить» откроется Форма для добавления билета, при нажатиии кнопки «Удалить» откроется Форма для удаления билета.

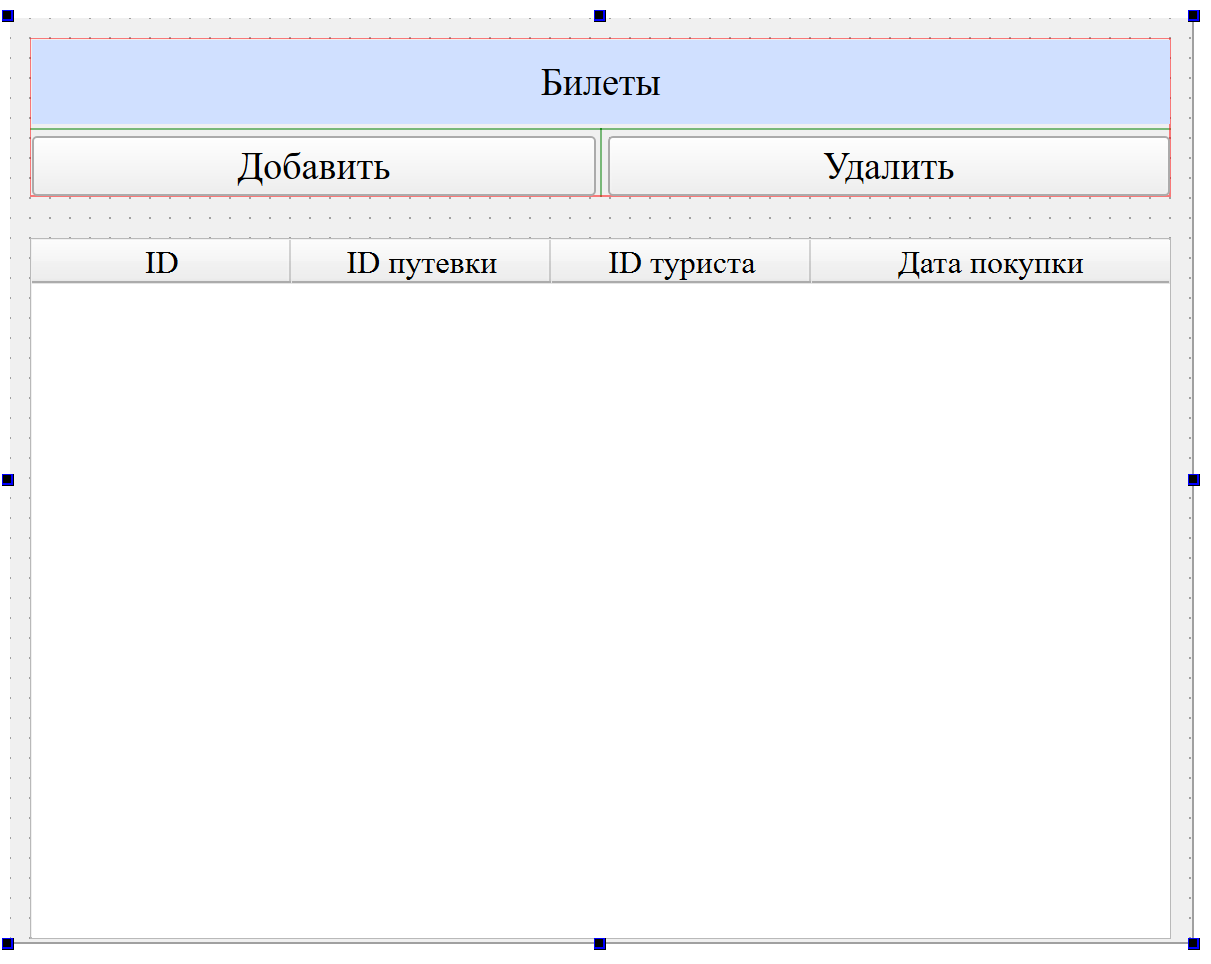
****

Рисунок 4 – Форма для просмотра всех билетов

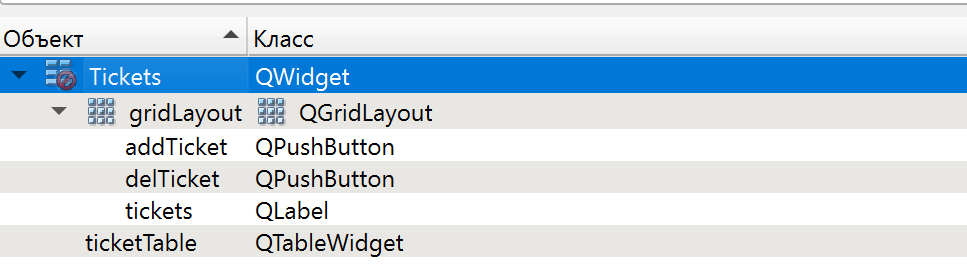
****

Рисунок 5 – Объекты с Формы для просмотра всех билетов

Для добавления нового билета существует Форма для добавления билета, куда пользователь вводит данные билета. Есть поля для ввода ID тура, ID туриста, Даты покупки и ID билета. При нажатии кнопки «Добавить» введенный билет добавляется в базу, если существует такой тур, турист и не существует билет с таким ID, иначе выводится сообщение о соответствующей ошибке. Далее форма закрывается.

****

Рисунок 6 – Форма для добавления билета

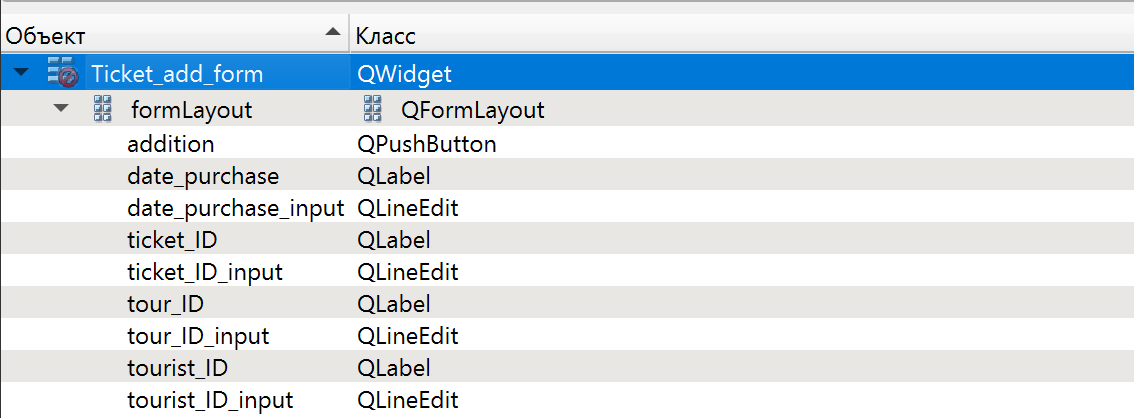
****

Рисунок 7 – Объекты с Формы для добавления билета

Для удаления билета существует Форма для удаления билета, куда пользователь вводит ID билета. Есть поле для ввода ID билета. При нажатии кнопки «Удалить» введенный билет удаляется из базы, если существует билет с таким ID, иначе выводится сообщение об ошибке. Далее форма закрывается.

****

Рисунок 8 – Форма для удаления билета

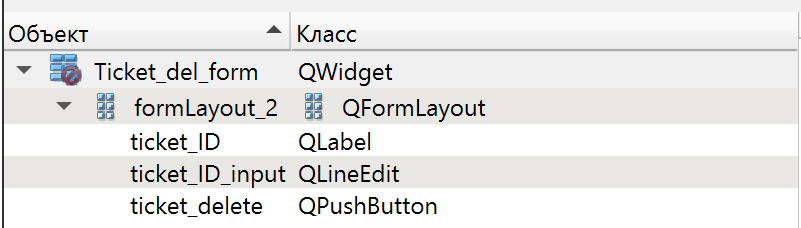
****

Рисунок 9 – Объекты с формы для удаления билета

Для просмотра всех туристов есть Форма для просмотра всех туристов, где можно увидеть список туристов, добавить нового туриста, удалить существующего туриста. При нажатии кнопки «Добавить» откроется Форма для добавления туриста, при нажатиии кнопки «Удалить» откроется Форма для удаления туриста.

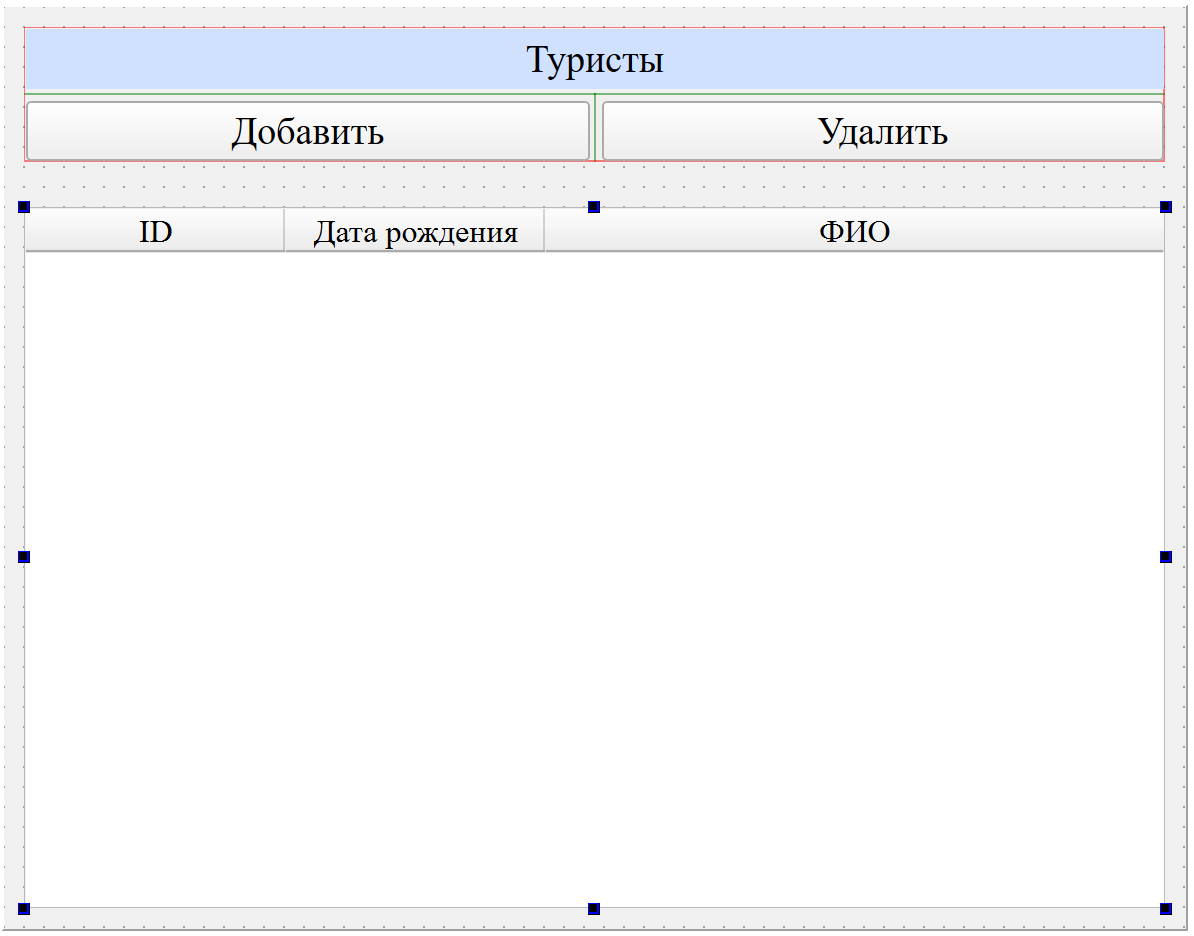
****

Рисунок 10 – Форма для просмотра всех туристов

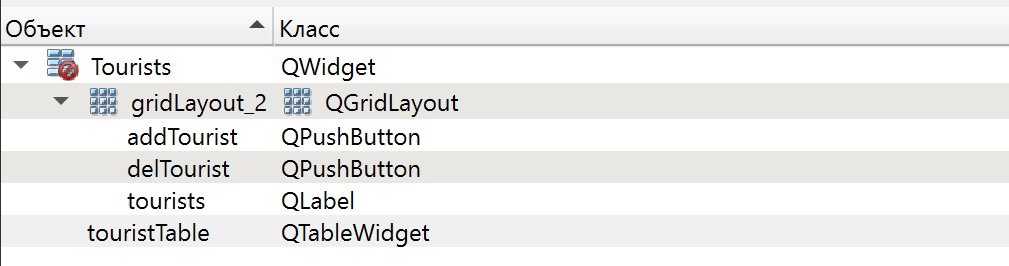
****

Рисунок 11 – Объекты с Формы для просмотра всех туристов

Для добавления нового туриста существует Форма для добавления туриста, куда пользователь вводит данные туриста. Есть поля для ввода ФИО туриста, ID туриста и Даты рождения. При нажатии кнопки «Добавить» введенный турист добавляется в базу, если не существует туриста с таким ID, иначе выводится сообщение об ошибке. Далее форма закрывается.

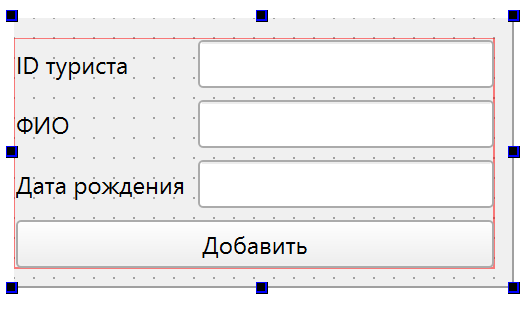
****

Рисунок 12 – Форма для добавления туриста

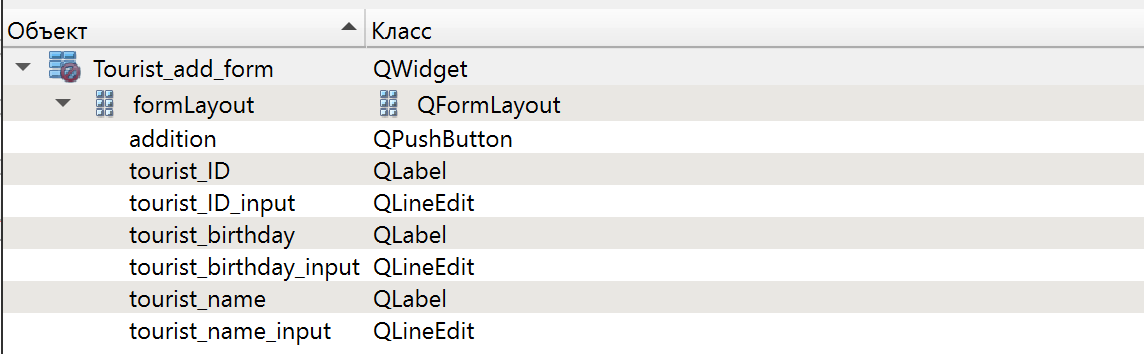
****

Рисунок 13 – Объекты с Формы для добавления туриста

Для удаления туриста существует Форма для удаления туриста, куда пользователь вводит ID туриста. Есть поле для ввода ID туриста. При нажатии кнопки «Удалить» введенный турист удаляется из базы, если существует турист с таким ID, и не существует билета, купленного на этого туриста, иначе выводится сообщение о соответствующей ошибке. Далее форма закрывается.

****

Рисунок 14 – Форма для удаления туриста

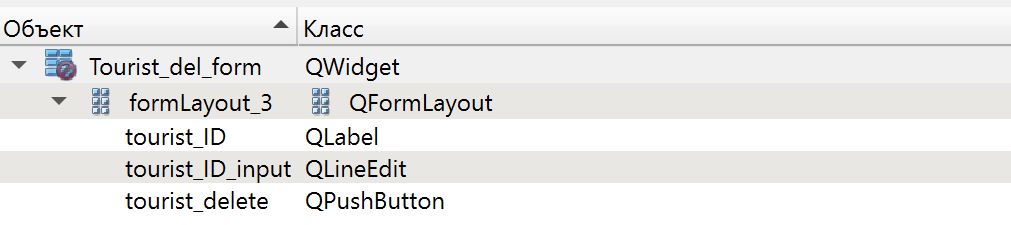
****

Рисунок 15 – Объекты с формы для удаления туриста

Для просмотра всех туров есть Форма для просмотра всех туров, где можно увидеть список туров, добавить новый тур, удалить существующий тур. При нажатии кнопки «Добавить» откроется Форма для добавления тура, при нажатиии кнопки «Удалить» откроется Форма для удаления тура.

