

**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

должность, уч. степень, звание

подпись, дата

Шумова Е. О.

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

подпись, дата

Кресик Е.А.

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
1.1 Анализ предметной области:	4
1.2 Формулировка технического задания:	4
2. Разработка классов:	5
2.1 Классы сущностей:	5
2.2 Интерфейсные классы:	5
2.3 Управляющие классы:	5
2.4 Классы-хранилища:	6
2.5 Используемые паттерны и их UML-диаграммы:	6
2.6 Полная диаграмма классов:	8
3. Разработка приложения:	9
3.1 Разработка интерфейса:	9
3.2 Реализация методов класса:	19
4. Тестирование приложения	27
Список использованной литературы:	45
Приложение 1	46

1. Постановка задачи:

1.1 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

1.2 Формулировка технического задания:

Необходимо разработать и реализовать систему классов «Продажа билетов»

Система должна быть способна реализовывать следующие задачи:

1. Работа с билетами
 - a. Добавление билета
 - b. Удаление билета
 - c. Просмотр билетов
2. Работа с турами
 - a. Добавление тура

- b. Удаление тура
- c. Просмотр туров
- 3. Работа с туристами
 - a. Добавление туриста
 - b. Удаление туриста
 - c. Просмотр туристов

2. Разработка классов:

2.1 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)

2.2 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

2.3 Управляющие классы:

MainWindow — класс отвечает за глобальное взаимодействие сущностей.

2.4Классы-хранилища:

Database — класс, в котором хранятся классы сущностей.

2.5Используемые паттерны и их UML-диаграммы:

- Паттерн Наблюдатель:

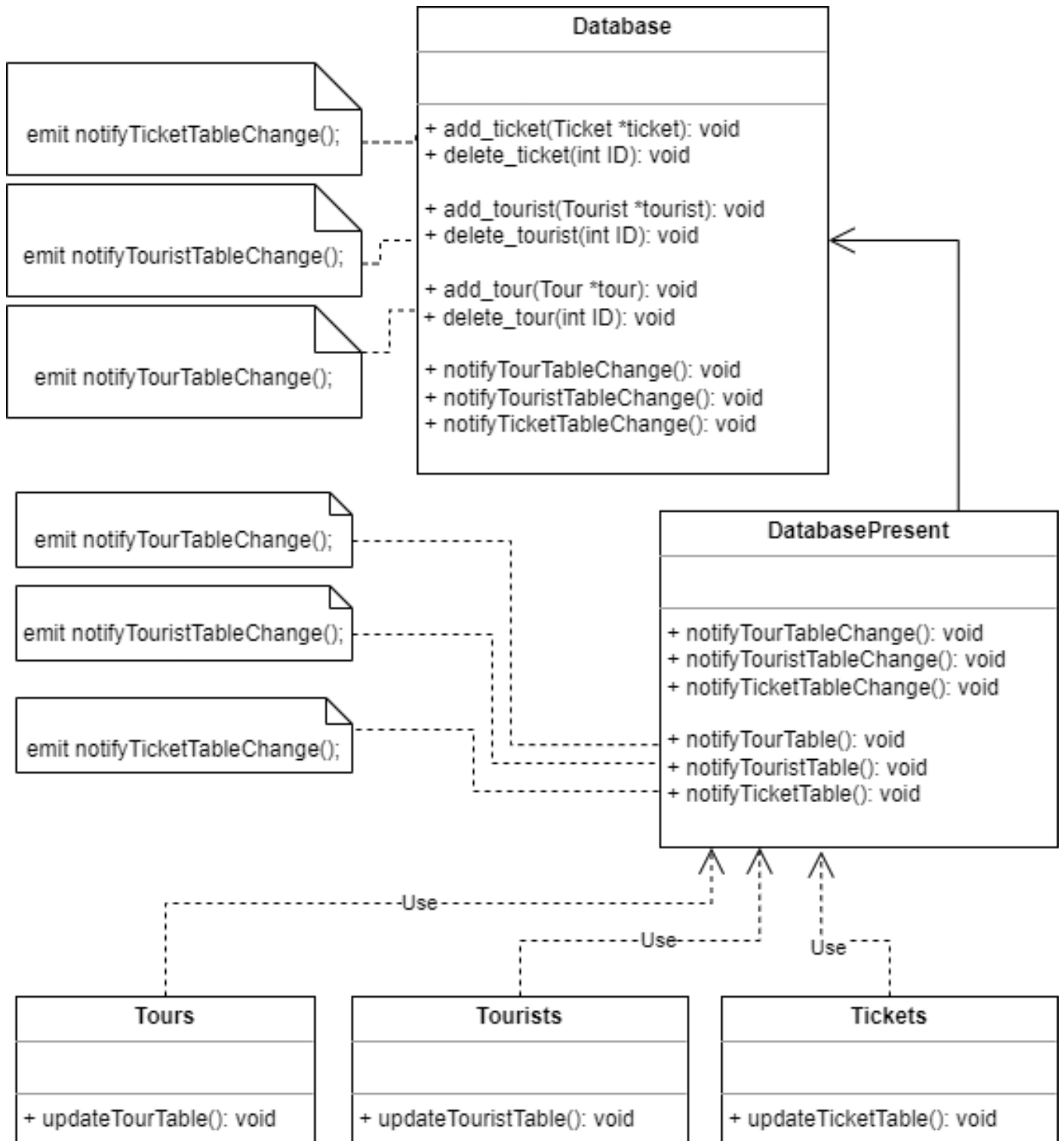
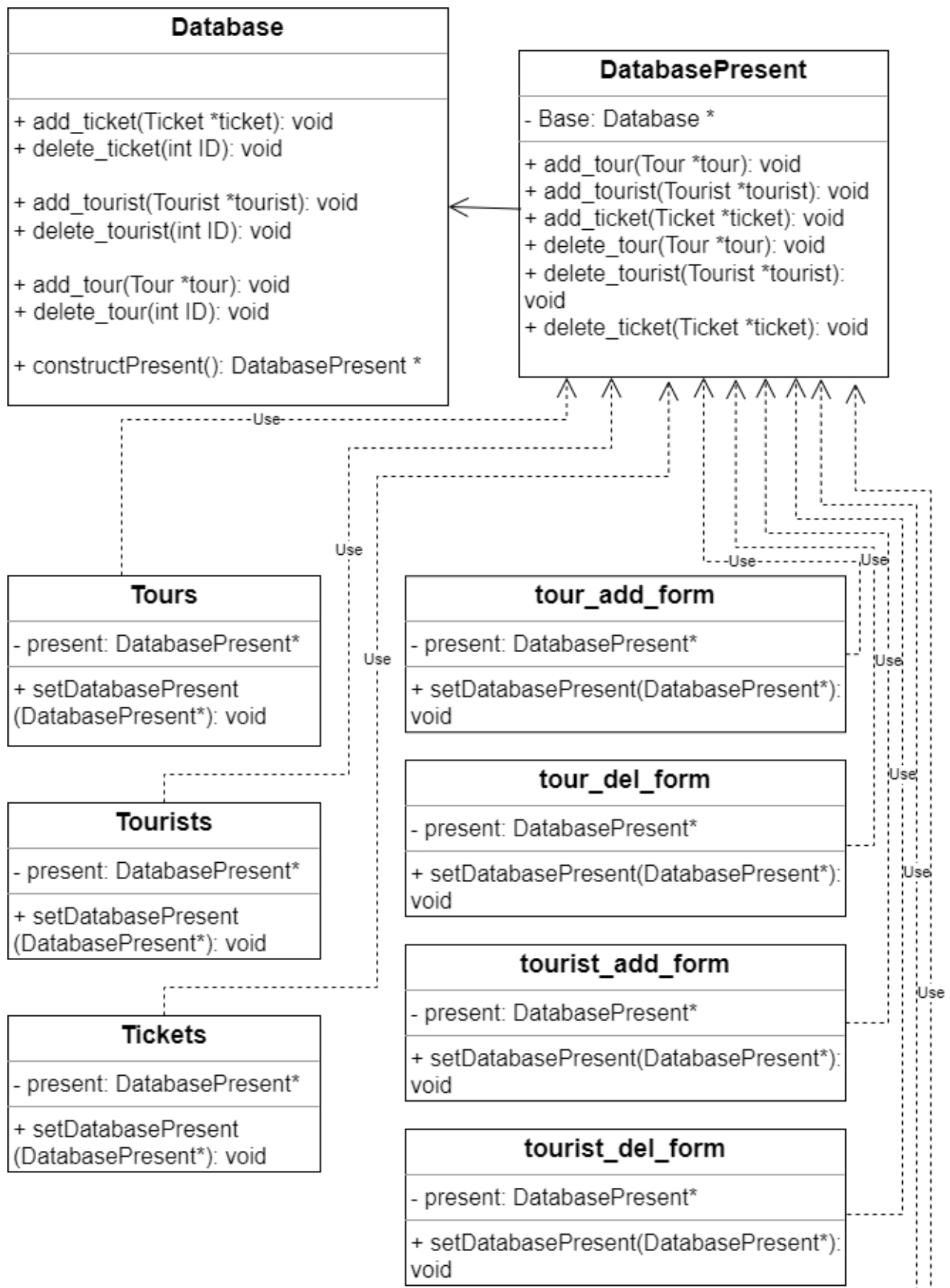