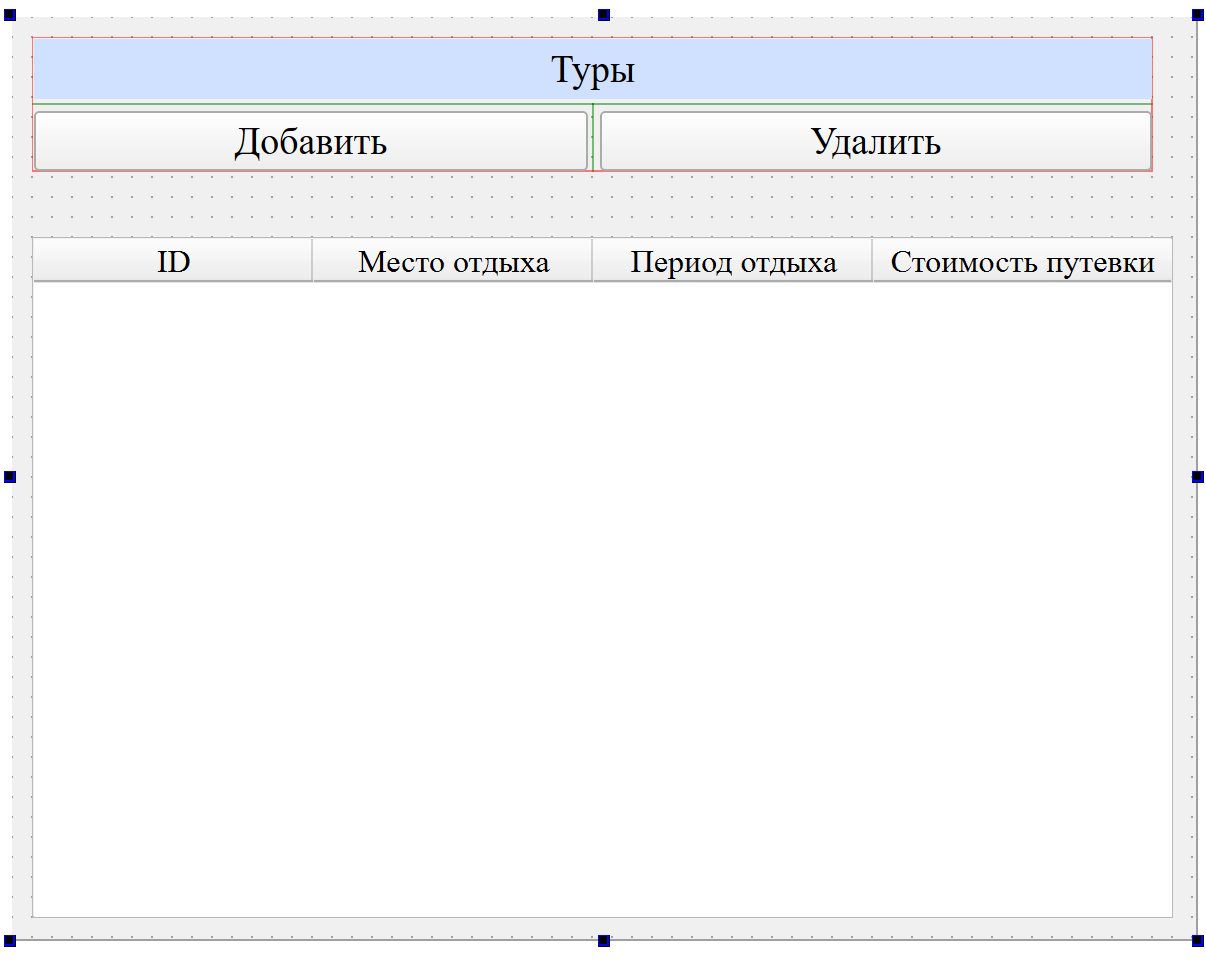
****

Рисунок 16 – Форма для просмотра всех туров

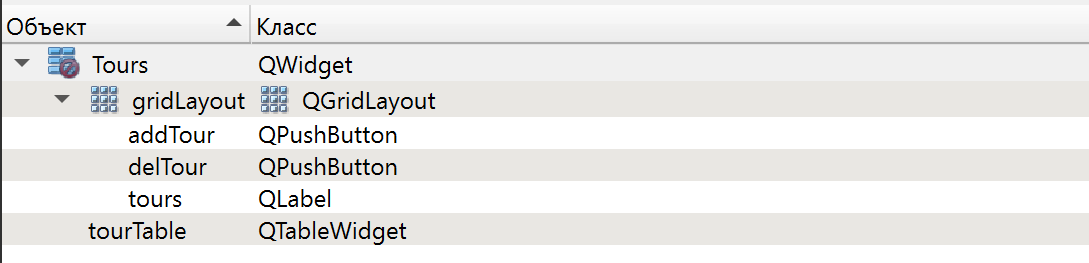
****

Рисунок 17 – Объекты с Формы для просмотра всех туров

Для добавления нового тура существует Форма для добавления тура, куда пользователь вводит данные тура. Есть поля для ввода ID тура, Места отдыха, Периода отдыха и стоимости. При нажатии кнопки «Добавить» введенный тур добавляется в базу, если не существует тур с таким ID, иначе выводится сообщение об ошибке. Далее форма закрывается.

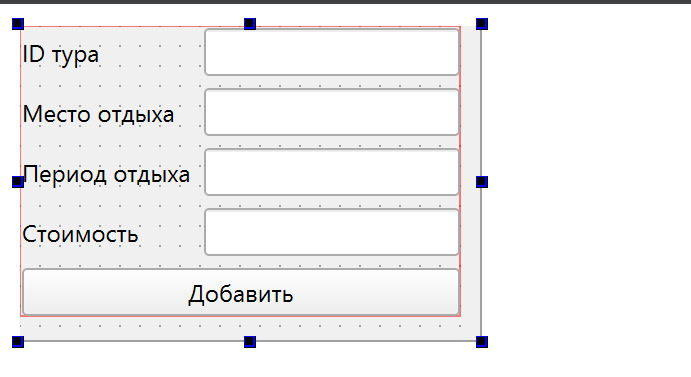
****

Рисунок 18 – Форма для добавления тура

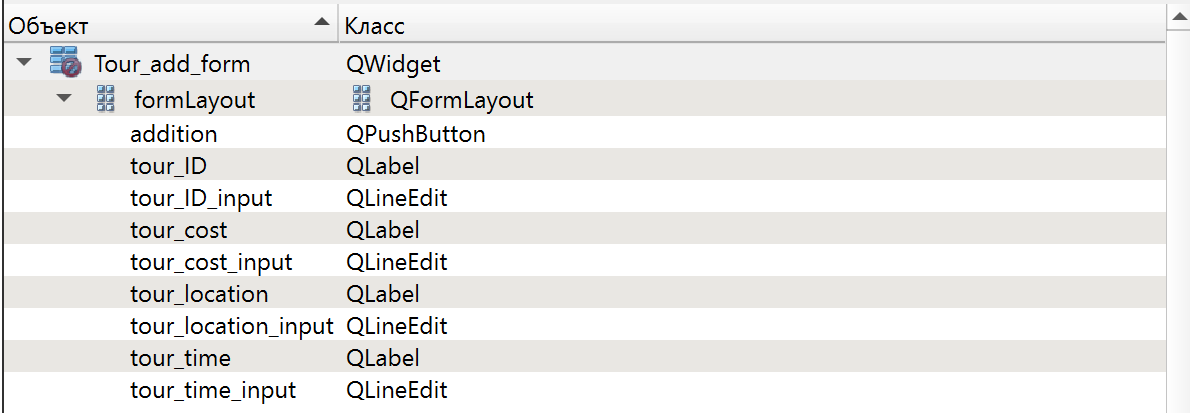
****

Рисунок 19 – Объекты с Формы для добавления тура

Для удаления тура существует Форма для удаления тура, куда пользователь вводит ID тура. Есть поле для ввода ID тура. При нажатии кнопки «Удалить» введенный тур удаляется из базы, если существует тур с таким ID, и не существует билетов, купленных на этот тур, иначе выводится сообщение о соответствующей ошибке. Далее форма закрывается.

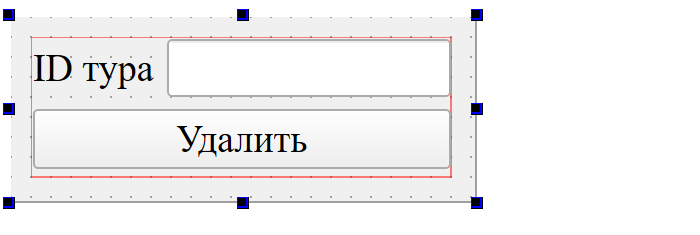
****

Рисунок 20 – Форма для удаления тура

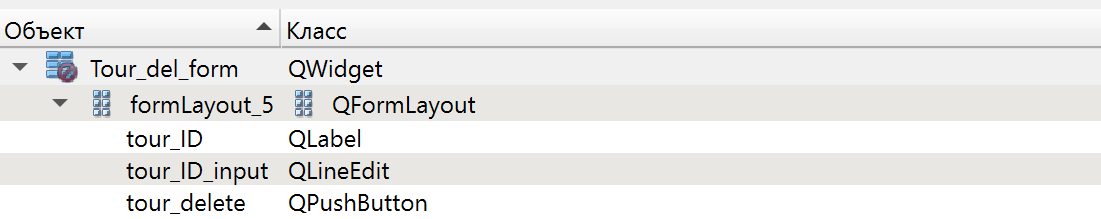
****

Рисунок 21 – Объекты с формы для удаления тура

Существует главная форма для выбора действий. При нажатии на кнопку «Посмотреть туры» происходит переход на Форму для просмотра всех билетов. При нажатии на кнопку «Посмотреть туристов» происходит переход на Форму для просмотра всех туристов. При нажатии на кнопку «Посмотреть билеты» происходит переход на Форму для просмотра всех билетов.

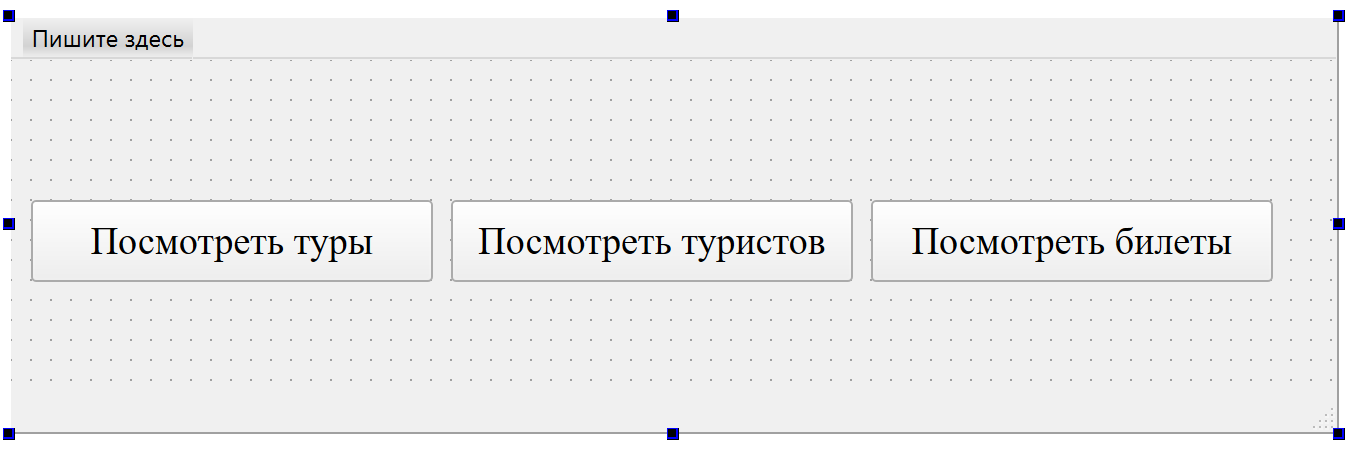
****

Рисунок 10 – Форма главного экрана

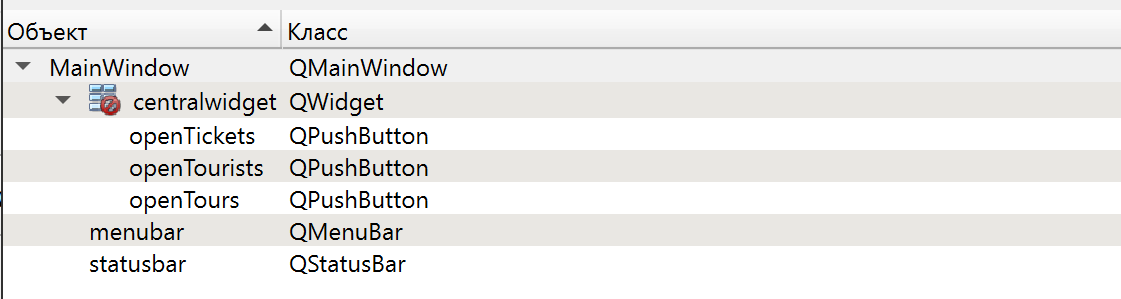
****

Рисунок 11 – Объекты с Формы главного экрана

## Реализация методов класса:

**Database**

Класс для хранения БД.

Поля:

* QVector<Ticket \*> ticket\_array; - для хранения списка билетов
* QVector<Tour \*> tour\_array; - для хранения списка туров
* QVector<Tourist \*> tourist\_array; - для хранения списка туристов

Методы:

* void notifyTourTableChange();

Сигнал для уведомления наблюдателей, что таблицу туров надо обновить.

* void notifyTouristTableChange();

Сигнал для уведомления наблюдателей, что таблицу туристов надо обновить.

* void notifyTicketTableChange();

Сигнал для уведомления наблюдателей, что таблицу билетов надо обновить.

* void add\_tour(Tour \*tour);

Функция для добавления тура в БД. В функцию передается объект класса Tour. Этот объект добавляется в вектор tour\_array. После добавления отправляется сигнал notifyTourTableChange().

* void add\_tourist(Tourist \*tourist);

Функция для добавления туриста в БД. В функцию передается объект класса Tourist. Этот объект добавляется в вектор tourist\_array. После добавления отправляется сигнал notifyTouristTableChange().

* void add\_ticket(Ticket \*ticket);

Функция для добавления билета в БД. В функцию передается объект класса Ticket. Этот объект добавляется в вектор ticket\_array. После добавления отправляется сигнал notifyTicketTableChange().

* void delete\_tour(int ID);

Функция для удаления тура из БД. В функцию передается объект класса Tour. Этот объект удаляется из вектора tour\_array. После удаления отправляется сигнал notifyTourTableChange().

* void delete\_tourist(int ID);

Функция для удаления туриста из БД. В функцию передается объект класса Tourist. Этот объект удаляется из вектора tourist\_array. После удаления отправляется сигнал notifyTouristTableChange().

* void delete\_ticket(int ID);

Функция для удаления билета из БД. В функцию передается объект класса Ticket. Этот объект удаляется из вектора ticket\_array. После удаления отправляется сигнал notifyTicketTableChange().

* DatabasePresent \*constructPresent(int i);

Функция для формирования представления БД (используется для паттерна «Посредник»). Создается указатель на объект класса DatabasePresent, в конструктор передаются адреса списков базы данных (ticket\_array, tour\_array, tourist\_array) а также адрес самой БД.

Далее соединяется сигнал notifyTourTableChange() со слотом notifyTourTable(), вызываемом в созданном ранее объекте класса DatabasePresent. Далее соединяется сигнал notifyTouristTableChange() со слотом notifyTouristTable(), вызываемом в созданном ранее объекте класса DatabasePresent. Далее соединяется сигнал notifyTicketTableChange() со слотом notifyTicketTable(), вызываемом в созданном ранее объекте класса DatabasePresent.

**DatabasePresent**

Поля:

* const QVector<Ticket \*>\* ticket\_array;

Для хранения указателя на список билетов в БД

* const QVector<Tour \*>\* tour\_array;

Для хранения указателя на список туров в БД

* const QVector<Tourist \*>\* tourist\_array;

Для хранения указателя на список туристов в БД

* Database \*Base;

Для хранения указателя на БД (используется для паттерна «Посредник»)

Методы:

* void notifyTourTableChange();

Сигнал для уведомления наблюдателей, что таблицу туров надо обновить.

* void notifyTouristTableChange();

Сигнал для уведомления наблюдателей, что таблицу туристов надо обновить.

* void notifyTicketTableChange();

Сигнал для уведомления наблюдателей, что таблицу билетов надо обновить.

* void notifyTourTable();

Отправляет сигнал notifyTourTableChange() объекту класса Tours, что таблицу туров надо обновить.

* void notifyTouristTable();

Отправляет сигнал notifyTouristTableChange() объекту класса Tourists, что таблицу туристов надо обновить.

* void notifyTicketTable();

Отправляет сигнал notifyTourTickeChange() объекту класса Tickets, что таблицу билетов надо обновить.

* void add\_tour(Tour \*tour);

Вызывает у объекта класса Database метод добавления тура в БД

* void add\_tourist(Tourist \*tourist);

Вызывает у объекта класса Database метод добавления туриста в БД

* void add\_ticket(Ticket \*ticket);

Вызывает у объекта класса Database метод добавления билета в БД

* void delete\_tour(int ID);

Вызывает у объекта класса Database метод удаления тура из БД

* void delete\_tourist(int ID);

Вызывает у объекта класса Database метод удаления туриста из БД

* void delete\_ticket(int ID);

Вызывает у объекта класса Database метод удаления билета из БД

* const QVector<Ticket \*>\* get\_ticket\_array();

Для получения указателя на список билетов в БД

* const QVector<Tour \*>\* get\_tour\_array();

Для получения указателя на список туров в БД

* const QVector<Tourist \*>\* get\_tourist\_array();

Для получения указателя на список туристов в БД

**MainWindow**

Поля:

* Ui::MainWindow \*ui; - объект формы
* Database Base; - хранит базу данных

Методы:

* void on\_openTours\_clicked();

Создается объект класса Tours. Для этого объекта устанавливается значение представления БД, которое конструируется с помощью метода constructPresent объекта класса Database. Открывает форму для просмотра всех туров. Обновляет таблицу туров на форме.

* void on\_openTourists\_clicked();

Создается объект класса Tourists. Для этого объекта устанавливается значение представления БД, которое конструируется с помощью метода constructPresent объекта класса Database. Открывает форму для просмотра всех туристов. Обновляет таблицу туристов на форме.

* void on\_openTickets\_clicked();

Создается объект класса Tickets. Для этого объекта устанавливается значение представления БД, которое конструируется с помощью метода constructPresent объекта класса Database. Открывает форму для просмотра всех билетов. Обновляет таблицу билетов на форме.

**Ticket**

Поля:

* int ticket\_ID; - ID билета
* int tour\_ID; - ID тура
* int tourist\_ID; - ID туриста
* QString date\_purchase; - дата покупки билета

Методы:

* QString getDatePurchase (); - для получения даты покупки билета
* int getTouristID (); - для получения ID туриста
* int getTourID(); - для получения ID тура
* int getTicketID (); - для получения ID билета

**Ticket\_add\_form**

Поля:

* Ui::Ticket\_add\_form \*ui; - объект формы
* DatabasePresent\* present; - хранит адрес представления БД

Методы:

* void on\_addition\_clicked();

Получаем данные ID тура, ID туриста, ID билета с формы. Проверяем, есть ли уже билет с таким ID в БД. Если есть, то уведомляем об ошибке. Проверяем, есть ли данные турист и тур в БД, если нет – уведомляем об ошибке. Если есть – создаем объект класса Ticket, добавляем его в БД. Закрываем форму.

* void setDatabasePresent(DatabasePresent\*);

Для установления значения адреса представления БД.

**Ticket\_del\_form**

Поля:

* Ui::Ticket\_del\_form \*ui; - объект формы
* DatabasePresent\* present; - хранит адрес представления БД

Методы:

* void on\_ ticket\_delete\_clicked();

Получаем ID билета с формы. Проверяем, есть ли уже билет с таким ID в БД. Если нет, то уведомляем об ошибке. Если есть – вызываем метод delete\_ticket у объекта класса DatabasePresent. Закрываем форму.

* void setDatabasePresent(DatabasePresent\*);

Для установления значения адреса представления БД

**Tickets**

Поля:

* Ui::Tickets \*ui; - объект формы
* DatabasePresent\* present; - хранит адрес представления БД

Методы:

* void on\_addTicket\_clicked();

Создаем объект класса формы Ticket\_add\_form, устанавливаем для него значение представления БД (вызываем метод setDatabasePresent). Открываем форму для добавления билета.

* void on\_delTicket\_clicked();

Создаем объект класса формы Ticket\_del\_form, устанавливаем для него значение представления БД (вызываем метод setDatabasePresent). Открываем форму для удаления билета.

* void updateTicketTable();

Для обновления таблицы билетов. Очищаем строки в таблице билетов на форме. Получаем список билетов из БД. Заполняем таблицу билетов на форме актуальными значениями.

* void setDatabasePresent(DatabasePresent\*);

Для установления значения адреса представления БД. Если объект класса DatabasePresent содержит не пустое значение, то отсоединяем сигнал notifyTicketTableChange() со слотом updateTicketTable(). Далее делаем значение объекта класса DatabasePresent равным переданному в функцию значению. Соединяем сигнал notifyTicketTableChange() со слотом updateTicketTable().

**Tour**

Поля:

* QString tour\_location; - хранится место отдыха
* int tour\_ID; - хранится ID тура
* QString tour\_time; - хранится период отдыха
* int tour\_cost; - хранится стоимость тура

Методы:

* QString getLocation (); - для получения места отдыха
* int getID (); - для получения ID тура
* QString getTime (); - для получения периода отдыха
* int getCost (); - для получения стоимости тура

**Tour\_add\_form**

Поля:

* Ui::Tour\_add\_form \*ui; - объект формы
* DatabasePresent\* present; - хранит адрес представления БД

Методы:

* void on\_addition\_clicked();

Получаем значение ID тура с формы. Проверяем, есть ли уже тур с таким ID в БД. Если есть, то уведомляем об ошибке и закрываем форму. Если нет – создаем объект класса Tour, добавляем его в БД. Закрываем форму.

* void setDatabasePresent(DatabasePresent\*);

Для установления значения адреса представления БД

**Tour\_del\_form**

Поля:

* Ui::Tour\_add\_form \*ui; - объект формы
* DatabasePresent\* present; - хранит адрес представления БД

Методы:

* void on\_del\_tour\_clicked();

Получаем значение ID тура с формы. Проверяем, есть ли уже тур с таким ID в БД. Если нет, то уведомляем об ошибке и закрываем форму. Если есть – проверяем, есть ли этот тур в купленных билетах. Если есть, то уведомляем об ошибке, что удалить тур не можем, и закрываем форму. Если нет – удаляем тур из БД. Закрываем форму.

* void setDatabasePresent(DatabasePresent\*);

Для установления значения адреса представления БД

**Tourist**

Поля:

* int tourist\_ID; - ID туриста
* QString tourist\_birthday; - день рождения
* QString tourist\_name; - ФИО туриста

Методы:

* QString getBirthday (); - для получения даты рождения
* int getID (); - для получения ID туриста
* QString getName (); - для получения ФИО туриста

**Tourist\_add\_form**

Поля:

* Ui::Tourist\_add\_form \*ui; - объект формы
* DatabasePresent\* present; - хранит адрес представления БД

Методы:

* void on\_addition\_clicked();

Получаем значение ID туриста с формы. Проверяем, есть ли уже турист с таким ID в БД. Если есть, то уведомляем об ошибке и закрываем форму. Если нет – создаем объект класса Tourist, добавляем его в БД. Закрываем форму.

* void setDatabasePresent(DatabasePresent\*);

Для утстановления значения адреса представления БД

**Tourist\_del\_form**

Поля:

* Ui::Tourist\_del\_form \*ui; - объект формы
* DatabasePresent\* present; - хранит адрес представления БД

Методы:

* void on\_del\_tourist\_clicked();

Получаем значение ID туриста с формы. Проверяем, есть ли уже тур с таким ID в БД. Если нет, то уведомляем об ошибке и закрываем форму. Если есть – проверяем, есть ли этот турист в купленных билетах. Если есть, то уведомляем об ошибке, что удалить туриста не можем, и закрываем форму. Если нет – удаляем туриста из БД. Закрываем форму.

* void setDatabasePresent(DatabasePresent\*);

Для утстановления значения адреса представления БД.

**Tourists**

Поля:

* Ui::Tourists \*ui; - объект формы
* DatabasePresent\* present; - хранит адрес представления БД

Методы:

* void on\_addTourist\_clicked();

Создаем объект класса формы Tourist\_add\_form, устанавливаем для него значение представления БД (вызываем метод setDatabasePresent). Открываем форму для добавления туриста.

* void on\_delTourist\_clicked();

Создаем объект класса формы Tourist\_del\_form, устанавливаем для него значение представления БД (вызываем метод setDatabasePresent). Открываем форму для удаления туриста.

* void updateTouristTable();

Для обновления таблицы туристов. Очищаем строки в таблице туристов на форме. Получаем список туристов из БД. Заполняем таблицу туристов на форме актуальными значениями.

* void setDatabasePresent(DatabasePresent\*);

Для установления значения адреса представления БД. Если объект класса DatabasePresent содержит не пустое значение, то отсоединяем сигнал notifyTicketTableChange() со слотом updateTicketTable(). Далее делаем значение объекта класса DatabasePresent равным переданному в функцию значению. Соединяем сигнал notifyTicketTableChange() со слотом updateTicketTable().

**Tours**

Поля:

* Ui::Tours \*ui; - объект формы
* DatabasePresent\* present; - хранит адрес представления БД

Методы:

* void on\_addTour\_clicked();

Создаем объект класса формы Tour\_add\_form, устанавливаем для него значение представления БД (вызываем метод setDatabasePresent). Открываем форму для добавления тура.

* void on\_delTour\_clicked();

Создаем объект класса формы Tour\_del\_form, устанавливаем для него значение представления БД (вызываем метод setDatabasePresent). Открываем форму для удаления тура.

* void updateTourTable();

Для обновления таблицы туров. Очищаем строки в таблице туров на форме. Получаем список туров из БД. Заполняем таблицу туров на форме актуальными значениями.

* void setDatabasePresent(DatabasePresent\*);

Для установления значения адреса представления БД. Если объект класса DatabasePresent содержит не пустое значение, то отсоединяем сигнал notifyTicketTableChange() со слотом updateTicketTable(). Далее делаем значение объекта класса DatabasePresent равным переданному в функцию значению. Соединяем сигнал notifyTicketTableChange() со слотом updateTicketTable().

# Тестирование приложения

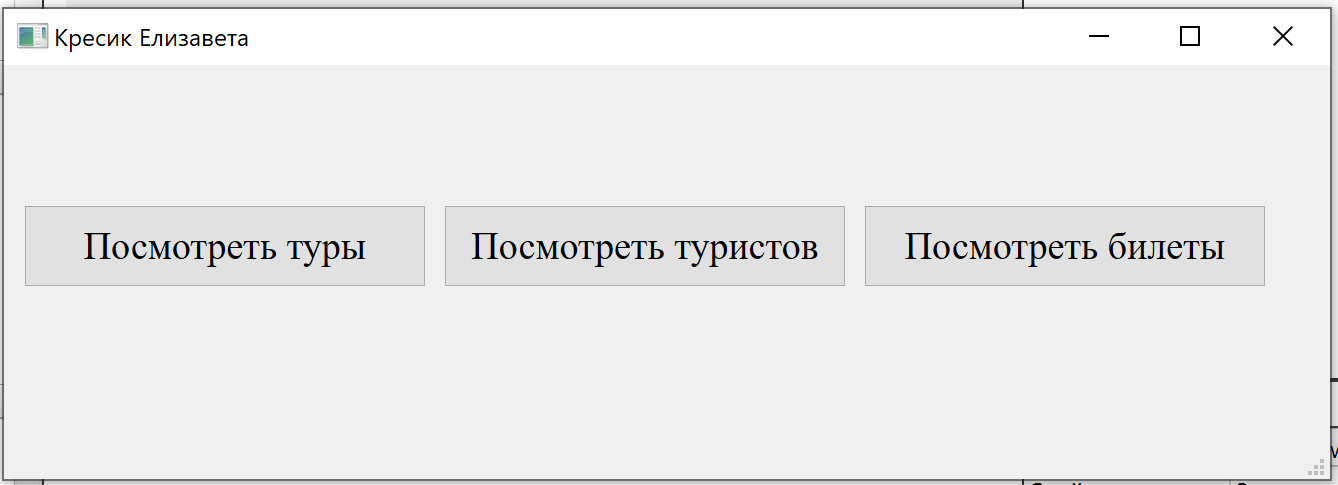
****

Рисунок 12 – При запуске программы показывается главная страница

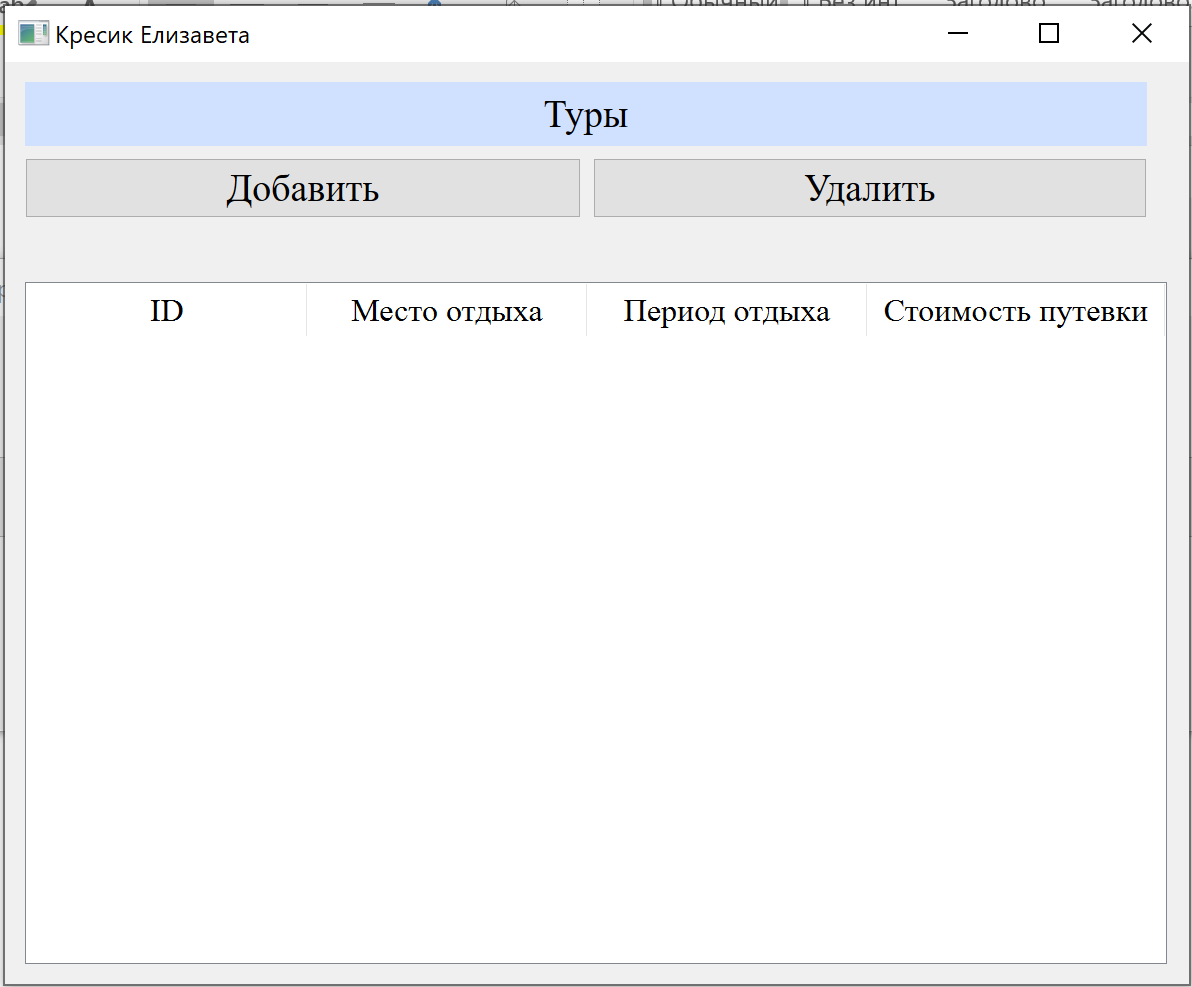


Рисунок 13 – Эта страница открывается после нажатия «Посмотреть туры» на главной странице

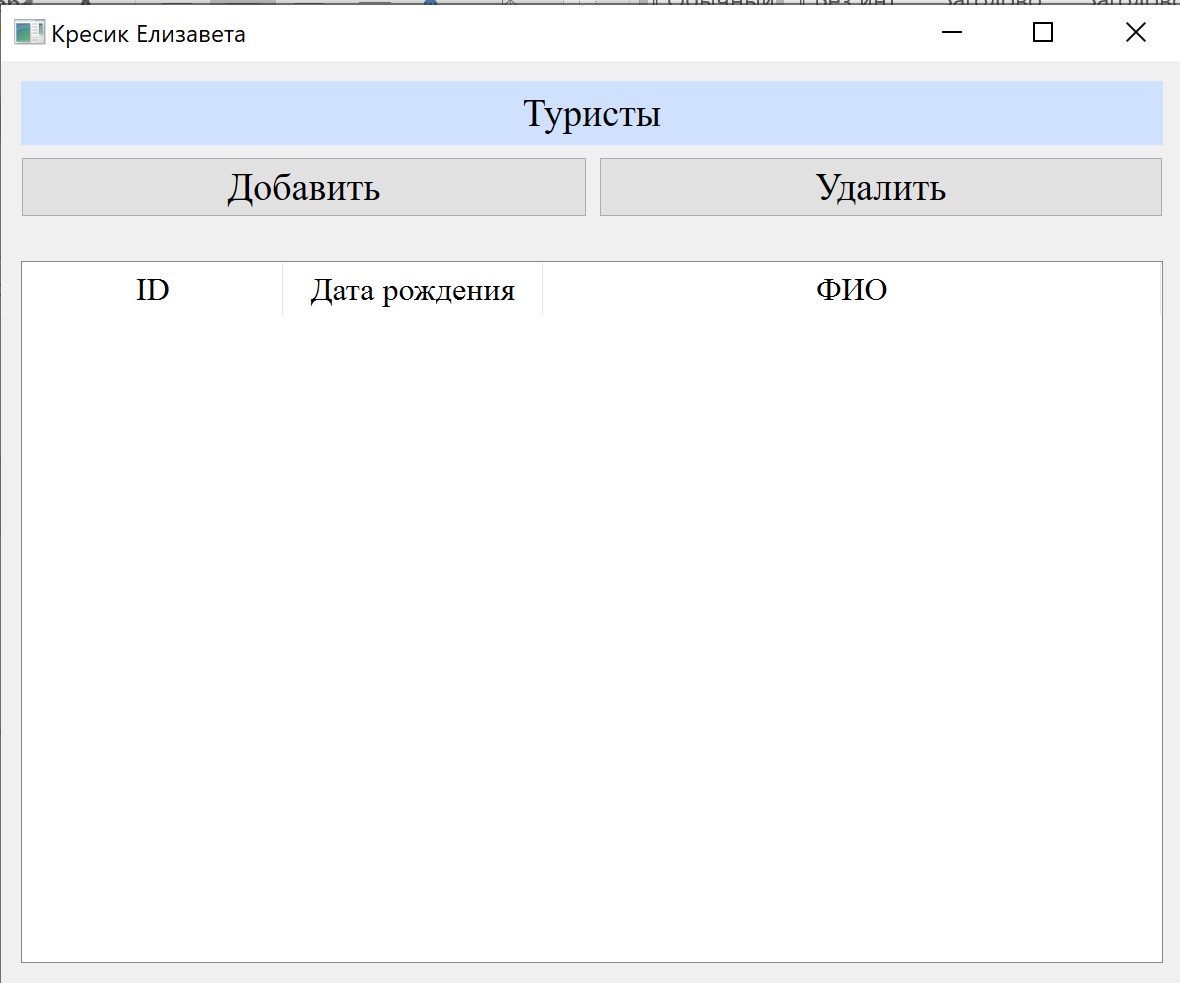


Рисунок 14 – Эта страница открывается после нажатия «Посмотреть туристорв» на главной странице

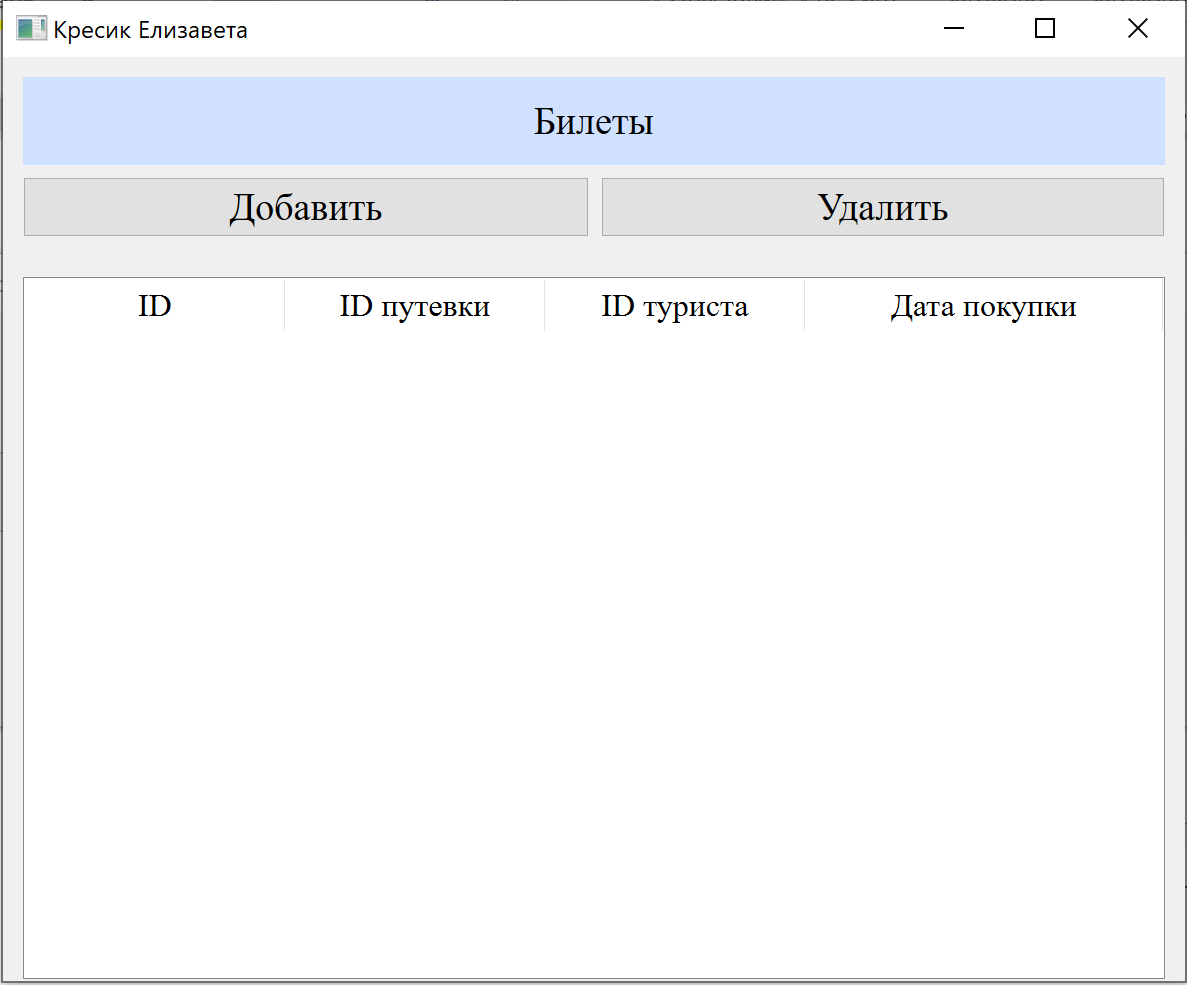


Рисунок 15 – Эта страница открывается после нажатия «Посмотреть билеты» на главной странице