Рисунок 1 – Диаграмма паттерна Наблюдатель

- Паттерн Посредник



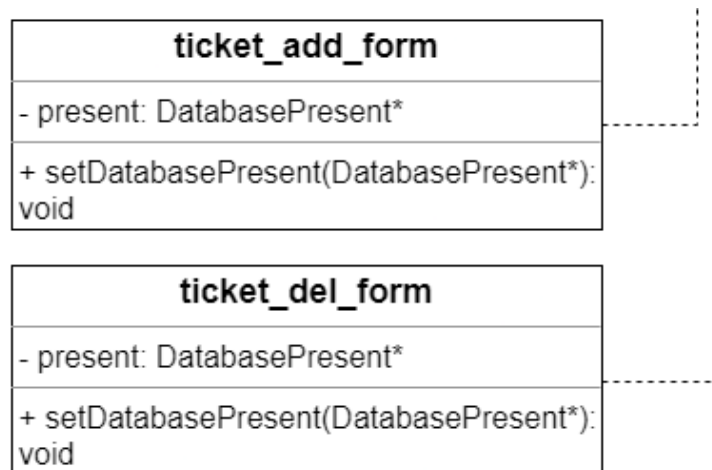
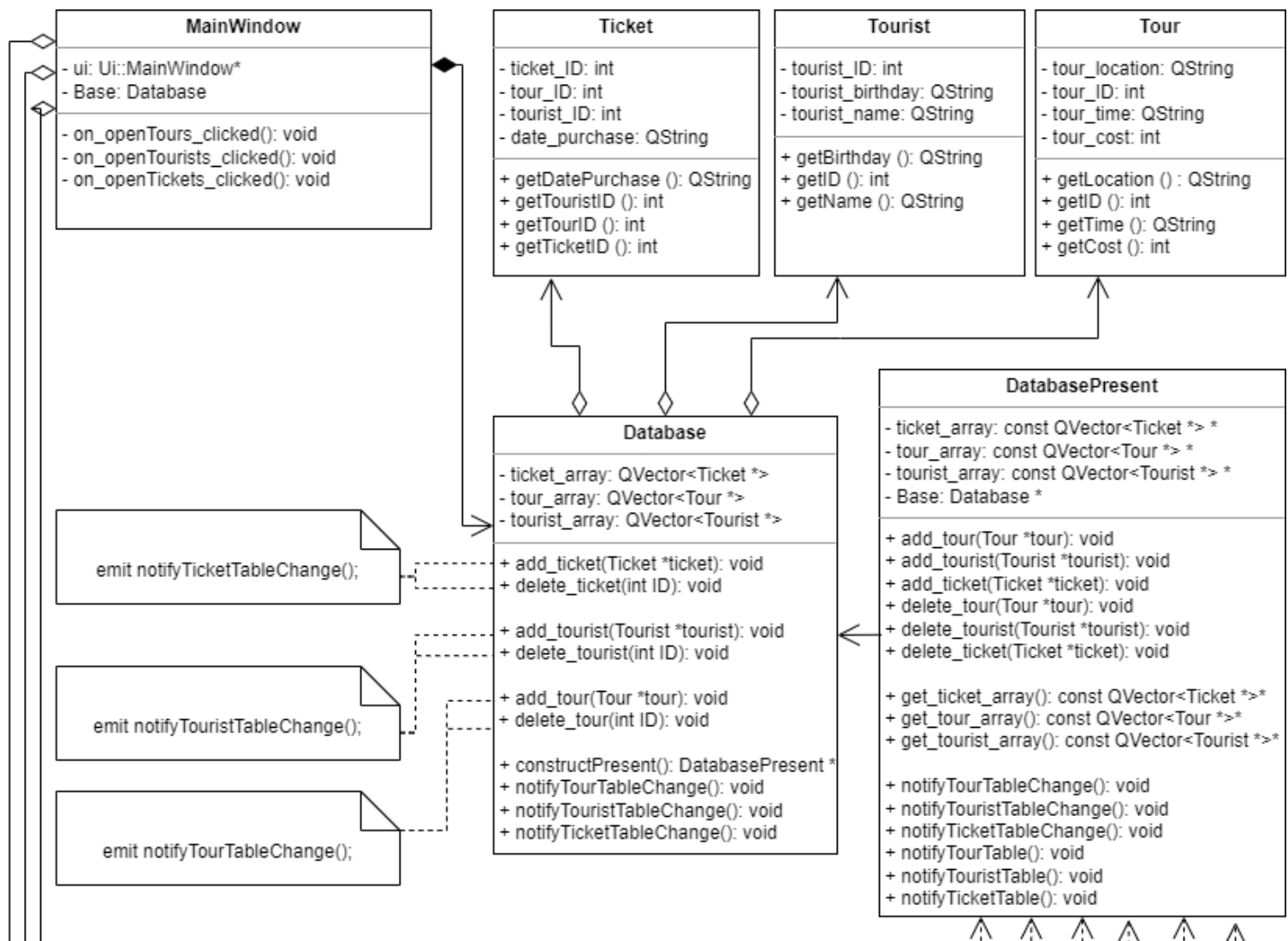


Рисунок 2 – Диаграмма паттерна Посредник

2.6 Полная диаграмма классов:



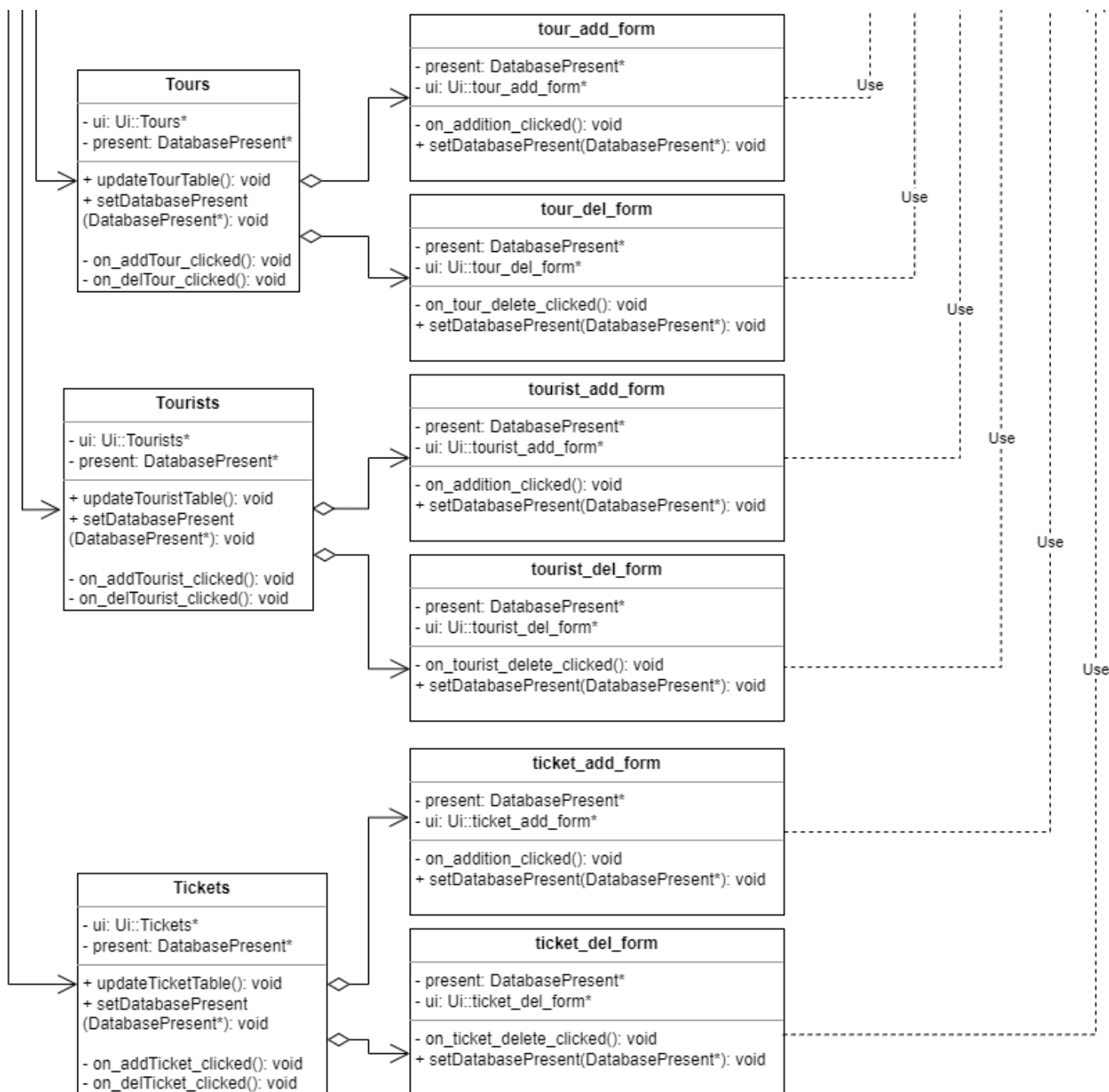


Рисунок 3 – Общая диаграмма классов

3. Разработка приложения:

3.1 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 нажатии кнопки «Удалить» откроется Форма для удаления билета.

The image shows a Qt Designer window with a form titled "Билеты". The form is designed with a light blue header bar containing the title. Below the header, there are two buttons: "Добавить" (Add) and "Удалить" (Delete). Underneath the buttons is a table with four columns: "ID", "ID путевки" (Itinerary ID), "ID туриста" (Tourist ID), and "Дата покупки" (Purchase date). The table is currently empty, showing only the header row.

Рисунок 4 – Форма для просмотра всех билетов

Объект	Класс
▼ Tickets	QWidget
▼ gridLayout	QGridLayout
addTicket	QPushButton
delTicket	QPushButton
tickets	QLabel
ticketTable	QTableWidget

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

Объект	Класс
Ticket_add_form	QWidget
formLayout	QFormLayout
addition	QPushButton
date_purchase	QLabel
date_purchase_input	QLineEdit
ticket_ID	QLabel
ticket_ID_input	QLineEdit
tour_ID	QLabel
tour_ID_input	QLineEdit
tourist_ID	QLabel
tourist_ID_input	QLineEdit

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.

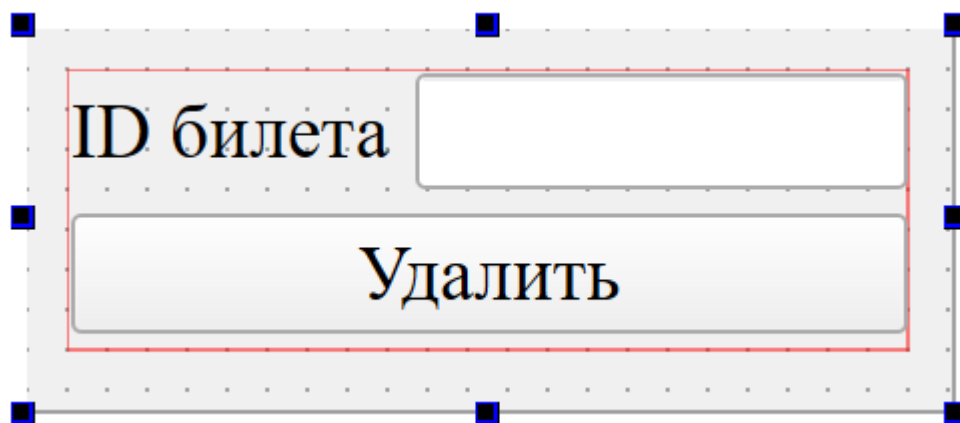


Рисунок 8 – Форма для удаления билета

Объект	Класс
Ticket_del_form	QWidget
formLayout_2	QFormLayout
ticket_ID	QLabel
ticket_ID_input	QLineEdit
ticket_delete	QPushButton

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 нажатии кнопки «Удалить» откроется Форма для удаления туриста.

Рисунок 10 – Форма для просмотра всех туристов

Объект	Класс
▼ Tourists	QWidget
▼ gridLayout_2	QGridLayout
addTourist	QPushButton
delTourist	QPushButton
tourists	QLabel
touristTable	QTableWidget

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

Объект	Класс
Tourist_add_form	QWidget
formLayout	QFormLayout
addition	QPushButton
tourist_ID	QLabel
tourist_ID_input	QLineEdit
tourist_birthday	QLabel
tourist_birthday_input	QLineEdit
tourist_name	QLabel
tourist_name_input	QLineEdit

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

Объект	Класс
Tourist_del_form	QWidget
formLayout_3	QFormLayout
tourist_ID	QLabel
tourist_ID_input	QLineEdit
tourist_delete	QPushButton

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 нажатии кнопки «Удалить» откроется Форма для удаления тура.

The image shows a Qt Designer window with a form titled "Туры". The form contains two buttons, "Добавить" and "Удалить", and a table with four columns: "ID", "Место отдыха", "Период отдыха", and "Стоимость путевки". The table is currently empty.

Рисунок 16 – Форма для просмотра всех туров

Объект	Класс
Tours	QWidget
gridLayout	QGridLayout
addTour	QPushButton
delTour	QPushButton
tours	QLabel
tourTable	QTableWidget

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

Объект	Класс
Tour_add_form	QWidget
formLayout	QFormLayout
addition	QPushButton
tour_ID	QLabel
tour_ID_input	QLineEdit
tour_cost	QLabel
tour_cost_input	QLineEdit
tour_location	QLabel
tour_location_input	QLineEdit
tour_time	QLabel
tour_time_input	QLineEdit

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

Объект	Класс
Tour_del_form	QWidget
formLayout_5	QFormLayout
tour_ID	QLabel
tour_ID_input	QLineEdit
tour_delete	QPushButton

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


Объект	Класс
▼ MainWindow	QMainWindow
▼  centralwidget	QWidget
openTickets	QPushButton
openTourists	QPushButton
openTours	QPushButton
menubar	QMenuBar
statusbar	QStatusBar

Рисунок 11 – Объекты с Формы главного экрана

3.2 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 установления 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 установления 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()`.

4. Тестирование приложения

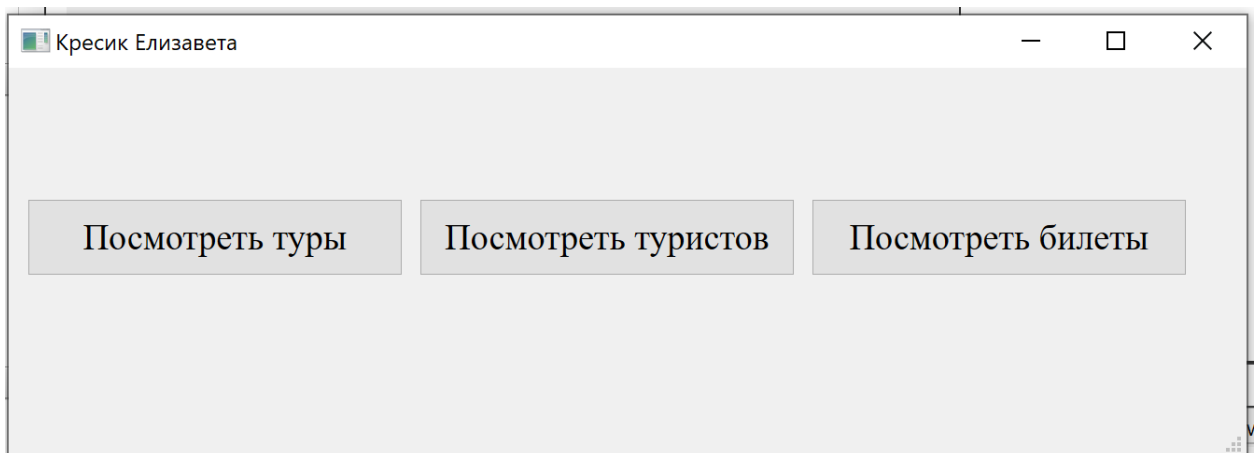


Рисунок 12 – При запуске программы показывается главная страница

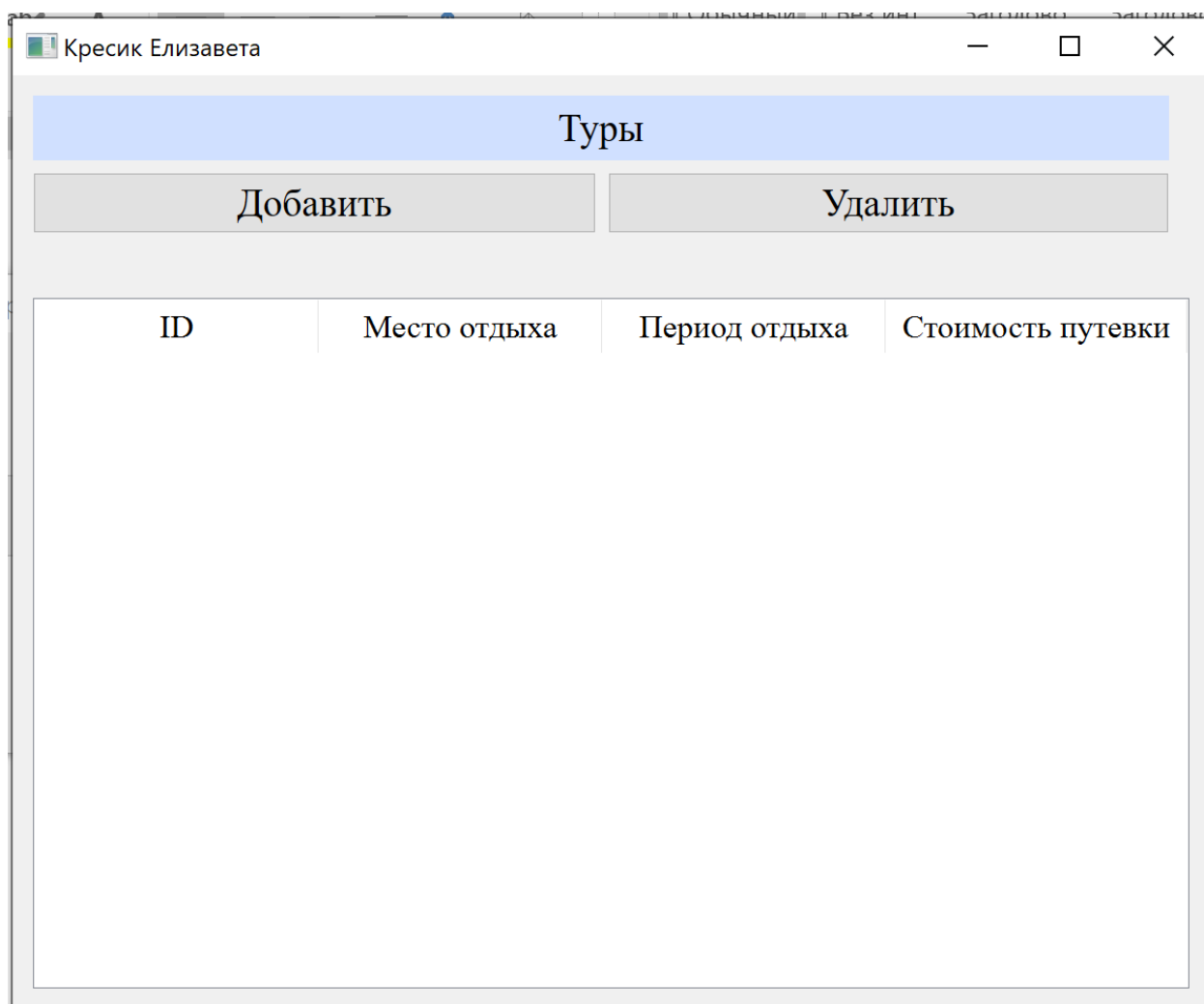


Рисунок 13 – Эта страница открывается после нажатия «Посмотреть туры» на главной странице