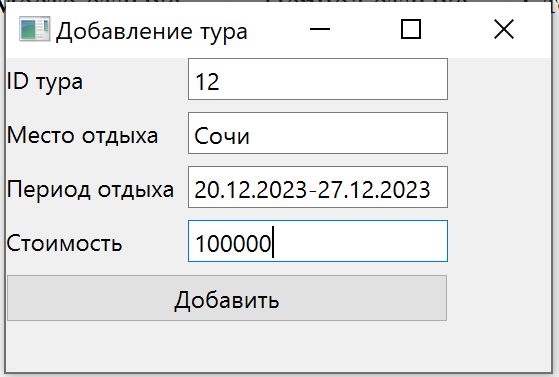


Рисунок 16 – Добавление тура

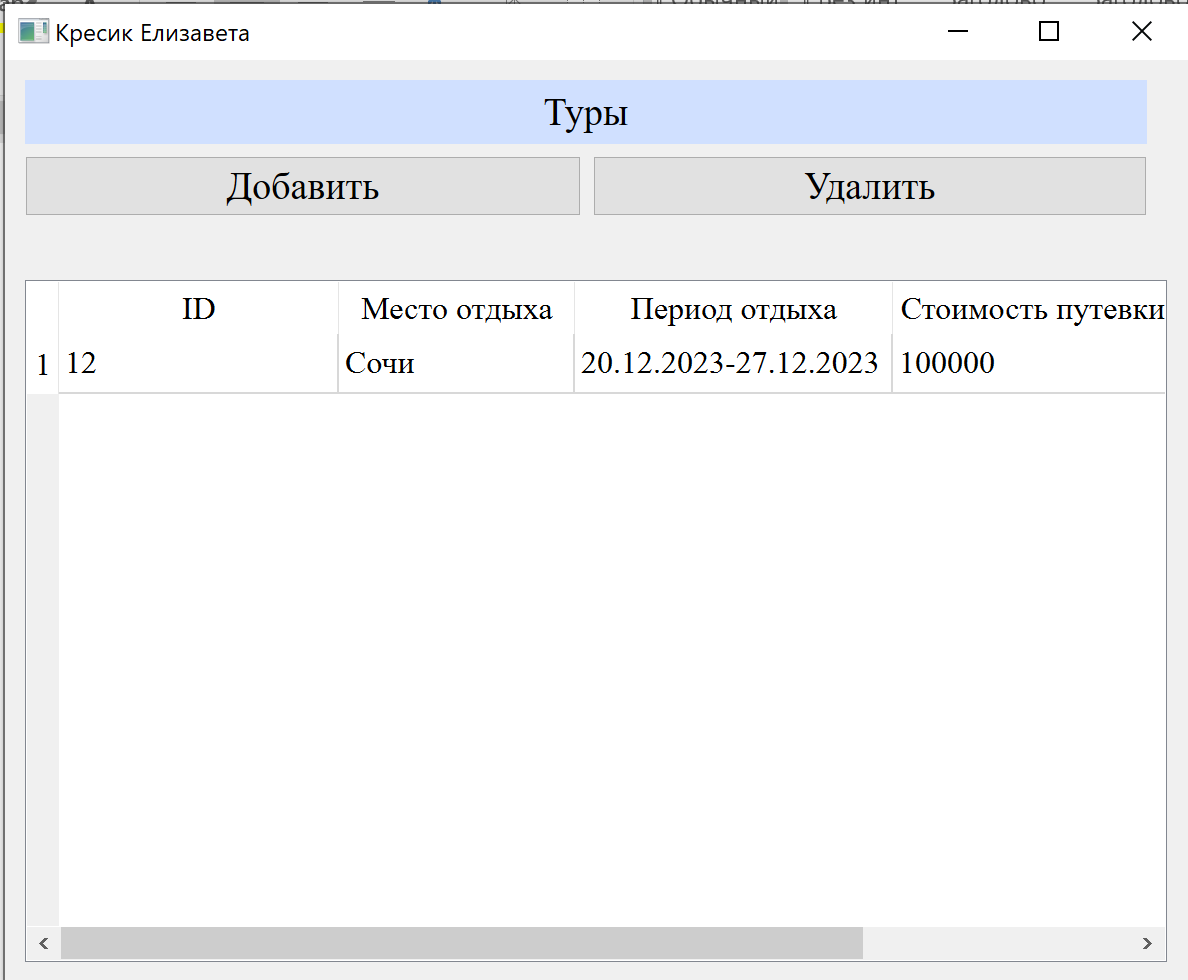


Рисунок 17 – Таблица туров после добавления тура

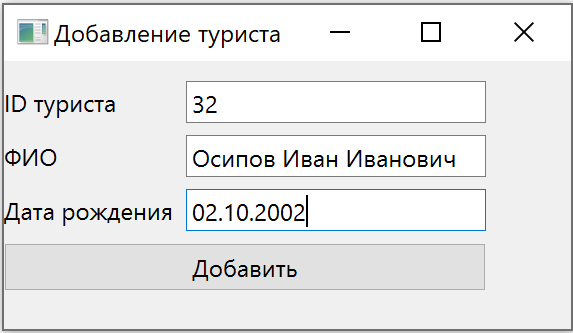


Рисунок 18 – Добавление туриста

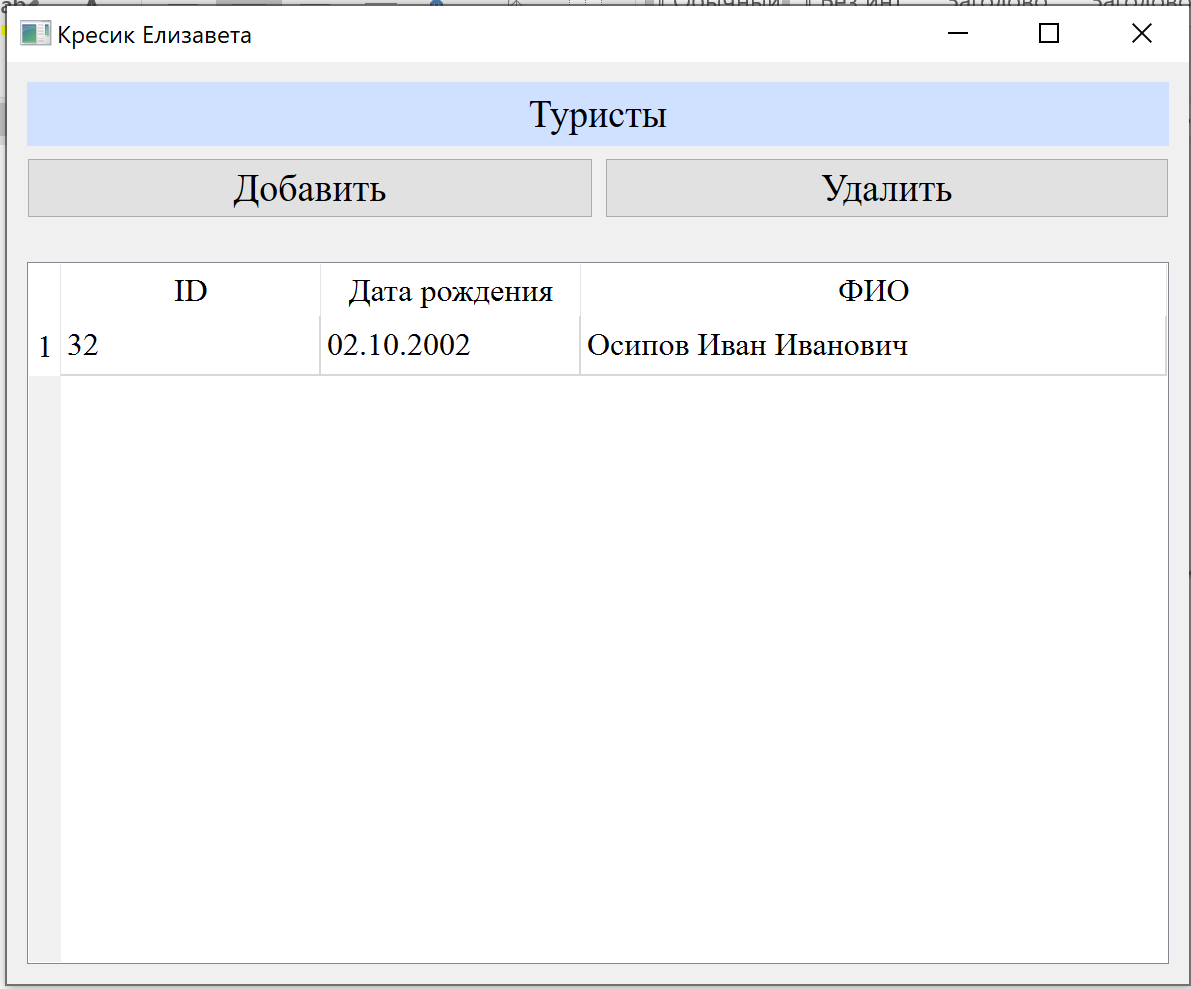


Рисунок 19 – Таблица туристов после добавления туриста

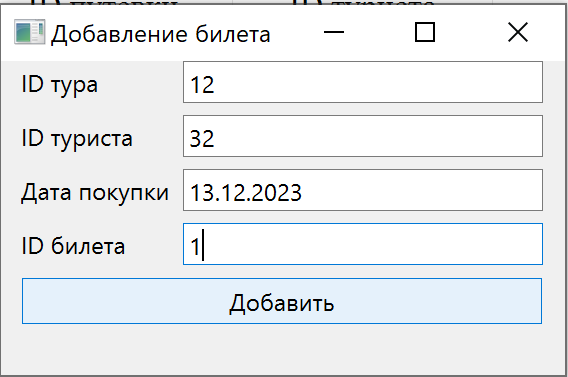


Рисунок 20 – Добавление билета

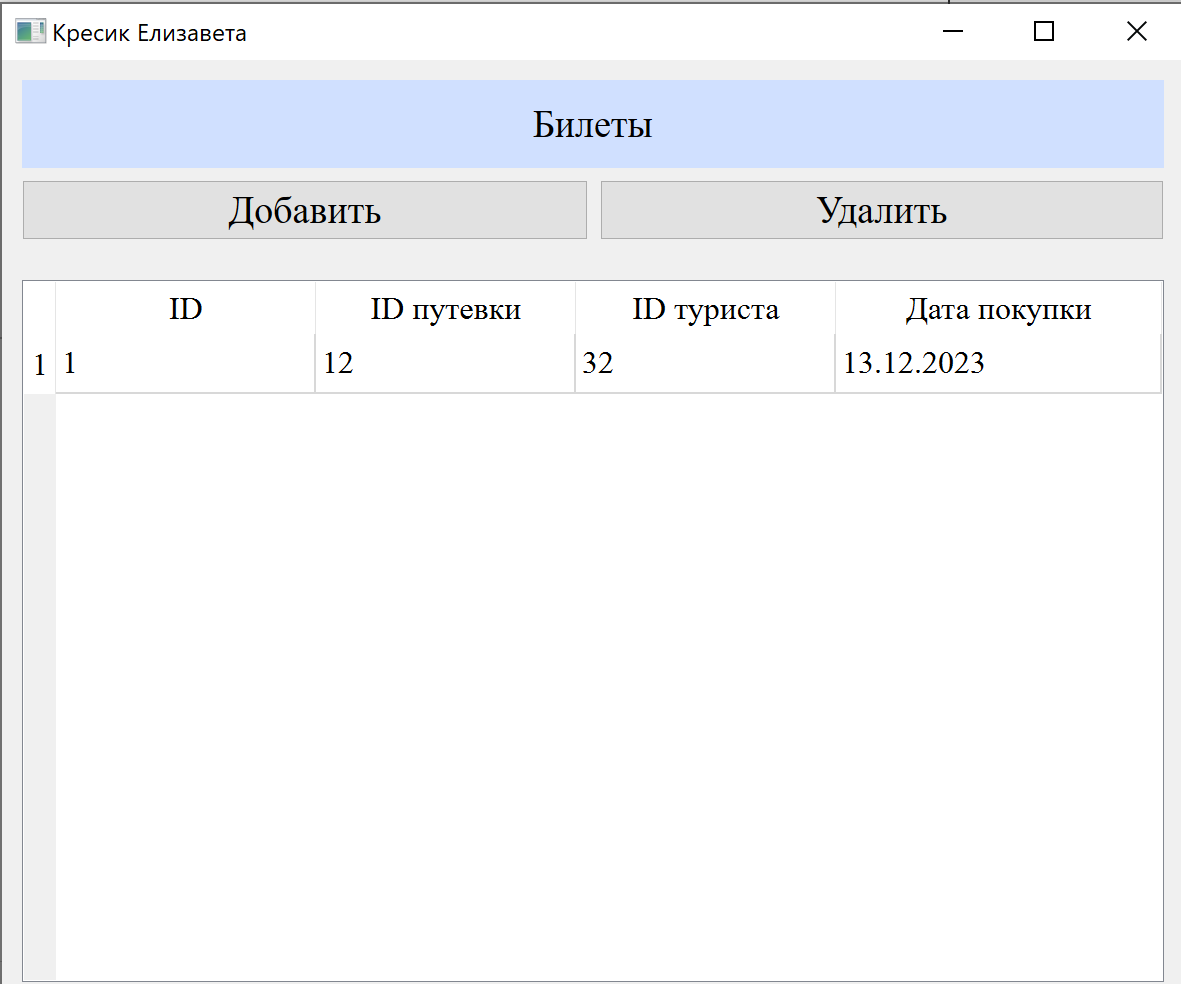


Рисунок 21 – Таблица билетов после добавления билета

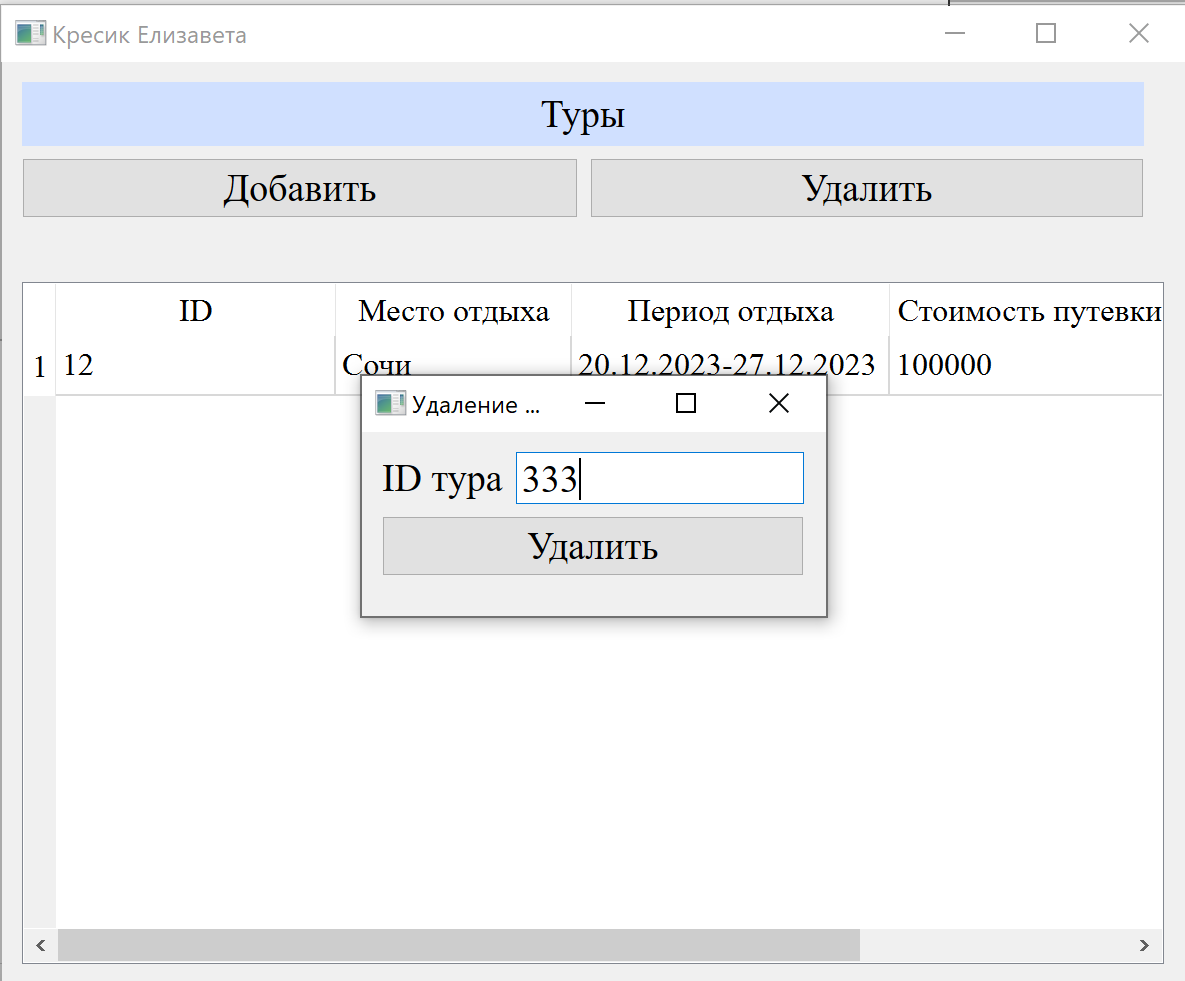


Рисунок 22 – Удаление несуществующего тура

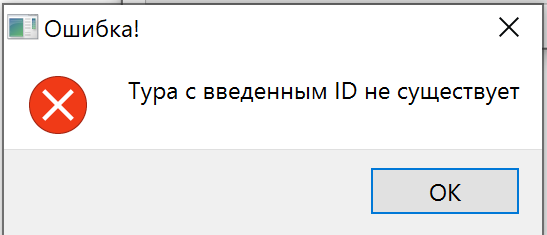


Рисунок 23 – Уведомление об ошибке при удалении несуществующего тура

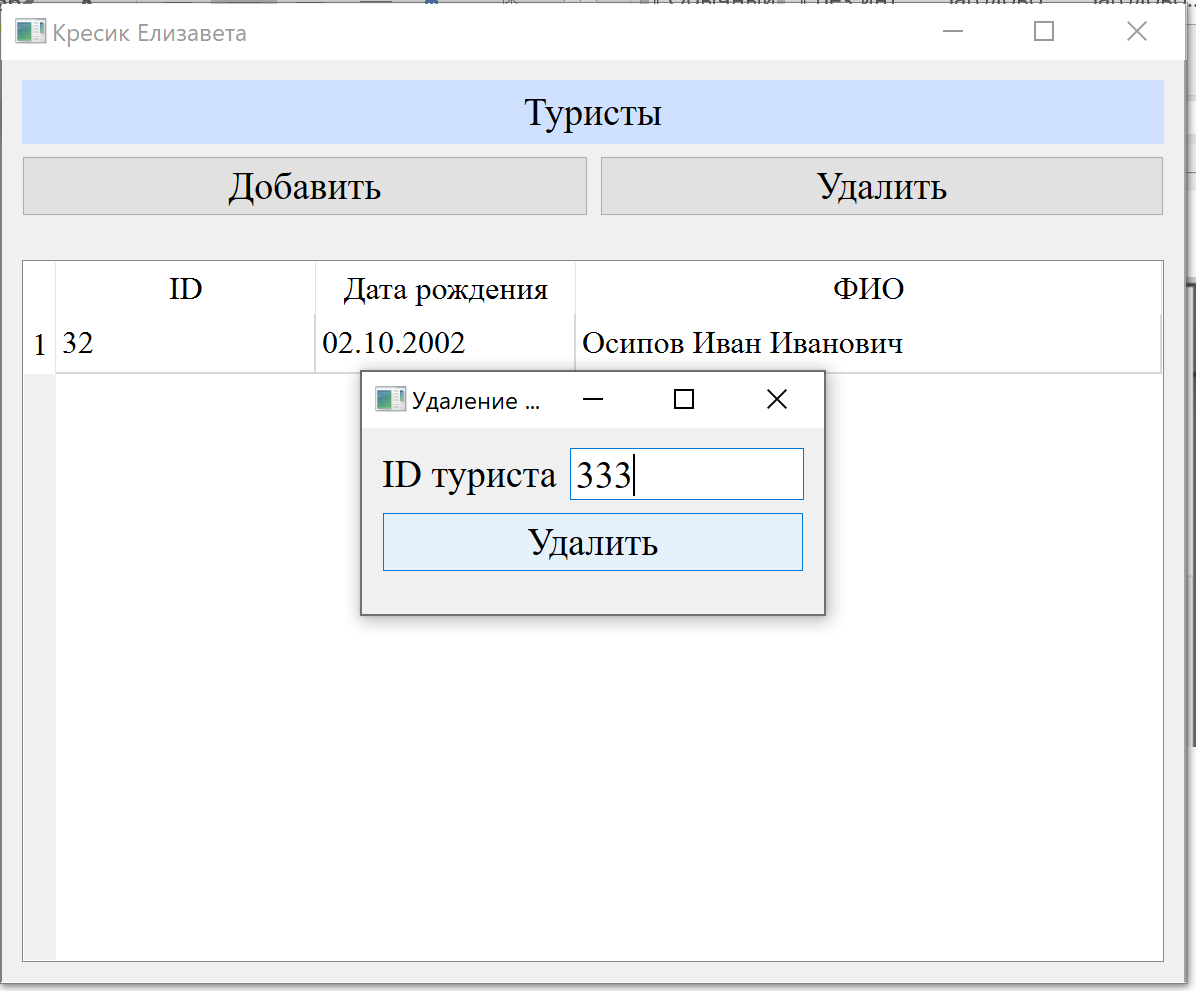


Рисунок 24 – Удаление несуществующего туриста

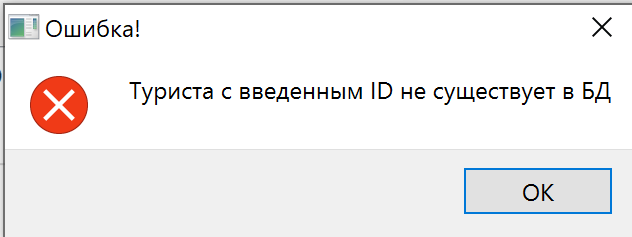


Рисунок 25 – Уведомление об ошибке при удалении несуществующего туриста

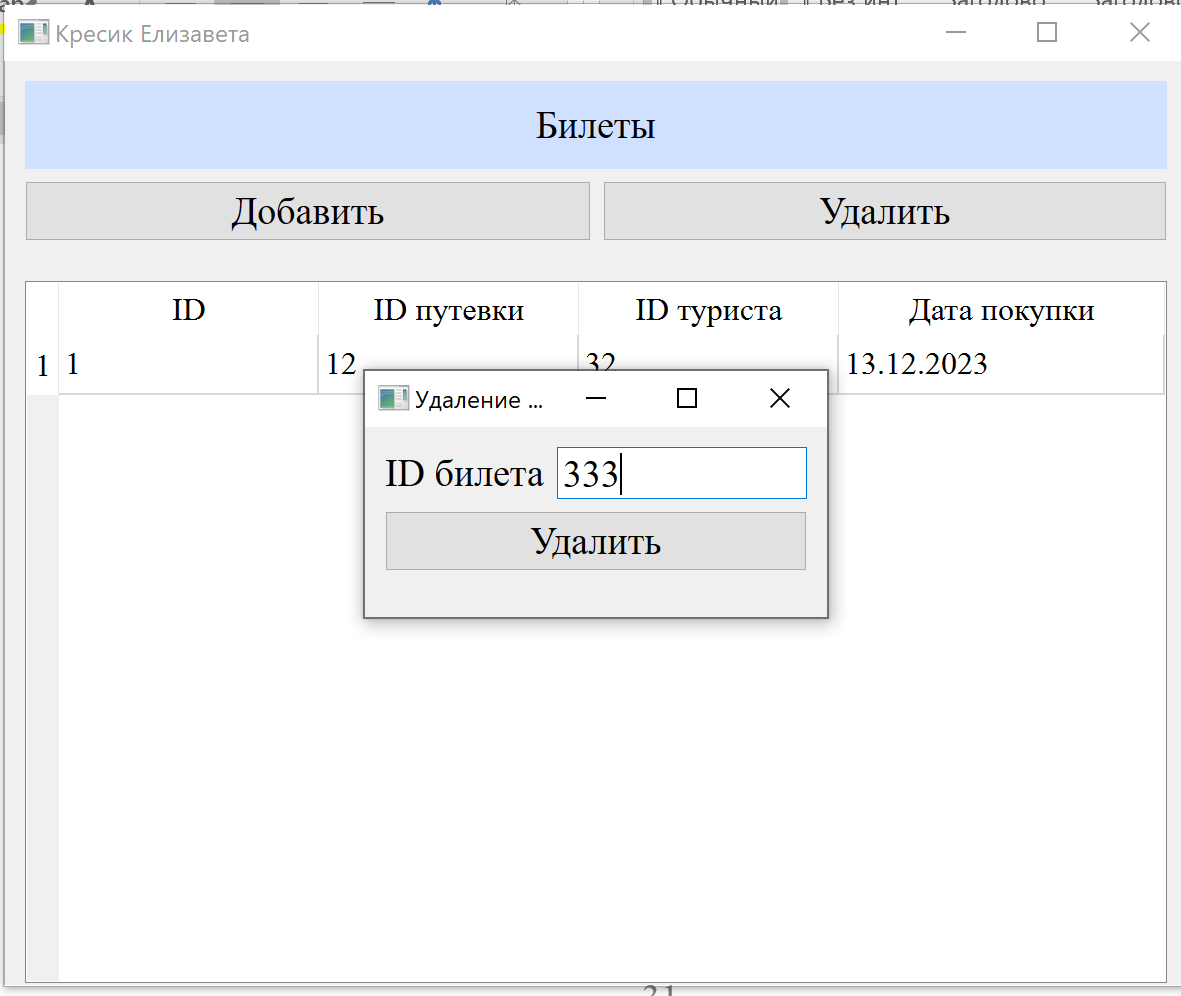


Рисунок 26 – Удаление несуществующего билета

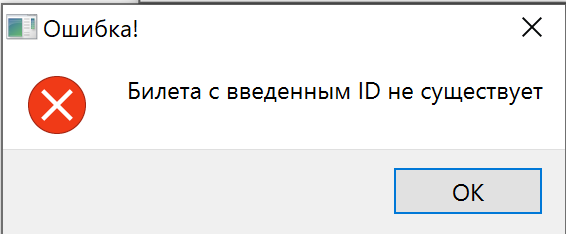


Рисунок 27 – Уведомление об ошибке при удалении несуществующего билета

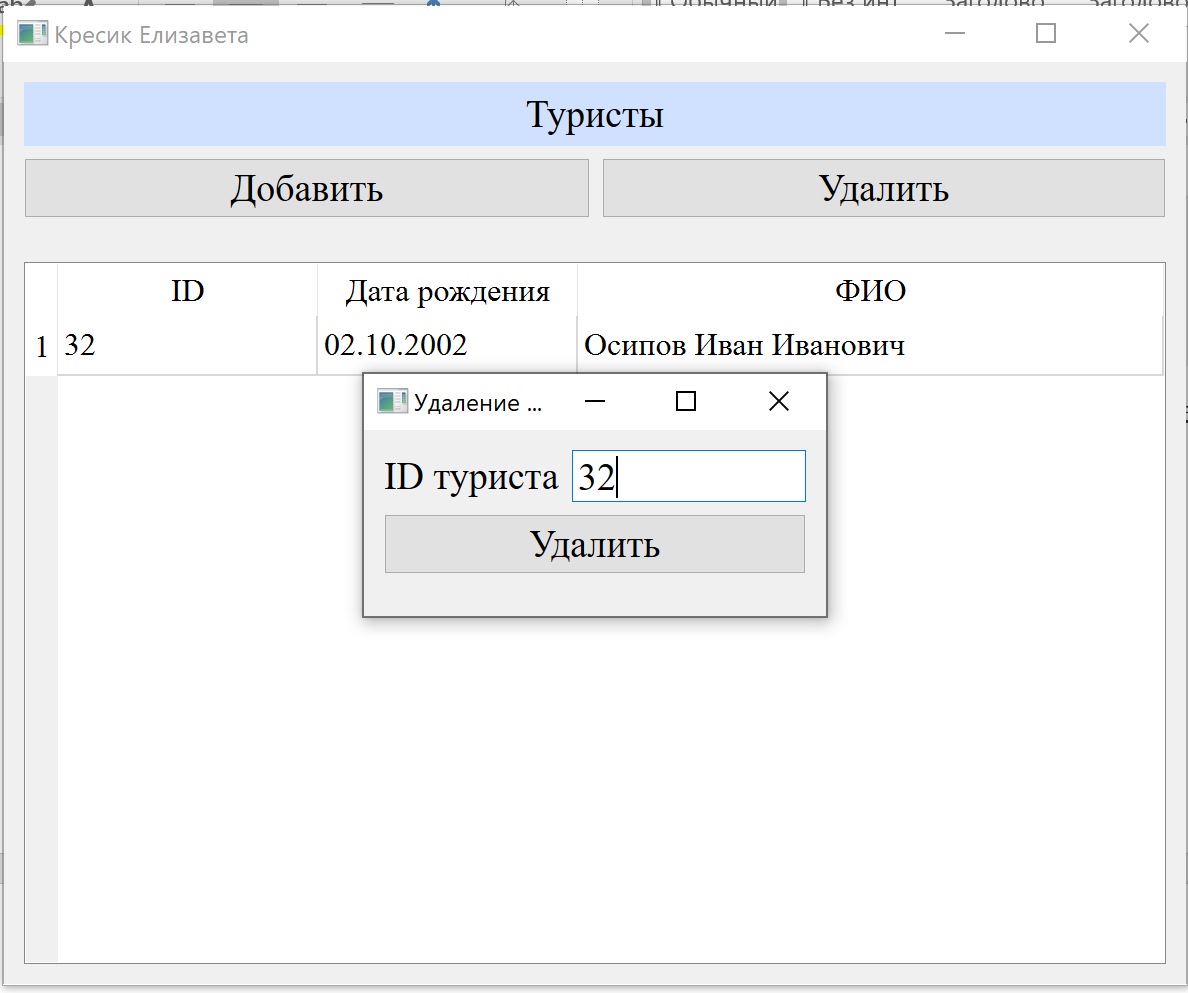


Рисунок 28 – Удаление туриста при наличии купленных на него билетов

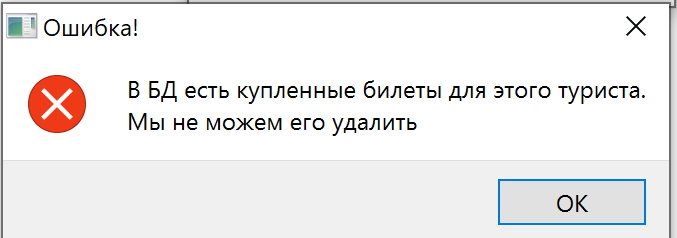


Рисунок 29 – Уведомление об ошибке при удалении туриста с наличием купленных на него билетов

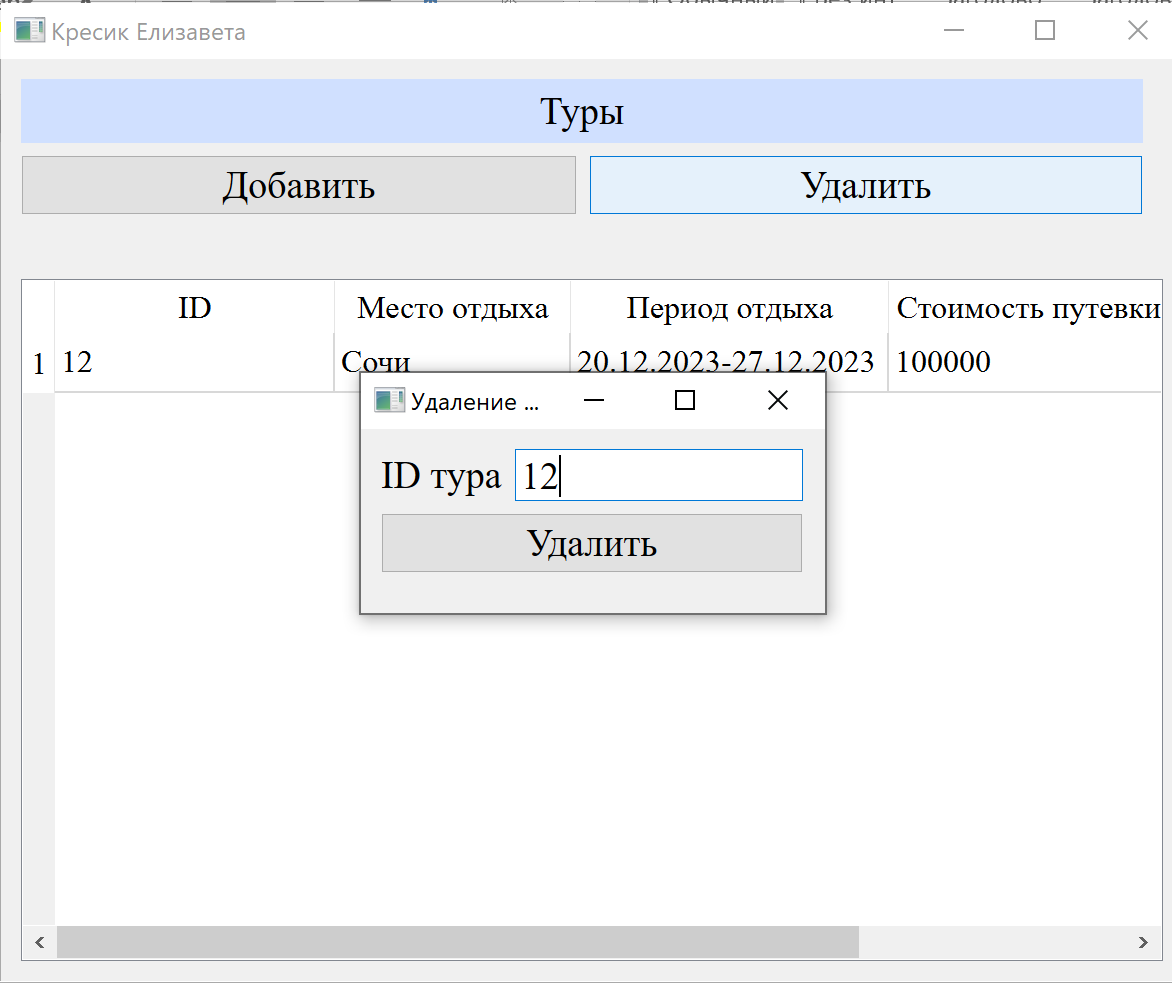


Рисунок 30 – Удаление тура при наличии купленных на него билетов

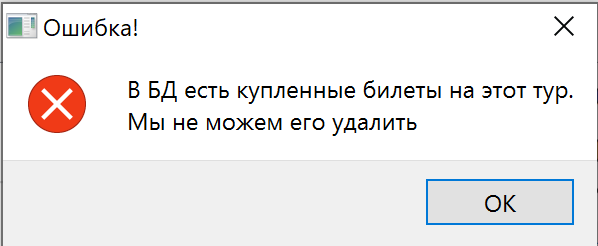


Рисунок 31 – Уведомление об ошибке при удалении тура с наличием купленных на него билетов