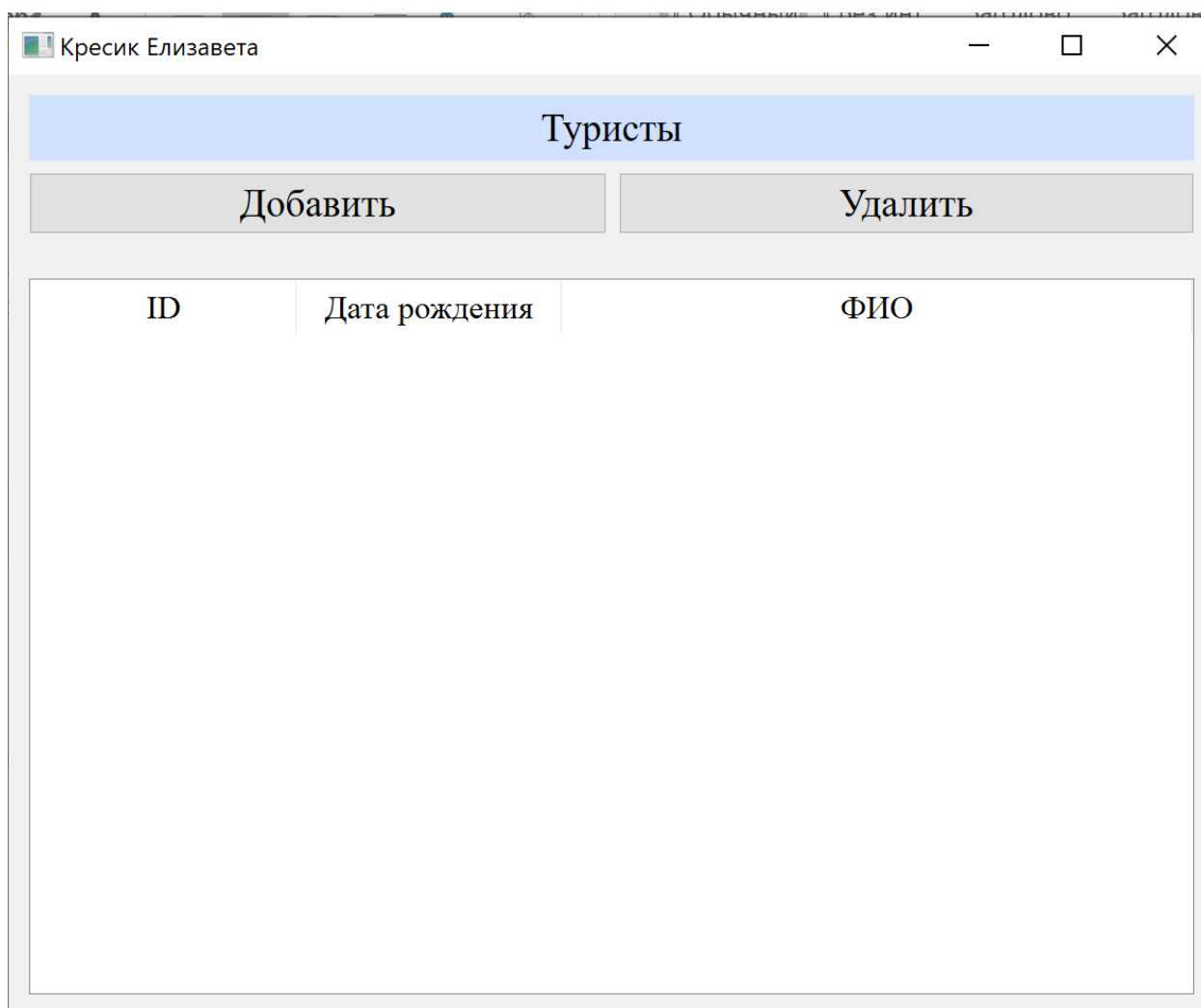


Рисунок 14 – Эта страница открывается после нажатия «Посмотреть туристоров» на главной странице

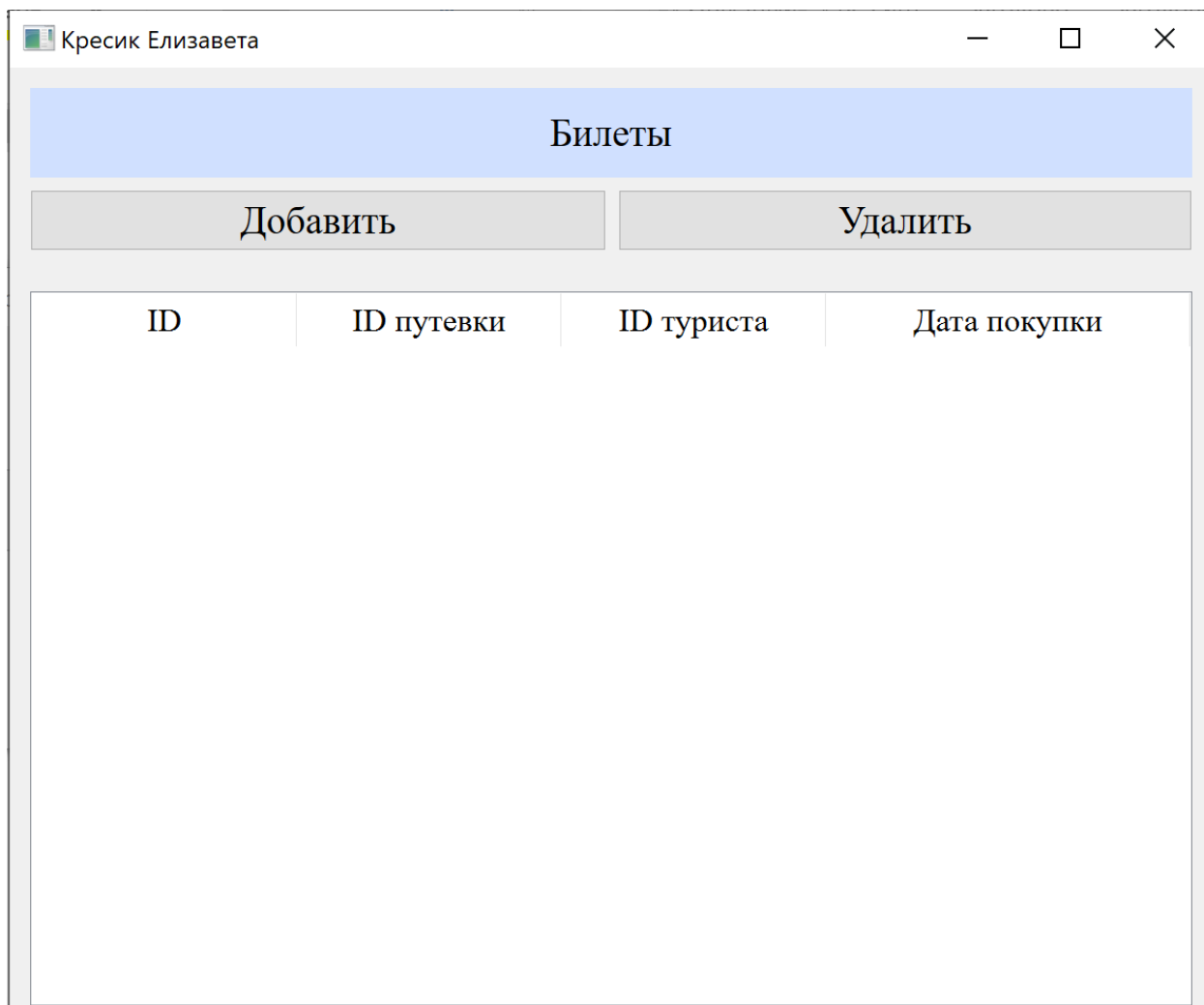


Рисунок 15 – Эта страница открывается после нажатия «Посмотреть билеты»
на главной странице

Добавление тура

ID тура: 12

Место отдыха: Сочи

Период отдыха: 20.12.2023-27.12.2023

Стоимость: 100000

Добавить

Рисунок 16 – Добавление тура

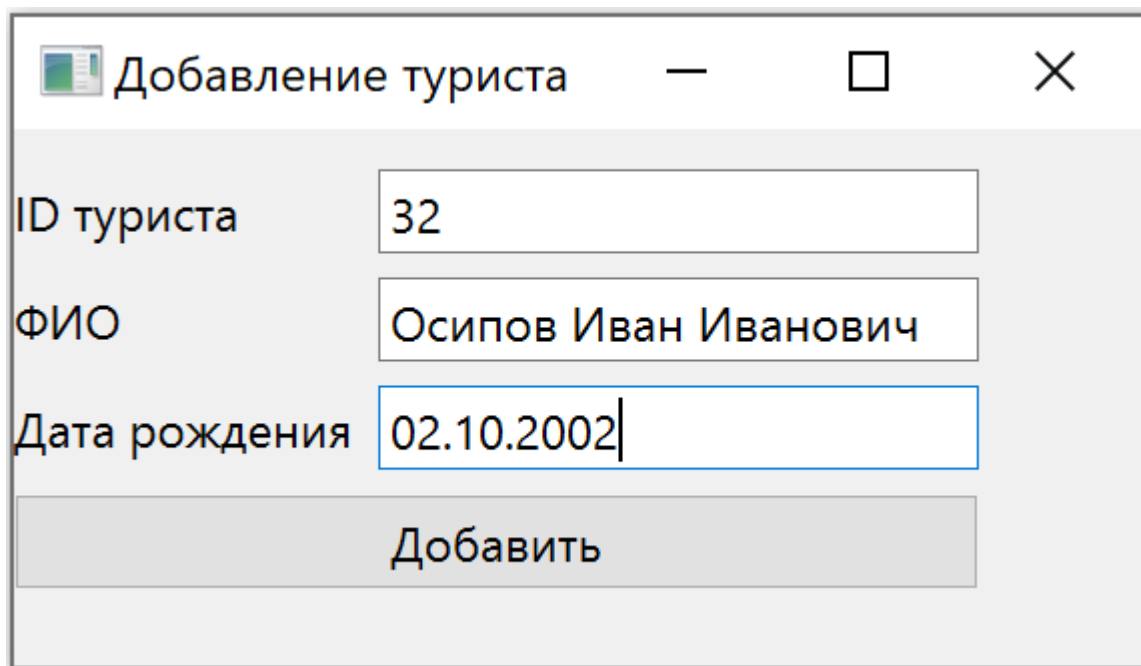
Кресик Елизавета

Туры

Добавить Удалить

	ID	Место отдыха	Период отдыха	Стоимость путевки
1	12	Сочи	20.12.2023-27.12.2023	100000