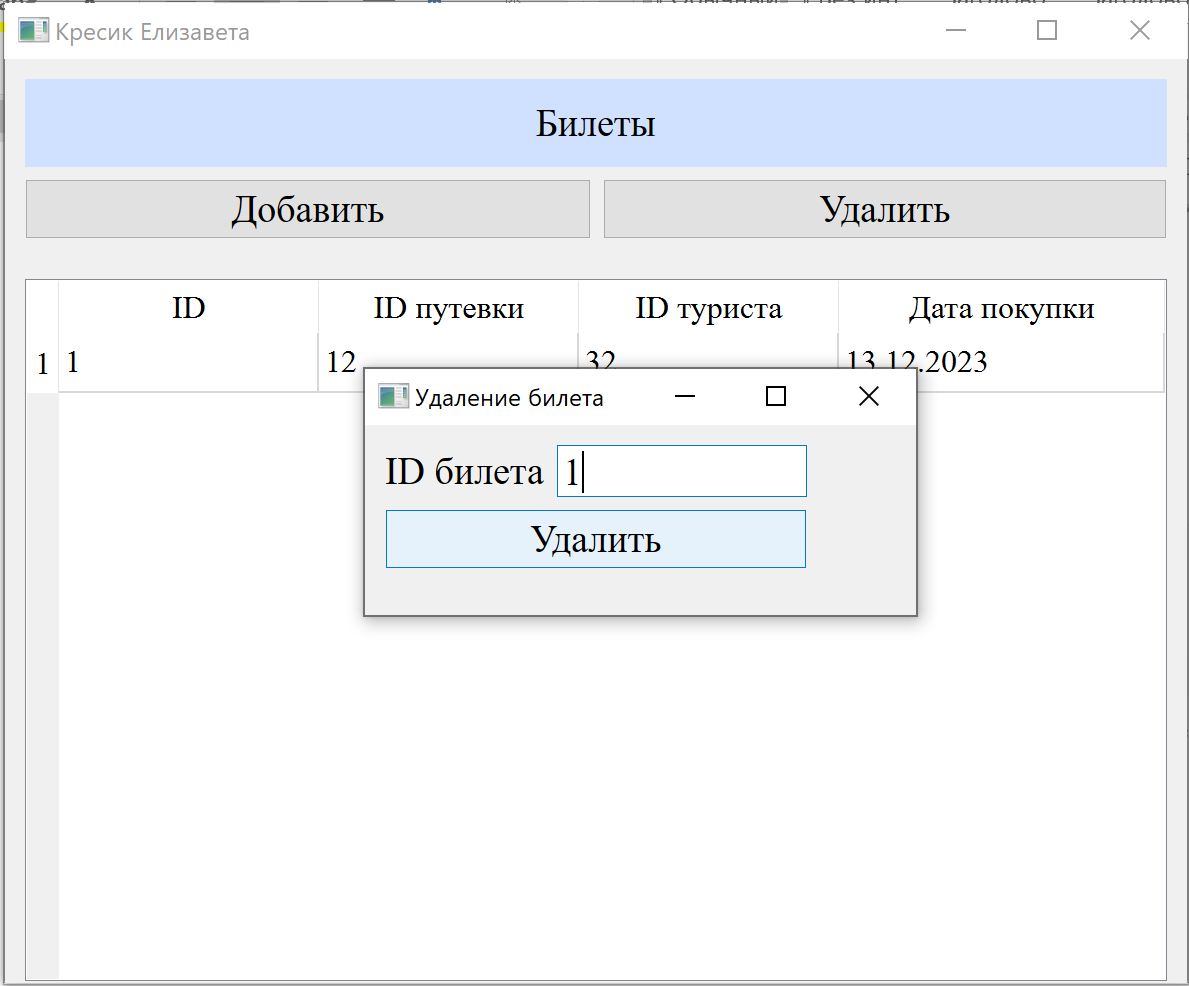


Рисунок 32 – Удаление билета

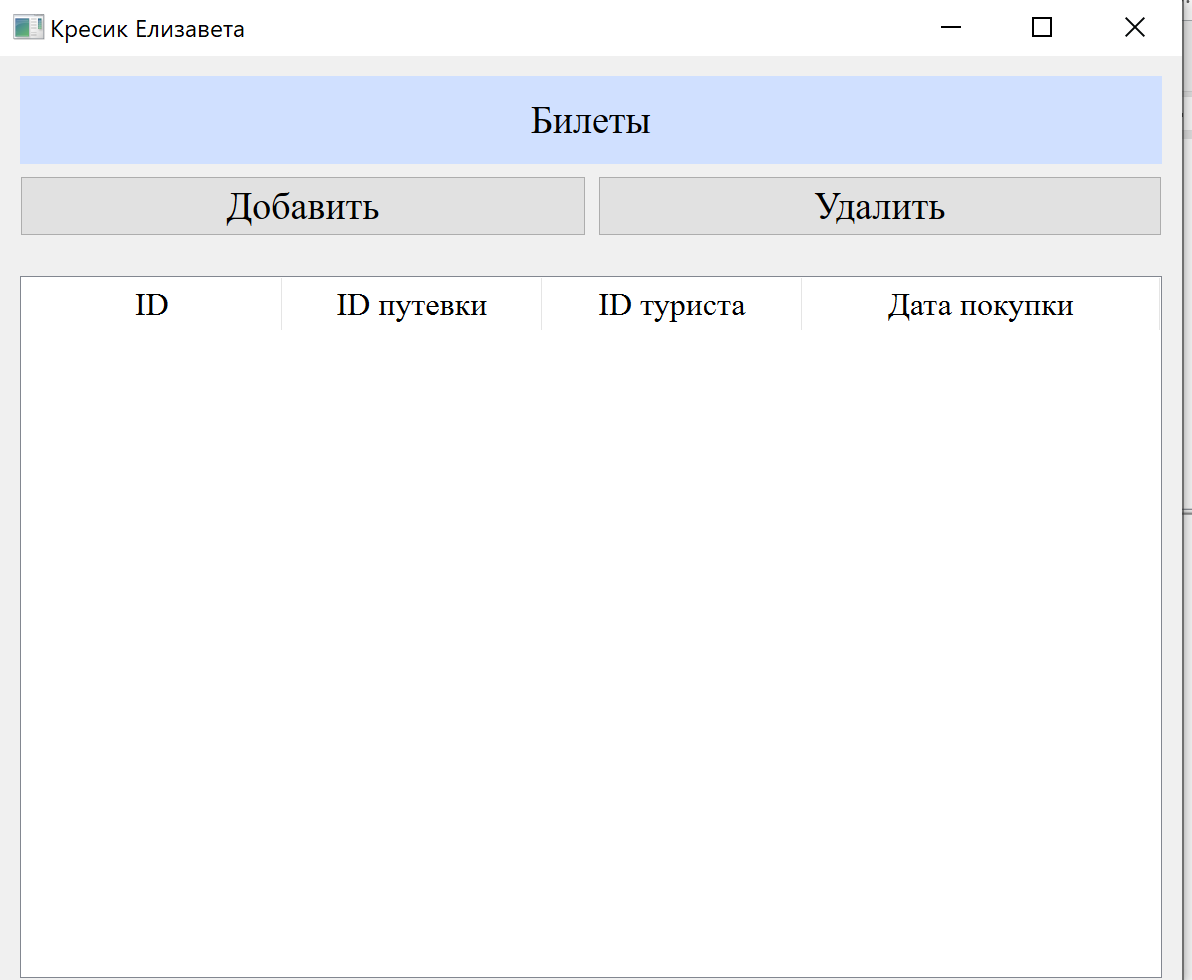


Рисунок 33 – Таблица билетов после удаления

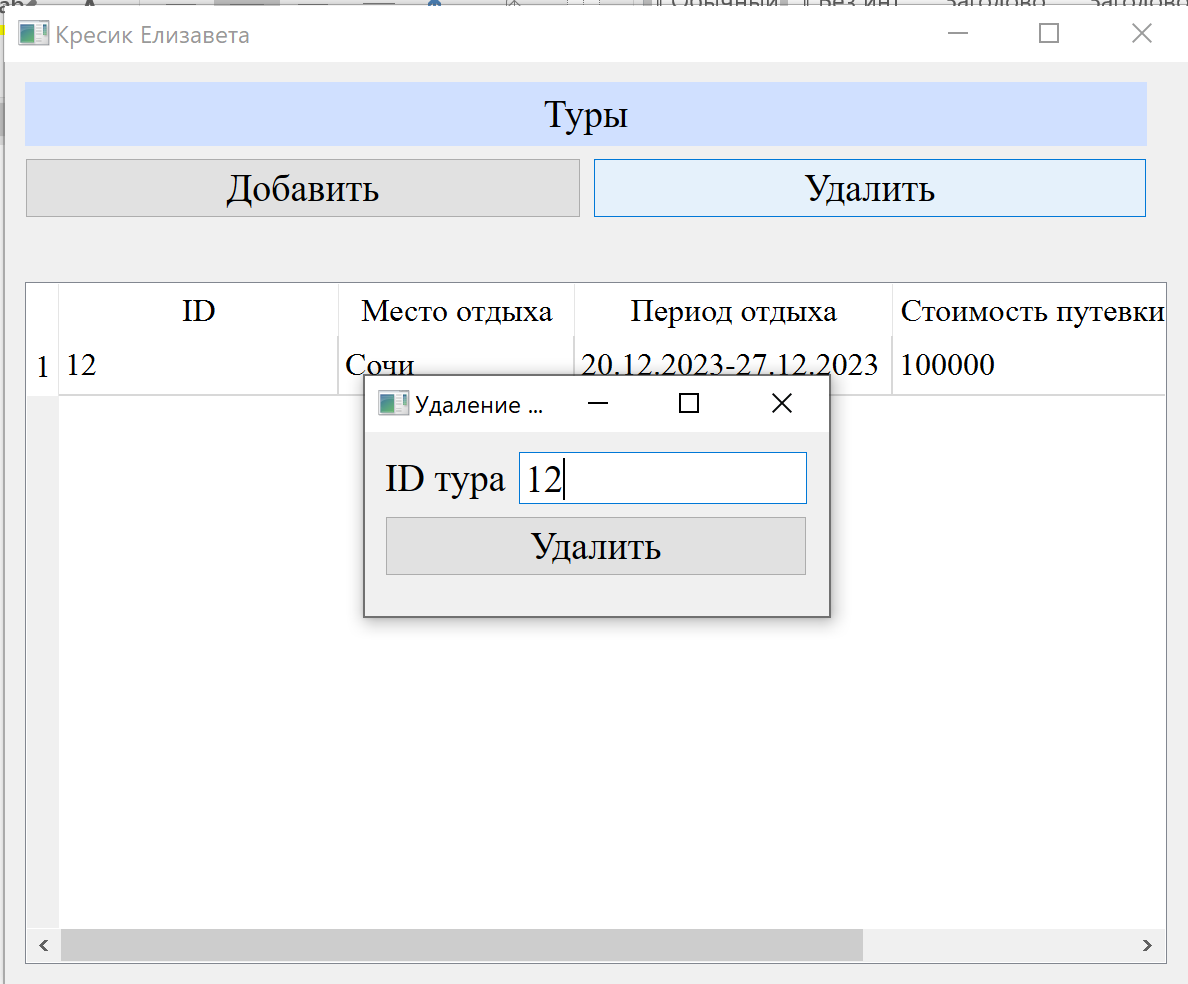


Рисунок 34 – Удаление тура

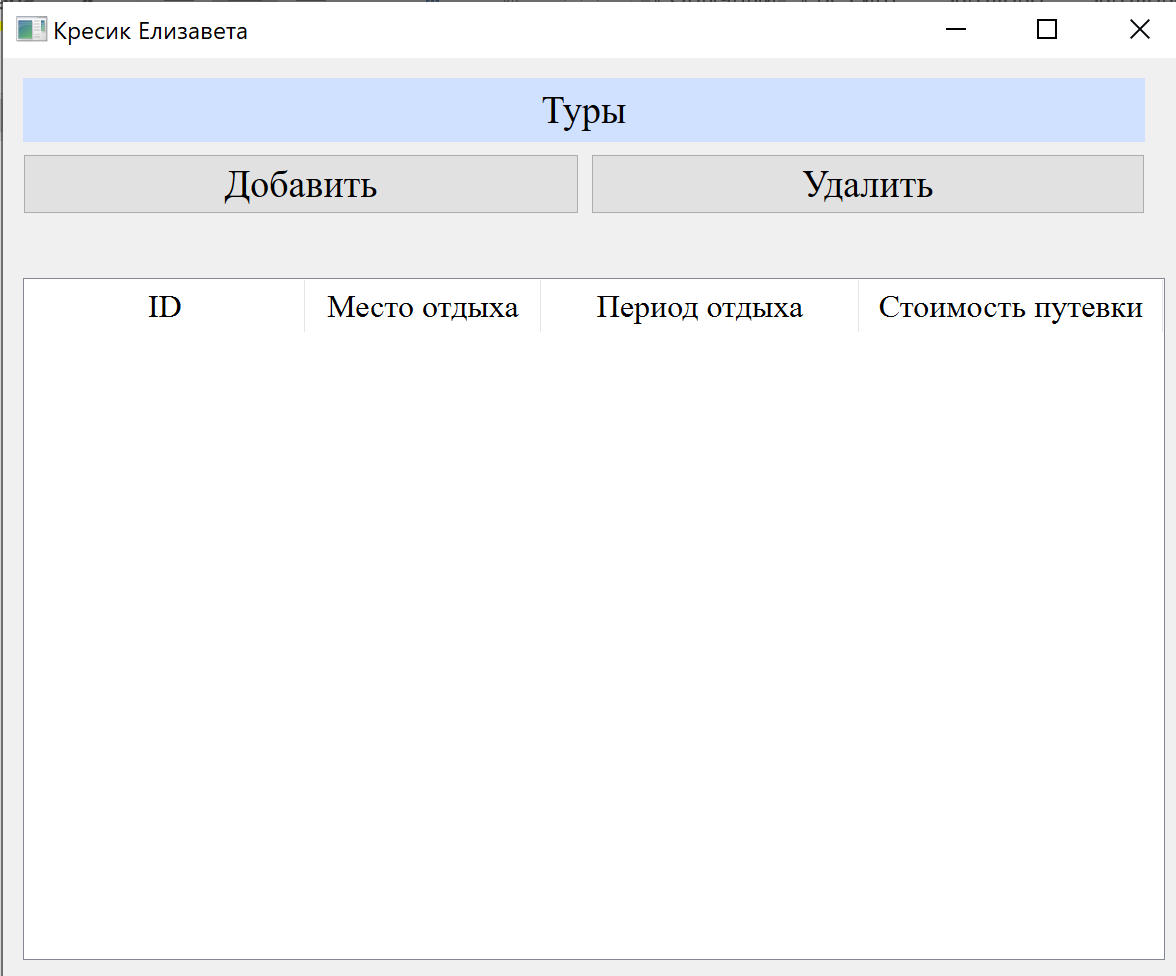


Рисунок 35 – Таблица туров после удаления

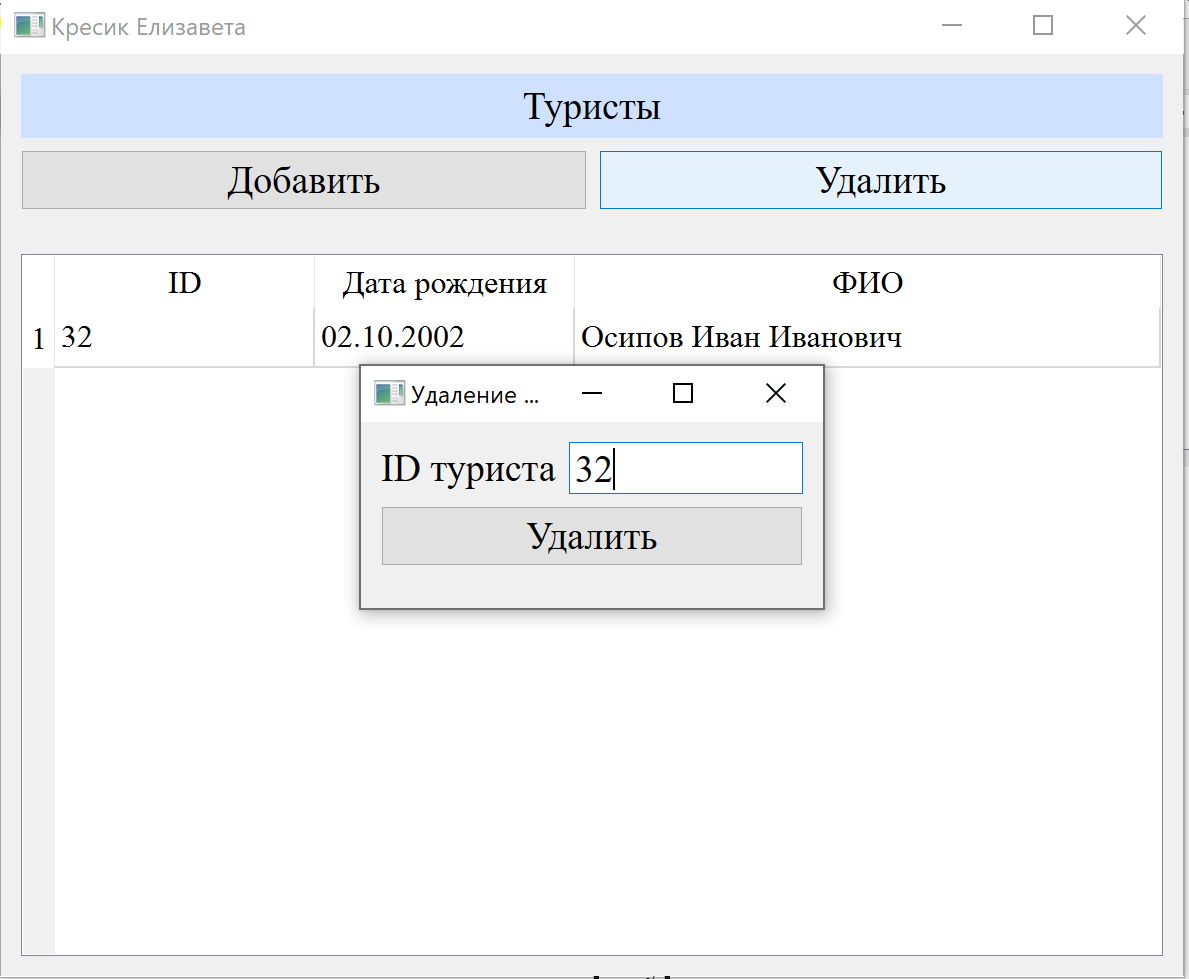


Рисунок 36 – Удаление туриста

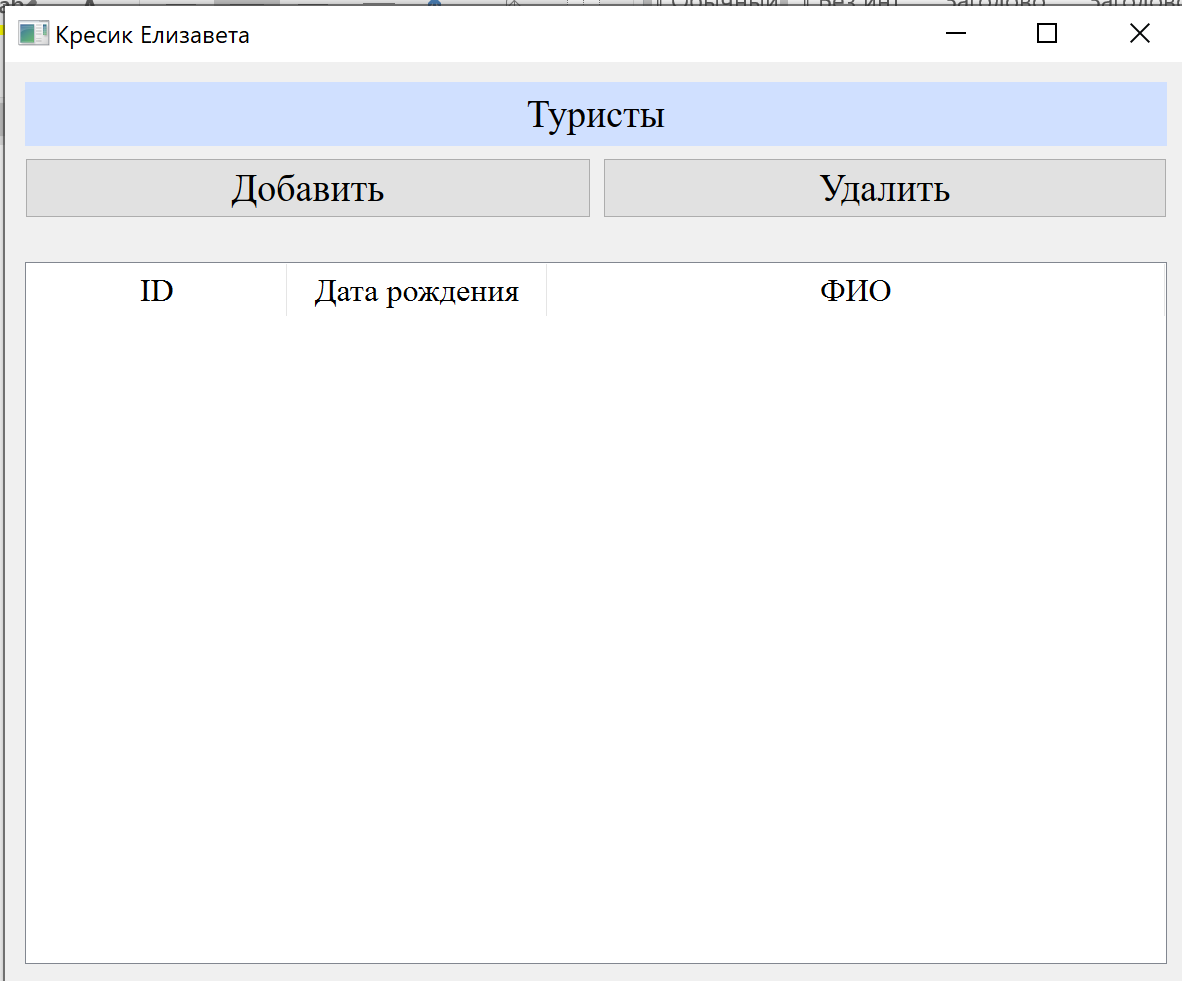


Рисунок 37 – Таблица туристов после удаления

# **Список использованной литературы:**

* Шумова Е. О. Методические указания к выполнению курсового проекта
* Александр Швец Погружение в паттерны проектирования
* Макс Шлее qt 5.10 профессиональное программирование на C++
* [cyberforum.ru](https://vk.com/away.php?to=http%3A%2F%2Fcyberforum.ru&cc_key=)
* [stackoverflow.com](https://vk.com/away.php?to=http%3A%2F%2Fstackoverflow.com&cc_key=)

# Приложение 1

**Код программы:**

**Файл database.h**

#ifndef DATABASE\_H

#define DATABASE\_H

#include <QObject>

#include "ticket.h"

#include "tour.h"

#include "tourist.h"

**class** **DatabasePresent**;

**class** **Database** : **public** QObject

{

Q\_OBJECT

**public:**

**explicit** Database(QObject \*parent = nullptr);

~Database();

**void** **add\_tour**(Tour \*tour);

**void** **add\_tourist**(Tourist \*tourist);

**void** **add\_ticket**(Ticket \*ticket);

**void** **delete\_tour**(**int** ID);

**void** **delete\_tourist**(**int** ID);

**void** **delete\_ticket**(**int** ID);

DatabasePresent \***constructPresent**(**int** i); //Метод для создания объекта класса DatabasePresent

**signals:**

**void** **notifyTourTableChange**(); //уведомляем наблюдателей

**void** **notifyTouristTableChange**(); //уведомляем наблюдателей

**void** **notifyTicketTableChange**(); //уведомляем наблюдателей

**private:**

QVector<Ticket \*> ticket\_array; //объявление коллекции, которая может хранить указатели на элементы класса Ticket

QVector<Tour \*> tour\_array;

QVector<Tourist \*> tourist\_array;

};

#include "databasepresent.h"

#endif // DATABASE\_H

**Файл databasepresent.h**

#ifndef DATABASEPRESENT\_H

#define DATABASEPRESENT\_H

#include <QObject>

#include "database.h"

**class** **DatabasePresent** : **public** QObject

{

Q\_OBJECT

**public:**

**explicit** DatabasePresent(QObject \*parent = nullptr);

DatabasePresent(QVector<Ticket \*> \*ticketArray, QVector<Tour \*> \*tourArray,

QVector<Tourist \*> \*touristArray, Database \*base);

**void** **add\_tour**(Tour \*tour);

**void** **add\_tourist**(Tourist \*tourist);

**void** **add\_ticket**(Ticket \*ticket);

**void** **delete\_tour**(**int** ID);

**void** **delete\_tourist**(**int** ID);

**void** **delete\_ticket**(**int** ID);

**const** QVector<Ticket \*>\* get\_ticket\_array() { **return** ticket\_array; }

**const** QVector<Tour \*>\* get\_tour\_array() { **return** tour\_array; }

**const** QVector<Tourist \*>\* get\_tourist\_array() { **return** tourist\_array; }

**public** slots:

**void** notifyTourTable();

**void** **notifyTouristTable**();

**void** **notifyTicketTable**();

**signals:**

**void** **notifyTourTableChange**(); //уведомляем наблюдателей

**void** **notifyTouristTableChange**(); //уведомляем наблюдателей

**void** **notifyTicketTableChange**(); //уведомляем наблюдателей

**private:**

**const** QVector<Ticket \*>\* ticket\_array;

**const** QVector<Tour \*>\* tour\_array;

**const** QVector<Tourist \*>\* tourist\_array;

Database \*Base; //Указатель на базу данных

};

#endif // DATABASEPRESENT\_H

**Файл mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include "database.h"

#include <QMainWindow>

QT\_BEGIN\_NAMESPACE

**namespace** Ui { **class** **MainWindow**; }

QT\_END\_NAMESPACE

**class** **MainWindow** : **public** QMainWindow

{

Q\_OBJECT

**public:**

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

**private** slots:

**void** on\_openTours\_clicked();

**void** **on\_openTourists\_clicked**();

**void** **on\_openTickets\_clicked**();

**private:**

Ui::MainWindow \*ui;

Database Base; //оригинал базы данных

};

#endif // MAINWINDOW\_H

**Файл ticket.h**

#ifndef TICKET\_H

#define TICKET\_H

#include <QObject>

**class** **Ticket** : **public** QObject

{

Q\_OBJECT

**public:**

**explicit** Ticket(QObject \*parent = nullptr);

Ticket(**int** tourID, **int** touristID, QString datePurchase, **int** ticketID); //Конструктор

~Ticket();

//Для получения значений полей:

QString **getDatePurchase** () { **return** date\_purchase; };

**int** **getTouristID** () { **return** tourist\_ID; };

**int** **getTourID** () { **return** tour\_ID; };

**int** **getTicketID** () { **return** ticket\_ID; };

**private:**

**int** ticket\_ID; //ID билета

**int** tour\_ID; //ID тура

**int** tourist\_ID; //ID туриста

QString date\_purchase; //Дата покупки

};

#endif // TICKET\_H

**Файл ticket\_add\_form.h**

#ifndef TICKET\_ADD\_FORM\_H

#define TICKET\_ADD\_FORM\_H

#include <QWidget>

#include "databasepresent.h"

**namespace** Ui {

**class** **Ticket\_add\_form**;

}

**class** **Ticket\_add\_form** : **public** QWidget

{

Q\_OBJECT

**public:**

**explicit** Ticket\_add\_form(QWidget \*parent = nullptr);

~Ticket\_add\_form();

**void** **setDatabasePresent**(DatabasePresent\*);

**private** slots:

**void** on\_addition\_clicked();

**private:**

Ui::Ticket\_add\_form \*ui;

DatabasePresent\* present;

};

#endif // TICKET\_ADD\_FORM\_H

**Файл ticket\_del\_form.h**

#ifndef TICKET\_DEL\_FORM\_H

#define TICKET\_DEL\_FORM\_H

#include <QWidget>

#include "databasepresent.h"

**namespace** Ui {

**class** **Ticket\_del\_form**;

}

**class** **Ticket\_del\_form** : **public** QWidget

{

Q\_OBJECT

**public:**

**explicit** Ticket\_del\_form(QWidget \*parent = nullptr);

~Ticket\_del\_form();

**void** **setDatabasePresent**(DatabasePresent\*);

**private** slots:

**void** on\_ticket\_delete\_clicked();

**private:**

Ui::Ticket\_del\_form \*ui;

DatabasePresent\* present;

};

#endif // TICKET\_DEL\_FORM\_H

**Файл tickets.h**

#ifndef TICKETS\_H

#define TICKETS\_H

#include <QWidget>

#include "databasepresent.h"

**namespace** Ui {

**class** **Tickets**;

}

**class** **Tickets** : **public** QWidget

{

Q\_OBJECT

**public:**

**explicit** Tickets(QWidget \*parent = nullptr);

~Tickets();

**void** **setDatabasePresent**(DatabasePresent\*);

**public** slots:

**void** updateTicketTable();

**private** slots:

**void** on\_addTicket\_clicked();

**void** **on\_delTicket\_clicked**();

**private:**

Ui::Tickets \*ui;

DatabasePresent\* present = nullptr;

};

#endif // TICKETS\_H

**Файл tour.h**

#ifndef TOUR\_H

#define TOUR\_H

#include <QObject>

**class** **Tour** : **public** QObject

{

Q\_OBJECT

**public:**

**explicit** Tour(QObject \*parent = nullptr);

Tour(QString location, **int** ID, QString time, **int** cost); //Конструктор

~Tour();

//Для получения значений полей:

QString **getLocation** () { **return** tour\_location; };

**int** **getID** () { **return** tour\_ID; };

QString **getTime** () { **return** tour\_time; };

**int** **getCost** () { **return** tour\_cost; };

**private:**

QString tour\_location; //Место отдыха

**int** tour\_ID; //ID тура

QString tour\_time; //Период отдыха

**int** tour\_cost; //Стоимость путевки

};

#endif // TOUR\_H

**Файл tour\_add\_form.h**

#ifndef TOUR\_ADD\_FORM\_H

#define TOUR\_ADD\_FORM\_H

#include <QWidget>

#include "databasepresent.h"

**namespace** Ui {

**class** **Tour\_add\_form**;

}

**class** **Tour\_add\_form** : **public** QWidget

{

Q\_OBJECT

**public:**

**explicit** Tour\_add\_form(QWidget \*parent = nullptr);

~Tour\_add\_form();

**void** **setDatabasePresent**(DatabasePresent\*);

**private** slots:

**void** on\_addition\_clicked();

**private:**

Ui::Tour\_add\_form \*ui;

DatabasePresent\* present;

};

#endif // TOUR\_ADD\_FORM\_H

**Файл tour\_del\_form.h**

#ifndef TOUR\_DEL\_FORM\_H

#define TOUR\_DEL\_FORM\_H

#include <QWidget>

#include "databasepresent.h"

**namespace** Ui {

**class** **Tour\_del\_form**;

}

**class** **Tour\_del\_form** : **public** QWidget

{

Q\_OBJECT

**public:**

**explicit** Tour\_del\_form(QWidget \*parent = nullptr);

~Tour\_del\_form();

**void** **setDatabasePresent**(DatabasePresent\*);

**private** slots:

**void** on\_tour\_delete\_clicked();

**private:**

Ui::Tour\_del\_form \*ui;

DatabasePresent\* present;

};

#endif // TOUR\_DEL\_FORM\_H

**Файл tourist.h**

#ifndef TOURIST\_H

#define TOURIST\_H

#include <QObject>

**class** **Tourist** : **public** QObject

{

Q\_OBJECT

**public:**

**explicit** Tourist(QObject \*parent = nullptr);

Tourist(**int** ID, QString birthday, QString name); //Конструктор

~Tourist();

//Для получения значений полей:

QString **getBirthday** () { **return** tourist\_birthday; }

**int** **getID** () { **return** tourist\_ID; }

QString **getName** () { **return** tourist\_name; }

**private:**

**int** tourist\_ID;

QString tourist\_birthday;

QString tourist\_name;

};

#endif // TOURIST\_H

**Файл tourist\_add\_form.h**

#ifndef TOURIST\_ADD\_FORM\_H

#define TOURIST\_ADD\_FORM\_H

#include <QWidget>

#include "databasepresent.h"

**namespace** Ui {

**class** **Tourist\_add\_form**;

}

**class** **Tourist\_add\_form** : **public** QWidget

{

Q\_OBJECT

**public:**

**explicit** Tourist\_add\_form(QWidget \*parent = nullptr);

~Tourist\_add\_form();

**void** **setDatabasePresent**(DatabasePresent\*);

**private** slots:

**void** on\_addition\_clicked();

**private:**

Ui::Tourist\_add\_form \*ui;

DatabasePresent\* present;

};

#endif // TOURIST\_ADD\_FORM\_H

**Файл tourist\_del\_form.h**

#ifndef TOURIST\_DEL\_FORM\_H

#define TOURIST\_DEL\_FORM\_H

#include <QWidget>

#include "databasepresent.h"

**namespace** Ui {

**class** **Tourist\_del\_form**;

}

**class** **Tourist\_del\_form** : **public** QWidget

{

Q\_OBJECT

**public:**

**explicit** Tourist\_del\_form(QWidget \*parent = nullptr);

~Tourist\_del\_form();

**void** **setDatabasePresent**(DatabasePresent\*);

**private** slots:

**void** on\_tourist\_delete\_clicked();

**private:**

Ui::Tourist\_del\_form \*ui;

DatabasePresent\* present;

};

#endif // TOURIST\_DEL\_FORM\_H

**Файл tourists.h**

#ifndef TOURISTS\_H

#define TOURISTS\_H

#include <QWidget>

#include "databasepresent.h"

**namespace** Ui {

**class** **Tourists**;

}

**class** **Tourists** : **public** QWidget

{

Q\_OBJECT

**public:**

**explicit** Tourists(QWidget \*parent = nullptr);

~Tourists();

**void** **setDatabasePresent**(DatabasePresent\*);

**public** slots:

**void** updateTouristTable();

**private** slots:

**void** on\_addTourist\_clicked();

**void** **on\_delTourist\_clicked**();

**private:**

Ui::Tourists \*ui;

DatabasePresent\* present = nullptr;

};

#endif // TOURISTS\_H

**Файл tours.h**

#ifndef TOURS\_H

#define TOURS\_H

#include <QWidget>

#include "databasepresent.h"

**namespace** Ui {

**class** **Tours**;

}

**class** **Tours** : **public** QWidget

{

Q\_OBJECT

**public:**

**explicit** Tours(QWidget \*parent = nullptr);

~Tours();

**void** **setDatabasePresent**(DatabasePresent\*);

**public** slots:

**void** updateTourTable();

**private** slots:

**void** on\_addTour\_clicked();

**void** **on\_delTour\_clicked**();

**private:**

Ui::Tours\* ui;

DatabasePresent\* present = nullptr;

};

#endif // TOURS\_H

**Файл database.cpp**

#include "database.h"

Database::Database(QObject \*parent)

: QObject{parent}

{

}

Database::~Database()

{

}

**void** Database::add\_tour(Tour \*tour)

{

tour\_array.push\_back(tour);

emit **notifyTourTableChange**();

}

**void** Database::add\_tourist(Tourist \*tourist)

{

tourist\_array.push\_back(tourist);

emit **notifyTouristTableChange**();

}

**void** Database::add\_ticket(Ticket \*ticket)

{

ticket\_array.push\_back(ticket);

emit **notifyTicketTableChange**();

}

**void** Database::delete\_tour(**int** ID)

{

**for** (**auto** it = tour\_array.begin(); it != tour\_array.end(); ++it)

{

**if** ((\*it)->getID() == ID)

{

tour\_array.erase(it);

**break**;

}

}

emit notifyTourTableChange();

}

**void** Database::delete\_tourist(**int** ID)

{

**for** (**auto** it = tourist\_array.begin(); it != tourist\_array.end(); ++it)

{

**if** ((\*it)->getID() == ID)

{

tourist\_array.erase(it);

**break**;

}

}

emit notifyTouristTableChange();

}

**void** Database::delete\_ticket(**int** ID)

{

**for** (**auto** it = ticket\_array.begin(); it != ticket\_array.end(); ++it)

{

**if** ((\*it)->getTicketID() == ID)

{

ticket\_array.erase(it);

**break**;

}

}

emit notifyTicketTableChange();

}

DatabasePresent\* Database::constructPresent(**int** i) //Метод для создания указателя на новый объект класса DatabasePresent

{

DatabasePresent\* present = **new** DatabasePresent(&ticket\_array, &tour\_array, &tourist\_array, **this**);

**if** (i == **1**)

connect(**this**, SIGNAL(notifyTourTableChange()), present, SLOT(notifyTourTable()));

**if** (i == **2**)

connect(**this**, SIGNAL(notifyTouristTableChange()), present, SLOT(notifyTouristTable()));

**if** (i == **3**)

connect(**this**, SIGNAL(notifyTicketTableChange()), present, SLOT(notifyTicketTable()));

**return** present;

}

**Файл databasepresent.cpp**

#include "databasepresent.h"

DatabasePresent::DatabasePresent(QObject \*parent)

: QObject{parent}

{

}

DatabasePresent::DatabasePresent(QVector<Ticket \*> \*ticketArray, QVector<Tour \*> \*tourArray,

QVector<Tourist \*> \*touristArray, Database \*base)

: QObject{nullptr}

{

ticket\_array = ticketArray;

tour\_array = tourArray;

tourist\_array = touristArray;

Base = base;

}

**void** DatabasePresent::notifyTourTable()

{

emit notifyTourTableChange(); //уведомляем наблюдателей

}

**void** DatabasePresent::notifyTouristTable()

{

emit notifyTouristTableChange(); //уведомляем наблюдателей

}

**void** DatabasePresent::notifyTicketTable()

{

emit notifyTicketTableChange(); //уведомляем наблюдателей

}

**void** DatabasePresent::add\_tour(Tour \*tour)

{

Base->add\_tour(tour);

}

**void** DatabasePresent::add\_tourist(Tourist \*tourist)

{

Base->add\_tourist(tourist);

}

**void** DatabasePresent::add\_ticket(Ticket \*ticket)

{

Base->add\_ticket(ticket);

}

**void** DatabasePresent::delete\_tour(**int** ID)

{

Base->delete\_tour(ID);

}

**void** DatabasePresent::delete\_tourist(**int** ID)

{

Base->delete\_tourist(ID);

}

**void** DatabasePresent::delete\_ticket(**int** ID)

{

Base->delete\_ticket(ID);

}

**Файл main.cpp**

#include "mainwindow.h"

#include <QApplication>

**int** **main**(**int** argc, **char** \*argv[])

{

QApplication a(argc, argv);

MainWindow w;

w.show();