Рисунок 17 – Таблица туров после добавления тура



Добавление туриста

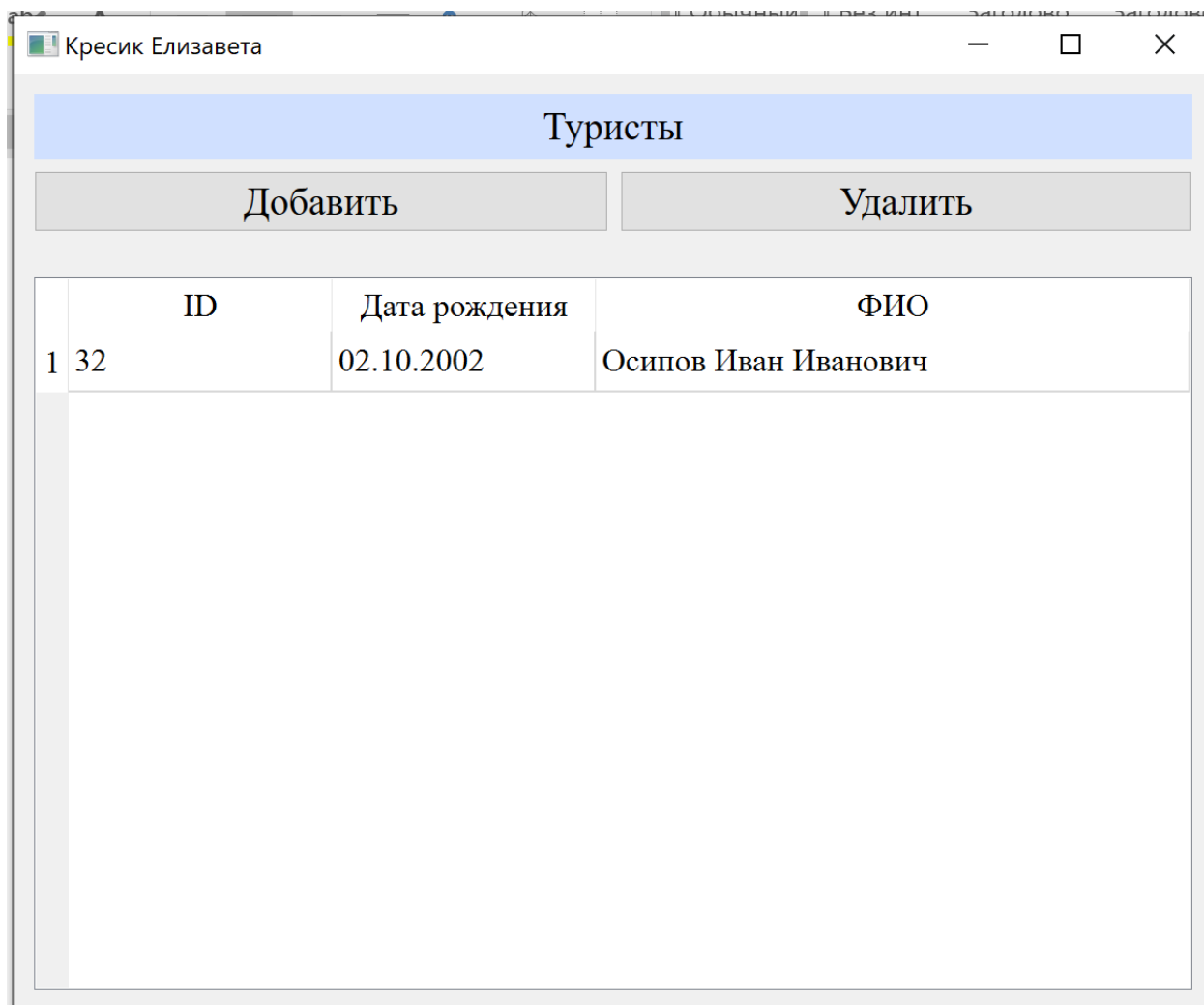
ID туриста: 32

ФИО: Осипов Иван Иванович

Дата рождения: 02.10.2002

Добавить

Рисунок 18 – Добавление туриста



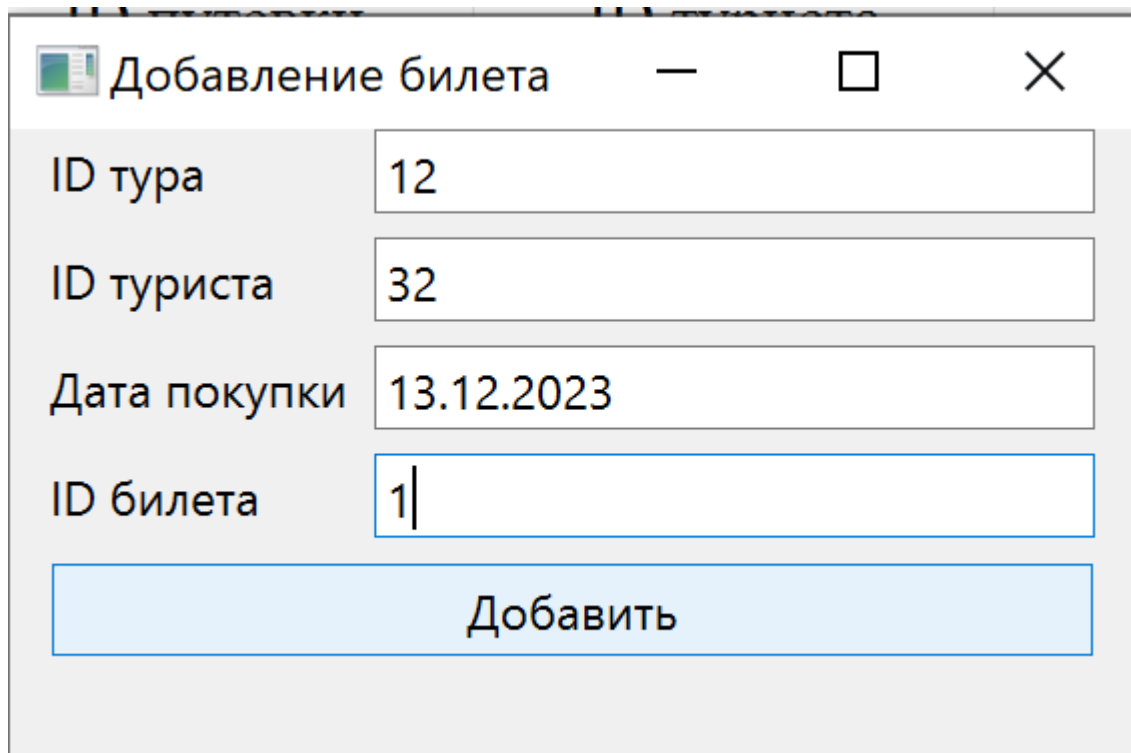
Кресик Елизавета

Туристы

Добавить Удалить

ID	Дата рождения	ФИО
1 32	02.10.2002	Осипов Иван Иванович

Рисунок 19 – Таблица туристов после добавления туриста