**return** a.exec();

}

**Файл mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "tours.h"

#include "tourists.h"

#include "tickets.h"

#include "databasepresent.h"

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(**new** Ui::MainWindow)

, Base(nullptr)

{

ui->setupUi(**this**);

}

MainWindow::~MainWindow()

{

**delete** ui;

}

**void** MainWindow::on\_openTours\_clicked()

{

Tours \*tours = **new** Tours(nullptr);

tours->setDatabasePresent(Base.constructPresent(**1**));

tours->show();

tours->updateTourTable();

}

**void** MainWindow::on\_openTourists\_clicked()

{

Tourists \*tourists = **new** Tourists(nullptr);

tourists->setDatabasePresent(Base.constructPresent(**2**));

tourists->show();

tourists->updateTouristTable();

}

**void** MainWindow::on\_openTickets\_clicked()

{

Tickets \*tickets = **new** Tickets(nullptr);

tickets->setDatabasePresent(Base.constructPresent(**3**));

tickets->show();

tickets->updateTicketTable();

}

**Файл ticket.cpp**

#include "ticket.h"

Ticket::Ticket(QObject \*parent)

: QObject{parent}

{

}

Ticket::Ticket(**int** tourID, **int** touristID, QString datePurchase, **int** ticketID)

{

ticket\_ID = ticketID;

tour\_ID = tourID;

tourist\_ID = touristID;

date\_purchase = datePurchase;

}

Ticket::~Ticket() {}

**Файл ticket\_add\_form.cpp**

#include "ticket\_add\_form.h"

#include "ui\_ticket\_add\_form.h"

#include <QValidator>

#include <QMessageBox>

Ticket\_add\_form::Ticket\_add\_form(QWidget \*parent) :

QWidget(parent),

ui(**new** Ui::Ticket\_add\_form)

{

ui->setupUi(**this**);

ui->tour\_ID\_input->setValidator(**new** QIntValidator);

ui->tourist\_ID\_input->setValidator(**new** QIntValidator);

ui->ticket\_ID\_input->setValidator(**new** QIntValidator);

}

Ticket\_add\_form::~Ticket\_add\_form()

{

**delete** ui;

**delete** present;

}

**void** Ticket\_add\_form::on\_addition\_clicked()

{

//Получаем значения из формы

**int** tourID = ui->tour\_ID\_input->text().toInt();

**int** touristID = ui->tourist\_ID\_input->text().toInt();

**int** ticketID = ui->ticket\_ID\_input->text().toInt();

//Проверка на то, есть ли уже билет с таким ID

**const** QVector<Ticket \*>\* vec\_ticket = present->get\_ticket\_array();

**for** (**auto** it = vec\_ticket->begin(); it != vec\_ticket->end(); ++it)

{

**if** ((\*it)->getTicketID() == ticketID)

{

QMessageBox::critical(**this**, "Ошибка!", "Билет с таким ID уже существует");

**return**;

}

}

**bool** flag\_tour = false;

**const** QVector<Tour \*>\* vec\_tour = present->get\_tour\_array();

**for** (**auto** it = vec\_tour->begin(); it != vec\_tour->end(); ++it)

{

**if** ((\*it)->getID() == tourID)

flag\_tour = true;

}

**bool** flag\_tourist = false;

**const** QVector<Tourist \*>\* vec\_tourist = present->get\_tourist\_array();

**for** (**auto** it = vec\_tourist->begin(); it != vec\_tourist->end(); ++it)

{

**if** ((\*it)->getID() == touristID)

flag\_tourist = true;

}

**if** (flag\_tour && flag\_tourist) //Значит и данный тур и данный турист существуют в базе, и мы можем создать билет

{

Ticket\* ticket = **new** Ticket(tourID, touristID, ui->date\_purchase\_input->text(), ticketID);

present->add\_ticket(ticket);

}

**else**

{

QMessageBox::critical(**this**, "Ошибка!", "ID тура или ID туриста отсутствует в базе");

**return**;

}

//в конце

close();

}

**void** Ticket\_add\_form::setDatabasePresent(DatabasePresent\* pr)

{

present = pr;

}

**Файл ticket\_del\_form.cpp**

#include "ticket\_del\_form.h"

#include "ui\_ticket\_del\_form.h"

#include <QValidator>

#include <QMessageBox>

Ticket\_del\_form::Ticket\_del\_form(QWidget \*parent) :

QWidget(parent),

ui(**new** Ui::Ticket\_del\_form)

{

ui->setupUi(**this**);

ui->ticket\_ID\_input->setValidator(**new** QIntValidator);

}

Ticket\_del\_form::~Ticket\_del\_form()

{

**delete** ui;

**delete** present;

}

**void** Ticket\_del\_form::on\_ticket\_delete\_clicked()

{

//Получаем значения из формы

**int** ticketID = ui->ticket\_ID\_input->text().toInt();

//Проверка на то, есть ли билет с таким ID в БД

**const** QVector<Ticket \*>\* vec\_ticket = present->get\_ticket\_array();

**bool** flag = false;

**for** (**auto** it = vec\_ticket->begin(); it != vec\_ticket->end(); ++it)

{

**if** ((\*it)->getTicketID() == ticketID)

flag = true;

}

**if** (!flag)

{

QMessageBox::critical(**this**, "Ошибка!", "Билета с введенным ID не существует");

close();

**return**;

}

present->delete\_ticket(ticketID);

close();

}

**void** Ticket\_del\_form::setDatabasePresent(DatabasePresent\* pr)

{

present = pr;

}

**Файл tickets.cpp**

#include "tickets.h"

#include "ui\_tickets.h"

#include "ticket\_add\_form.h"

#include "ticket\_del\_form.h"

Tickets::Tickets(QWidget \*parent) :

QWidget(parent),

ui(**new** Ui::Tickets)

{

ui->setupUi(**this**);

}

Tickets::~Tickets()

{

**delete** ui;

**delete** present;

}

**void** Tickets::on\_addTicket\_clicked()

{

Ticket\_add\_form \*add\_form = **new** Ticket\_add\_form(nullptr);

add\_form->setDatabasePresent(present);

add\_form->show();

}

**void** Tickets::on\_delTicket\_clicked()

{

Ticket\_del\_form \*del\_form = **new** Ticket\_del\_form(nullptr);

del\_form->setDatabasePresent(present);

del\_form->show();

}

**void** Tickets::setDatabasePresent(DatabasePresent\* pr)

{

**if** (present)

{

disconnect(present, SIGNAL(notifyTicketTableChange()), **this**, SLOT(updateTicketTable()));

}

present = pr;

connect(present, SIGNAL(notifyTicketTableChange()), **this**, SLOT(updateTicketTable()));

}

**void** Tickets::updateTicketTable()

{

ui->ticketTable->setRowCount(**0**); //Очистить строки

**const** QVector<Ticket \*>\* vec\_ticket = present->get\_ticket\_array();

ui->ticketTable->setEditTriggers(QTableWidget::NoEditTriggers);

**for** (**int** i = **0**; i < vec\_ticket->size(); i++)

{

ui->ticketTable->insertRow(i);

ui->ticketTable->setItem(i, **0**, **new** QTableWidgetItem(QString::number(vec\_ticket->at(i)->getTicketID())));

ui->ticketTable->setItem(i, **1**, **new** QTableWidgetItem(QString::number(vec\_ticket->at(i)->getTourID())));

ui->ticketTable->setItem(i, **2**, **new** QTableWidgetItem(QString::number(vec\_ticket->at(i)->getTouristID())));

ui->ticketTable->setItem(i, **3**, **new** QTableWidgetItem(vec\_ticket->at(i)->getDatePurchase()));

}

}

**Файл tour.cpp**

#include "tour.h"

Tour::Tour(QObject \*parent)

: QObject{parent}

{

}

Tour::Tour(QString location, **int** ID, QString time, **int** cost)

{

tour\_location = location;

tour\_ID = ID;

tour\_time = time;

tour\_cost = cost;

}

Tour::~Tour() {}

**Файл tour\_add\_form.cpp**

#include "tour\_add\_form.h"

#include "ui\_tour\_add\_form.h"

#include <QValidator>

#include <QMessageBox>

Tour\_add\_form::Tour\_add\_form(QWidget \*parent) :

QWidget(parent),

ui(**new** Ui::Tour\_add\_form)

{

ui->setupUi(**this**);

ui->tour\_ID\_input->setValidator(**new** QIntValidator);

ui->tour\_cost\_input->setValidator(**new** QIntValidator);

}

Tour\_add\_form::~Tour\_add\_form()

{

**delete** ui;

**delete** present;

}

**void** Tour\_add\_form::on\_addition\_clicked()

{

//Получаем значения из формы

**int** tourID = ui->tour\_ID\_input->text().toInt();

//Проверка на то, есть ли уже тур с таким ID

**const** QVector<Tour \*>\* vec\_tour = present->get\_tour\_array();

**for** (**auto** it = vec\_tour->begin(); it != vec\_tour->end(); ++it)

{

**if** ((\*it)->getID() == tourID)

{

QMessageBox::critical(**this**, "Ошибка!", "Тур с таким ID уже существует");

**return**;

}

}

Tour\* tour = **new** Tour(ui->tour\_location\_input->text(), tourID, ui->tour\_time\_input->text(),

ui->tour\_cost\_input->text().toInt());

present->add\_tour(tour);

//в конце

close();

}

**void** Tour\_add\_form::setDatabasePresent(DatabasePresent\* pr)

{

present = pr;

}

**Файл tour\_del\_form.cpp**

#include "tour\_del\_form.h"

#include "ui\_tour\_del\_form.h"

#include <QValidator>

#include <QMessageBox>

Tour\_del\_form::Tour\_del\_form(QWidget \*parent) :

QWidget(parent),

ui(**new** Ui::Tour\_del\_form)

{

ui->setupUi(**this**);

ui->tour\_ID\_input->setValidator(**new** QIntValidator);

}

Tour\_del\_form::~Tour\_del\_form()

{

**delete** ui;

**delete** present;

}

**void** Tour\_del\_form::on\_tour\_delete\_clicked()

{

//Получаем значения из формы

**int** tourID = ui->tour\_ID\_input->text().toInt();

//Проверка на то, есть ли тур с таким ID в БД

**const** QVector<Tour \*>\* vec\_tour = present->get\_tour\_array();

**bool** flag = false;

**for** (**auto** it = vec\_tour->begin(); it != vec\_tour->end(); ++it)

{

**if** ((\*it)->getID() == tourID)

flag = true;

}

**if** (!flag)

{

QMessageBox::critical(**this**, "Ошибка!", "Тура с введенным ID не существует");

close();

**return**;

}

//Проверка на то, есть ли этот тур в купленных билетах

**const** QVector<Ticket \*>\* vec\_ticket = present->get\_ticket\_array();

flag = false;

**for** (**auto** it = vec\_ticket->begin(); it != vec\_ticket->end(); ++it)

{

**if** ((\*it)->getTourID() == tourID)

flag = true;

}

**if** (flag)

{

QMessageBox::critical(**this**, "Ошибка!", "В БД есть купленные билеты на этот тур.**\n**Мы не можем его удалить");

close();

**return**;

}

present->delete\_tour(tourID);

close();

}

**void** Tour\_del\_form::setDatabasePresent(DatabasePresent\* pr)

{

present = pr;

}

**Файл tourist.cpp**

#include "tourist.h"

Tourist::Tourist(QObject \*parent)

: QObject{parent}

{

}

Tourist::Tourist(**int** ID, QString birthday, QString name)

{

tourist\_ID = ID;

tourist\_birthday = birthday;

tourist\_name = name;

}

Tourist::~Tourist() {}

**Файл tourist\_add\_form.cpp**

#include "tourist\_add\_form.h"

#include "ui\_tourist\_add\_form.h"

#include <QValidator>

#include <QMessageBox>

Tourist\_add\_form::Tourist\_add\_form(QWidget \*parent) :

QWidget(parent),

ui(**new** Ui::Tourist\_add\_form)

{

ui->setupUi(**this**);

ui->tourist\_ID\_input->setValidator(**new** QIntValidator);

}

Tourist\_add\_form::~Tourist\_add\_form()

{

**delete** ui;

**delete** present;

}

**void** Tourist\_add\_form::on\_addition\_clicked()

{

//Получаем значения из формы

**int** touristID = ui->tourist\_ID\_input->text().toInt();

//Проверка на то, есть ли уже турист с таким ID

**const** QVector<Tourist \*>\* vec\_tourist = present->get\_tourist\_array();

**for** (**auto** it = vec\_tourist->begin(); it != vec\_tourist->end(); ++it)

{

**if** ((\*it)->getID() == touristID)

{

QMessageBox::critical(**this**, "Ошибка!", "Турист с таким ID уже существует");

**return**;

}

}

Tourist\* tourist = **new** Tourist(touristID, ui->tourist\_birthday\_input->text(), ui->tourist\_name\_input->text());

present->add\_tourist(tourist);

//в конце

close();

}

**void** Tourist\_add\_form::setDatabasePresent(DatabasePresent\* pr)

{

present = pr;

}

**Файл tourist\_del\_form.cpp**

#include "tourist\_del\_form.h"

#include "ui\_tourist\_del\_form.h"

#include <QValidator>

#include <QMessageBox>

Tourist\_del\_form::Tourist\_del\_form(QWidget \*parent) :

QWidget(parent),

ui(**new** Ui::Tourist\_del\_form)

{

ui->setupUi(**this**);

ui->tourist\_ID\_input->setValidator(**new** QIntValidator);

}

Tourist\_del\_form::~Tourist\_del\_form()

{

**delete** ui;

**delete** present;

}

**void** Tourist\_del\_form::on\_tourist\_delete\_clicked()

{

//Получаем значения из формы

**int** touristID = ui->tourist\_ID\_input->text().toInt();

//Проверка на то, есть ли турист с таким ID в БД

**const** QVector<Tourist \*>\* vec\_tourist = present->get\_tourist\_array();

**bool** flag = false;

**for** (**auto** it = vec\_tourist->begin(); it != vec\_tourist->end(); ++it)

{

**if** ((\*it)->getID() == touristID)

flag = true;

}

**if** (!flag)

{

QMessageBox::critical(**this**, "Ошибка!", "Туриста с введенным ID не существует в БД");

close();

**return**;

}

//Проверка на то, есть ли этот турист в купленных билетах

**const** QVector<Ticket \*>\* vec\_ticket = present->get\_ticket\_array();

flag = false;

**for** (**auto** it = vec\_ticket->begin(); it != vec\_ticket->end(); ++it)

{

**if** ((\*it)->getTouristID() == touristID)

flag = true;

}

**if** (flag)

{

QMessageBox::critical(**this**, "Ошибка!", "В БД есть купленные билеты для этого туриста.**\n**Мы не можем его удалить");

close();

**return**;

}

present->delete\_tourist(touristID);

close();

}

**void** Tourist\_del\_form::setDatabasePresent(DatabasePresent\* pr)

{

present = pr;

}

**Файл tourists.cpp**

#include "tourists.h"

#include "ui\_tourists.h"

#include "tourist\_add\_form.h"

#include "tourist\_del\_form.h"

Tourists::Tourists(QWidget \*parent) :

QWidget(parent),

ui(**new** Ui::Tourists)

{

ui->setupUi(**this**);

}

Tourists::~Tourists()

{

**delete** ui;

**delete** present;

}

**void** Tourists::on\_addTourist\_clicked()

{

Tourist\_add\_form \*add\_form = **new** Tourist\_add\_form(nullptr);

add\_form->setDatabasePresent(present);

add\_form->show();

}

**void** Tourists::on\_delTourist\_clicked()

{

Tourist\_del\_form \*del\_form = **new** Tourist\_del\_form(nullptr);

del\_form->setDatabasePresent(present);

del\_form->show();

}

**void** Tourists::setDatabasePresent(DatabasePresent\* pr)

{

**if** (present)

{

disconnect(present, SIGNAL(notifyTouristTableChange()), **this**, SLOT(updateTouristTable()));

}

present = pr;

connect(present, SIGNAL(notifyTouristTableChange()), **this**, SLOT(updateTouristTable()));

}

**void** Tourists::updateTouristTable()

{

ui->touristTable->setRowCount(**0**); //Очистить строки

**const** QVector<Tourist \*>\* vec\_tourist = present->get\_tourist\_array();

ui->touristTable->setEditTriggers(QTableWidget::NoEditTriggers);

**for** (**int** i = **0**; i < vec\_tourist->size(); i++)

{

ui->touristTable->insertRow(i);

ui->touristTable->setItem(i, **0**, **new** QTableWidgetItem(QString::number(vec\_tourist->at(i)->getID())));

ui->touristTable->setItem(i, **1**, **new** QTableWidgetItem(vec\_tourist->at(i)->getBirthday()));

ui->touristTable->setItem(i, **2**, **new** QTableWidgetItem(vec\_tourist->at(i)->getName()));

}

}

**Файл tours.cpp**

#include "tours.h"

#include "ui\_tours.h"

#include "tour\_add\_form.h"

#include "tour\_del\_form.h"

Tours::Tours(QWidget \*parent) :

QWidget(parent),

ui(**new** Ui::Tours)

{

ui->setupUi(**this**);

// регистрация слушателей

//

}

Tours::~Tours()

{

**delete** ui;

**delete** present;

}

**void** Tours::updateTourTable()

{

ui->tourTable->setRowCount(**0**); //Очистить строки

**const** QVector<Tour \*>\* vec\_tour = present->get\_tour\_array();

ui->tourTable->setEditTriggers(QTableWidget::NoEditTriggers);

**for** (**int** i = **0**; i < vec\_tour->size(); i++)

{

ui->tourTable->insertRow(i);

ui->tourTable->setItem(i, **0**, **new** QTableWidgetItem(QString::number(vec\_tour->at(i)->getID())));

ui->tourTable->setItem(i, **1**, **new** QTableWidgetItem(vec\_tour->at(i)->getLocation()));

ui->tourTable->setItem(i, **2**, **new** QTableWidgetItem(vec\_tour->at(i)->getTime()));

ui->tourTable->setItem(i, **3**, **new** QTableWidgetItem(QString::number(vec\_tour->at(i)->getCost())));

}

}

**void** Tours::on\_addTour\_clicked()

{

Tour\_add\_form \*add\_form = **new** Tour\_add\_form(nullptr);

add\_form->setDatabasePresent(present);

add\_form->show();

}

**void** Tours::on\_delTour\_clicked()

{

Tour\_del\_form \*del\_form = **new** Tour\_del\_form(nullptr);

del\_form->setDatabasePresent(present);

del\_form->show();

}

**void** Tours::setDatabasePresent(DatabasePresent\* pr)

{

**if** (present)

{

disconnect(present, SIGNAL(notifyTourTableChange()), **this**, SLOT(updateTourTable()));

}

present = pr;

connect(present, SIGNAL(notifyTourTableChange()), **this**, SLOT(updateTourTable()));

}