Добавление билета

ID тура	12
ID туриста	32
Дата покупки	13.12.2023
ID билета	1

Добавить

Рисунок 20 – Добавление билета

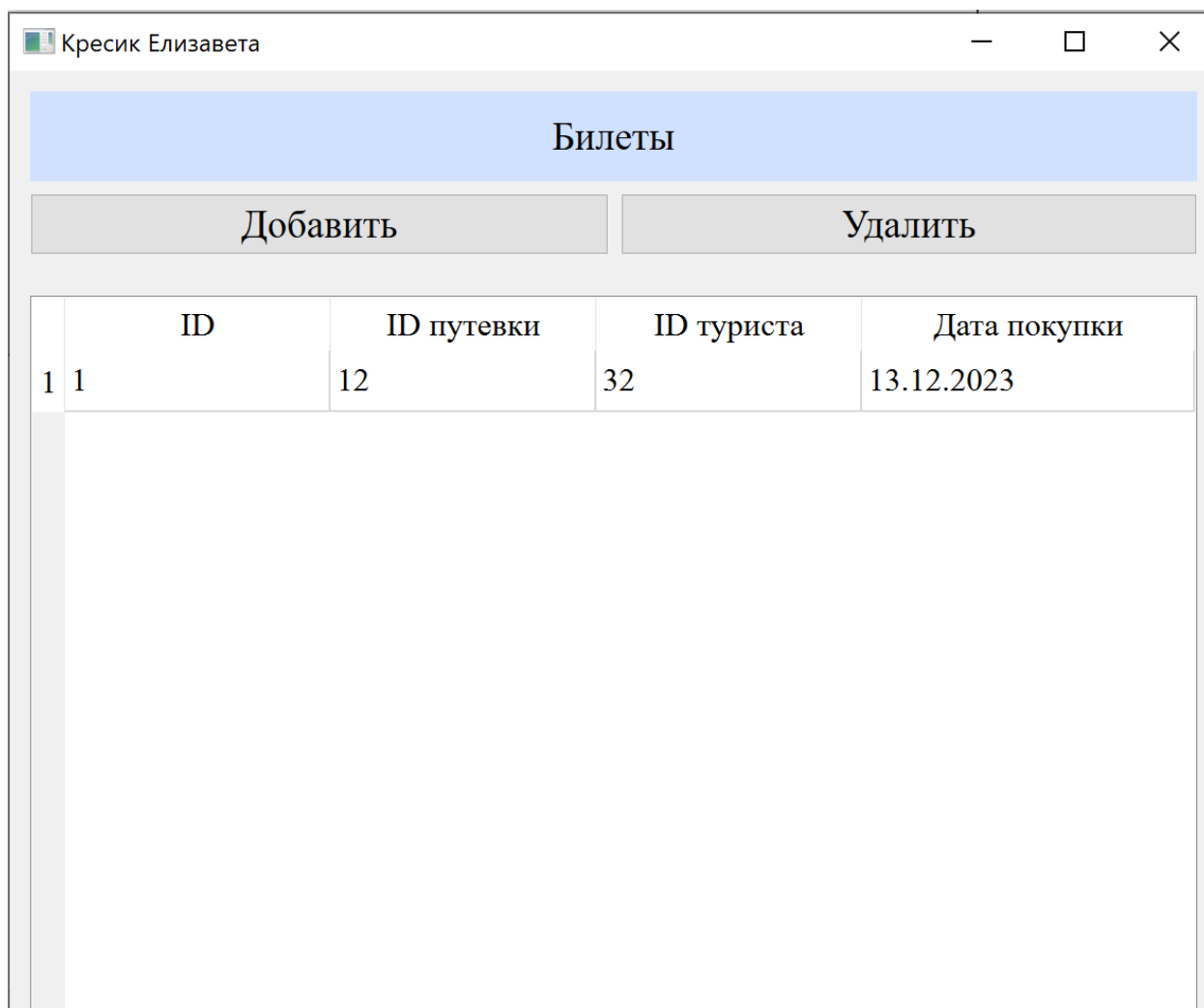


Рисунок 21 – Таблица билетов после добавления билета

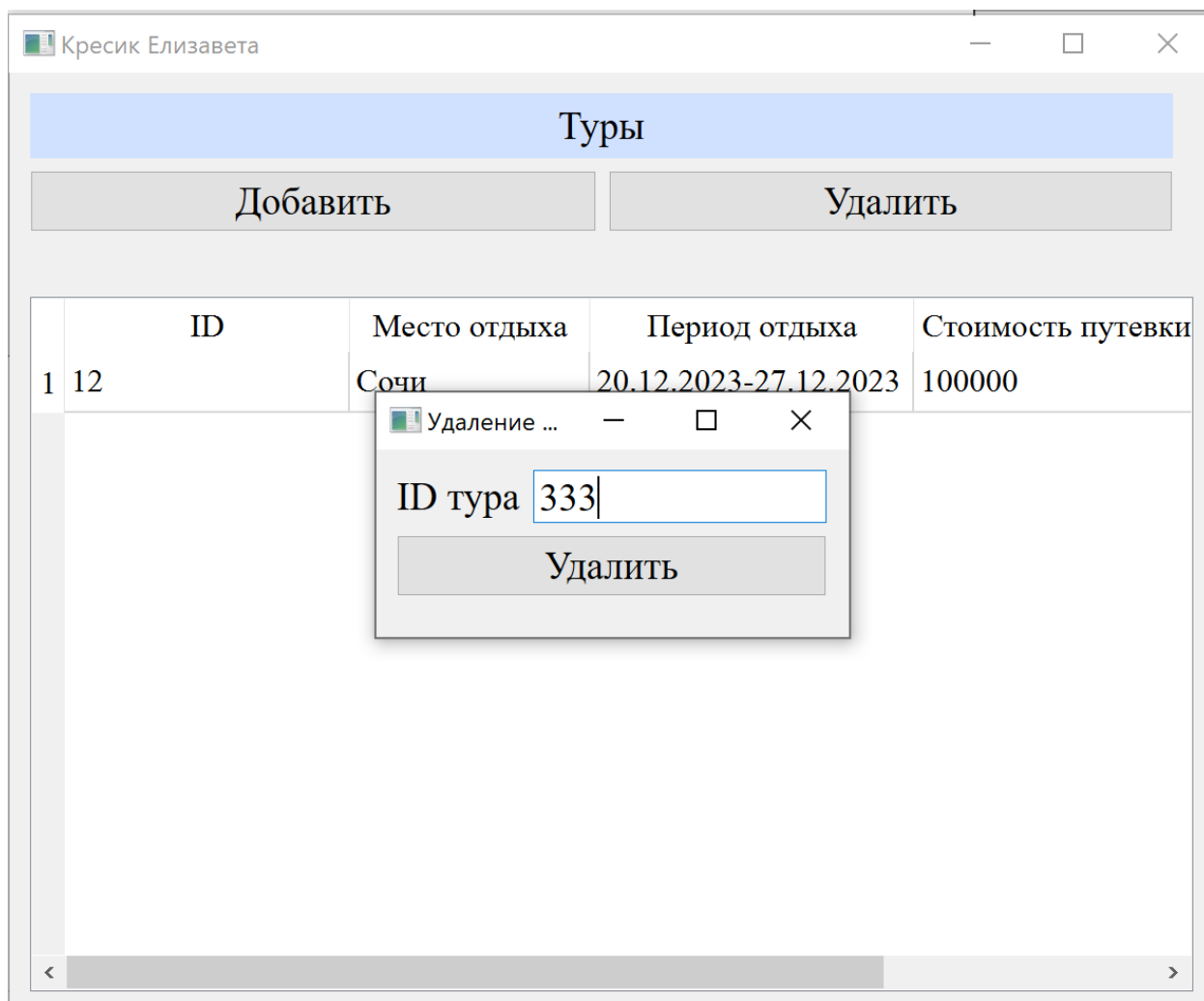


Рисунок 22 – Удаление несуществующего тура

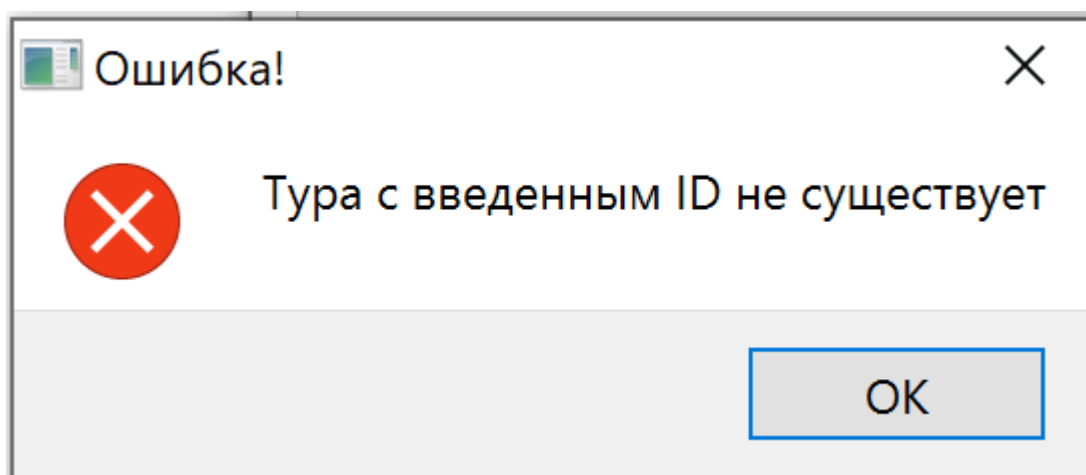


Рисунок 23 – Уведомление об ошибке при удалении несуществующего тура

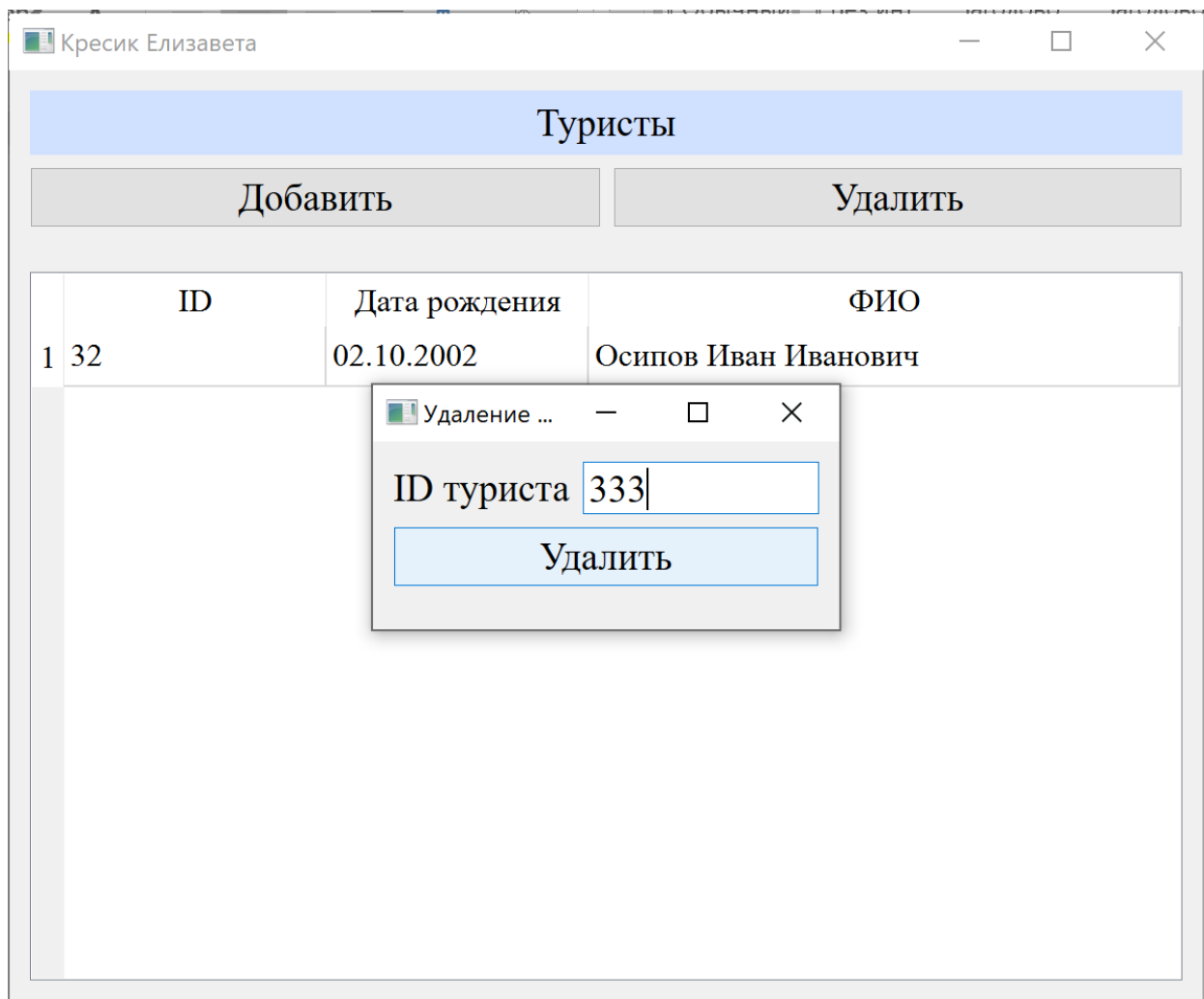


Рисунок 24 – Удаление несуществующего туриста

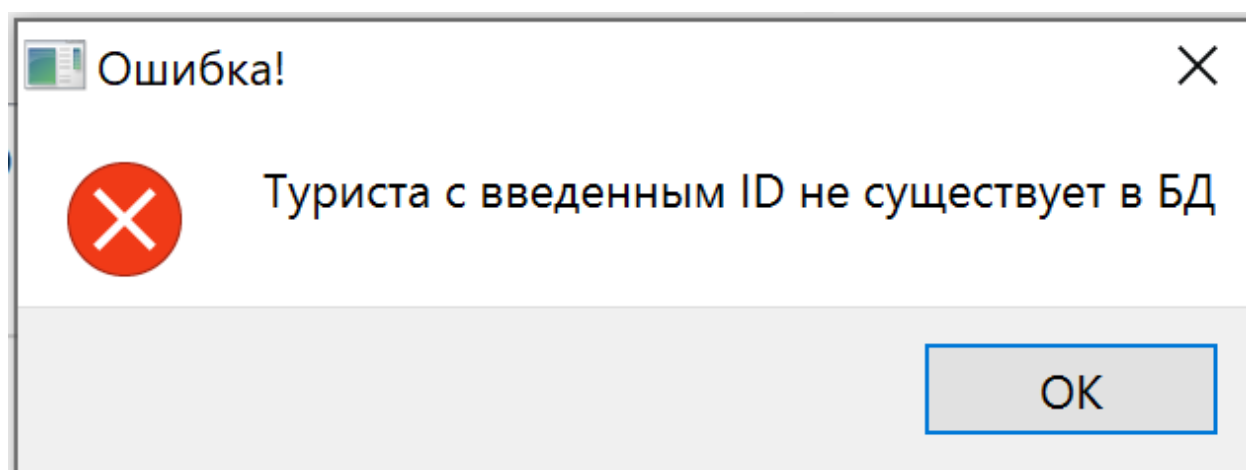


Рисунок 25 – Уведомление об ошибке при удалении несуществующего туриста

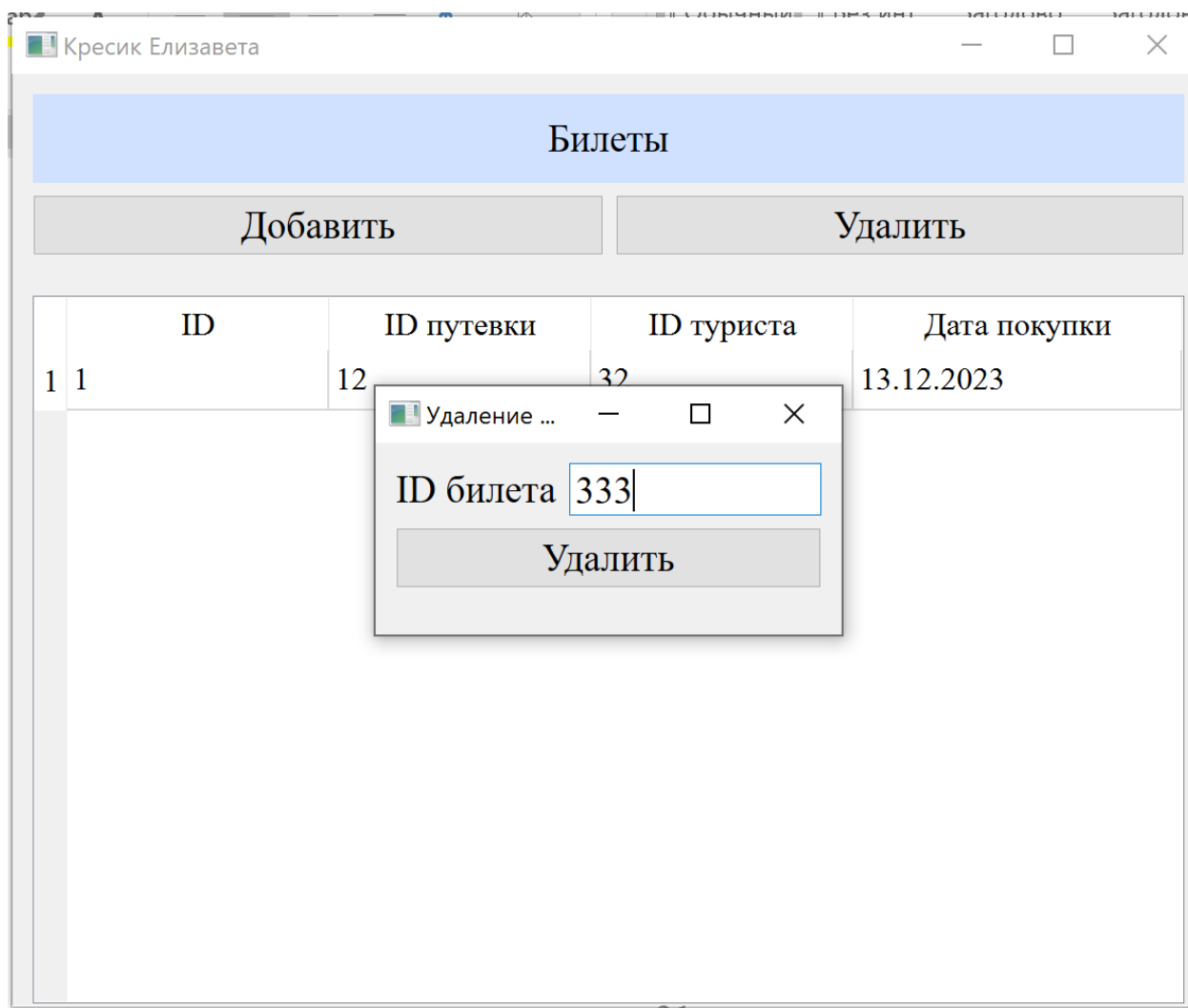


Рисунок 26 – Удаление несуществующего билета

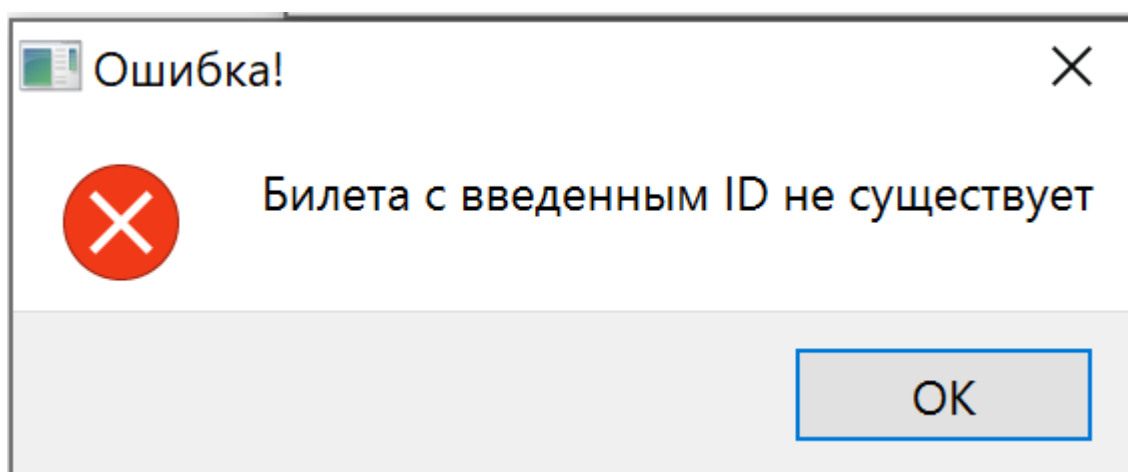


Рисунок 27 – Уведомление об ошибке при удалении несуществующего билета

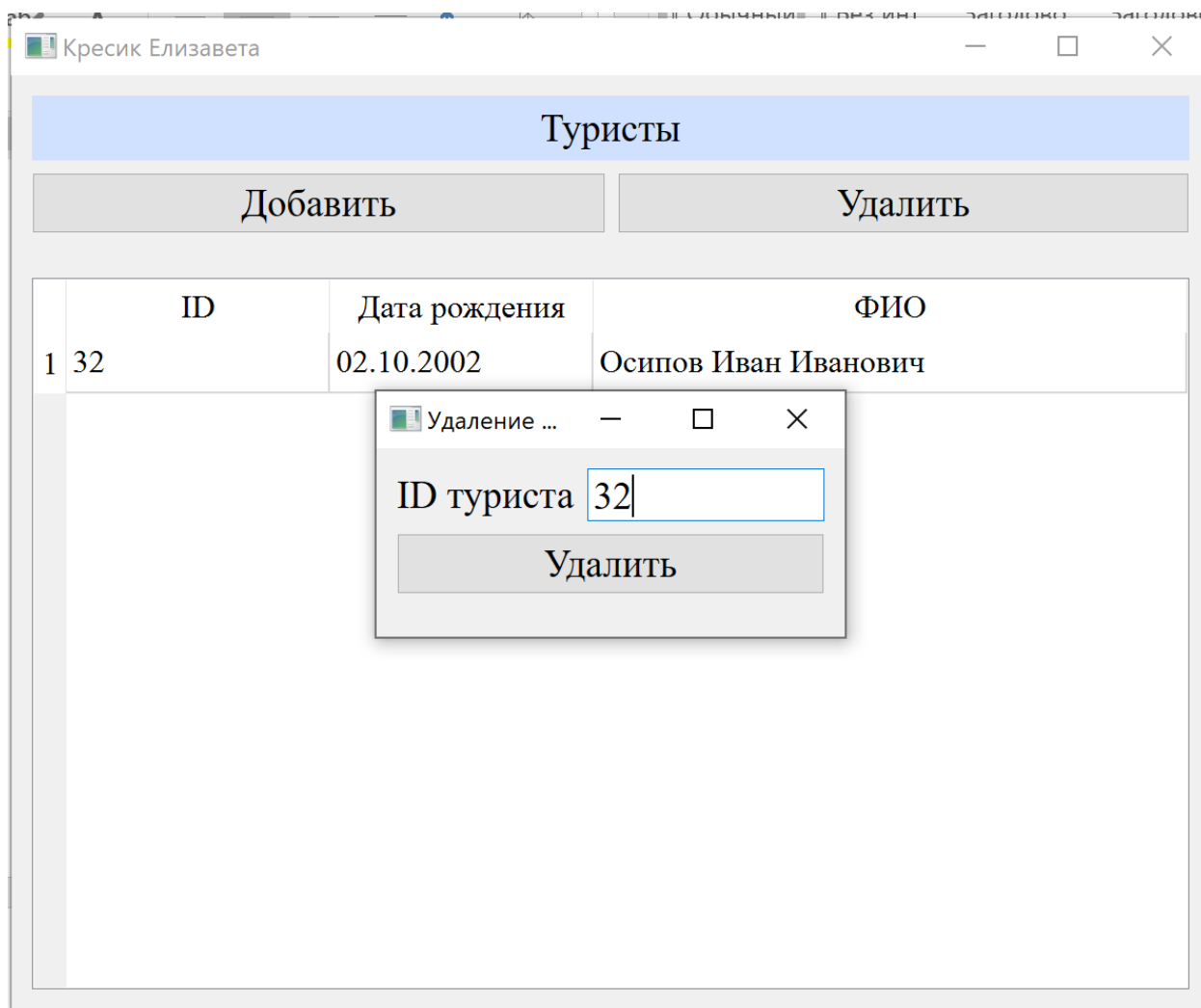


Рисунок 28 – Удаление туриста при наличии купленных на него билетов

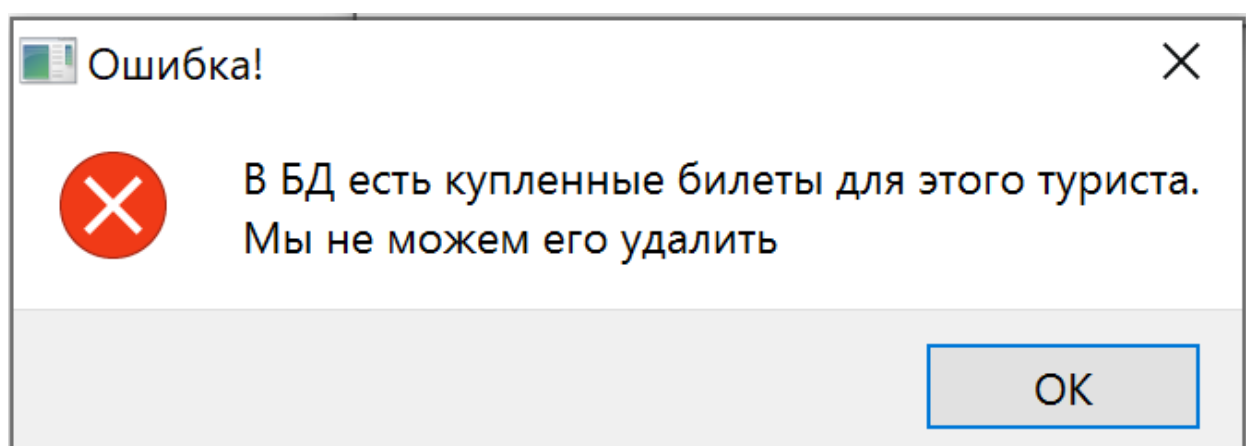


Рисунок 29 – Уведомление об ошибке при удалении туриста с наличием купленных на него билетов

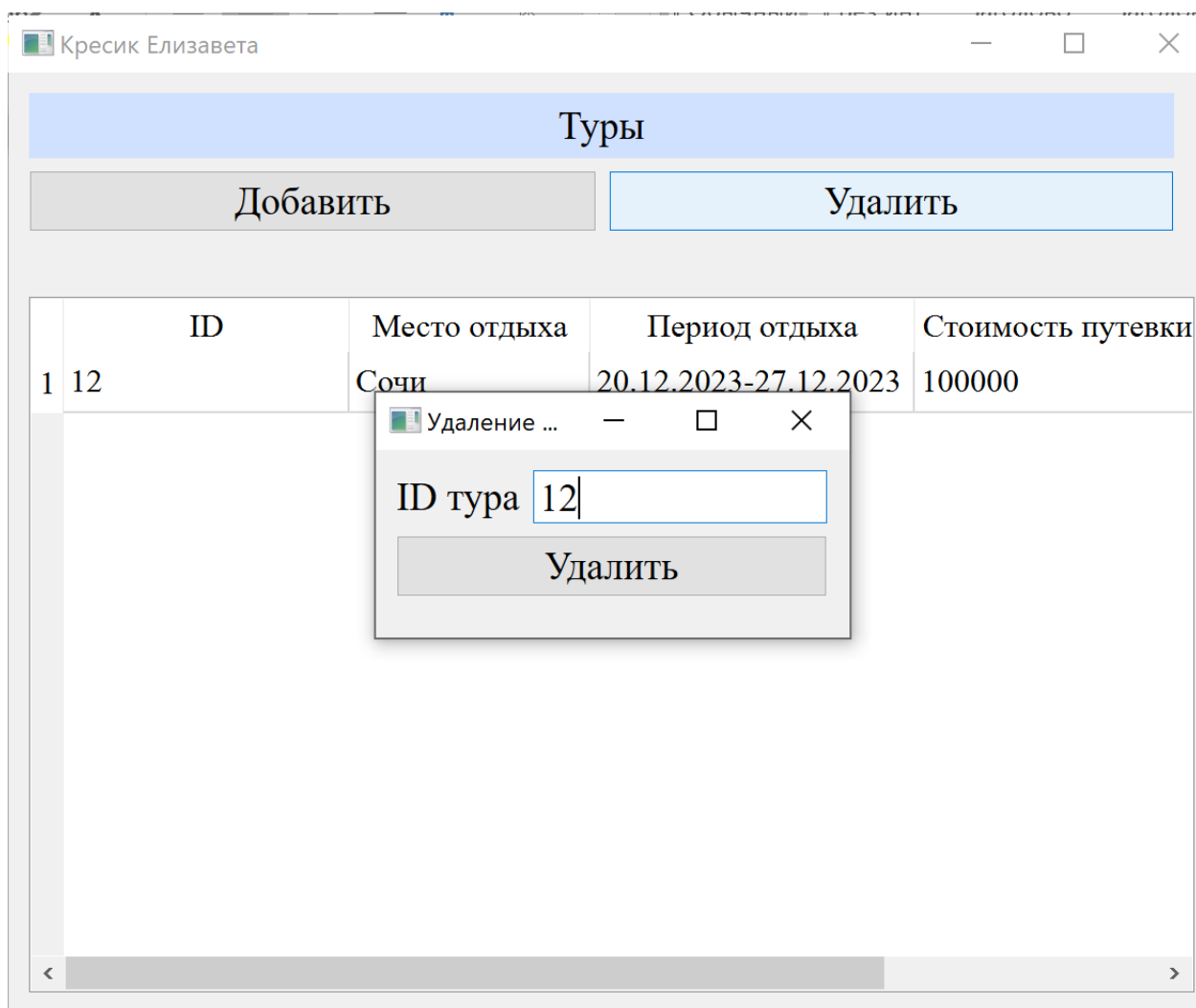


Рисунок 30 – Удаление тура при наличии купленных на него билетов

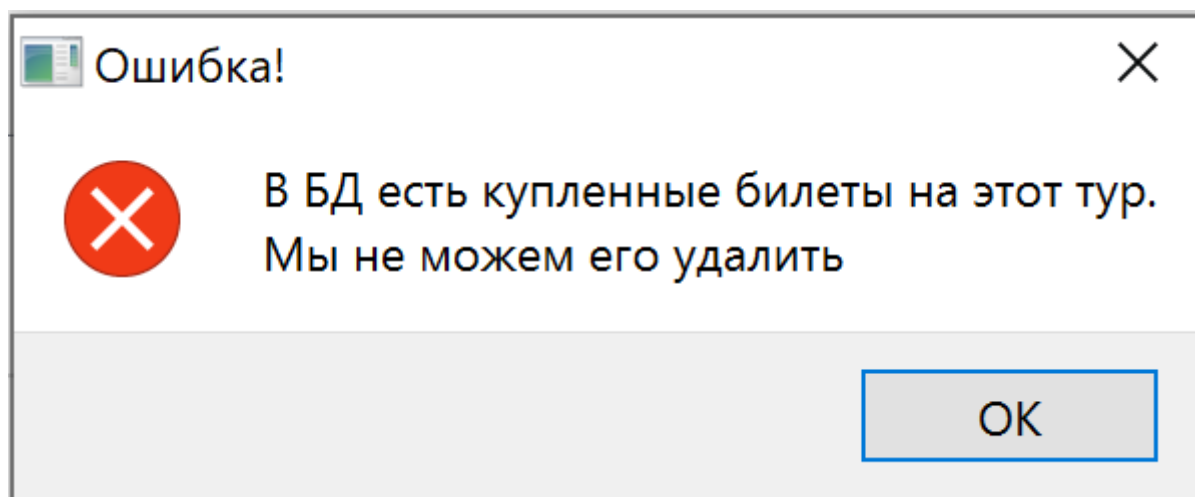


Рисунок 31 – Уведомление об ошибке при удалении тура с наличием купленных на него билетов

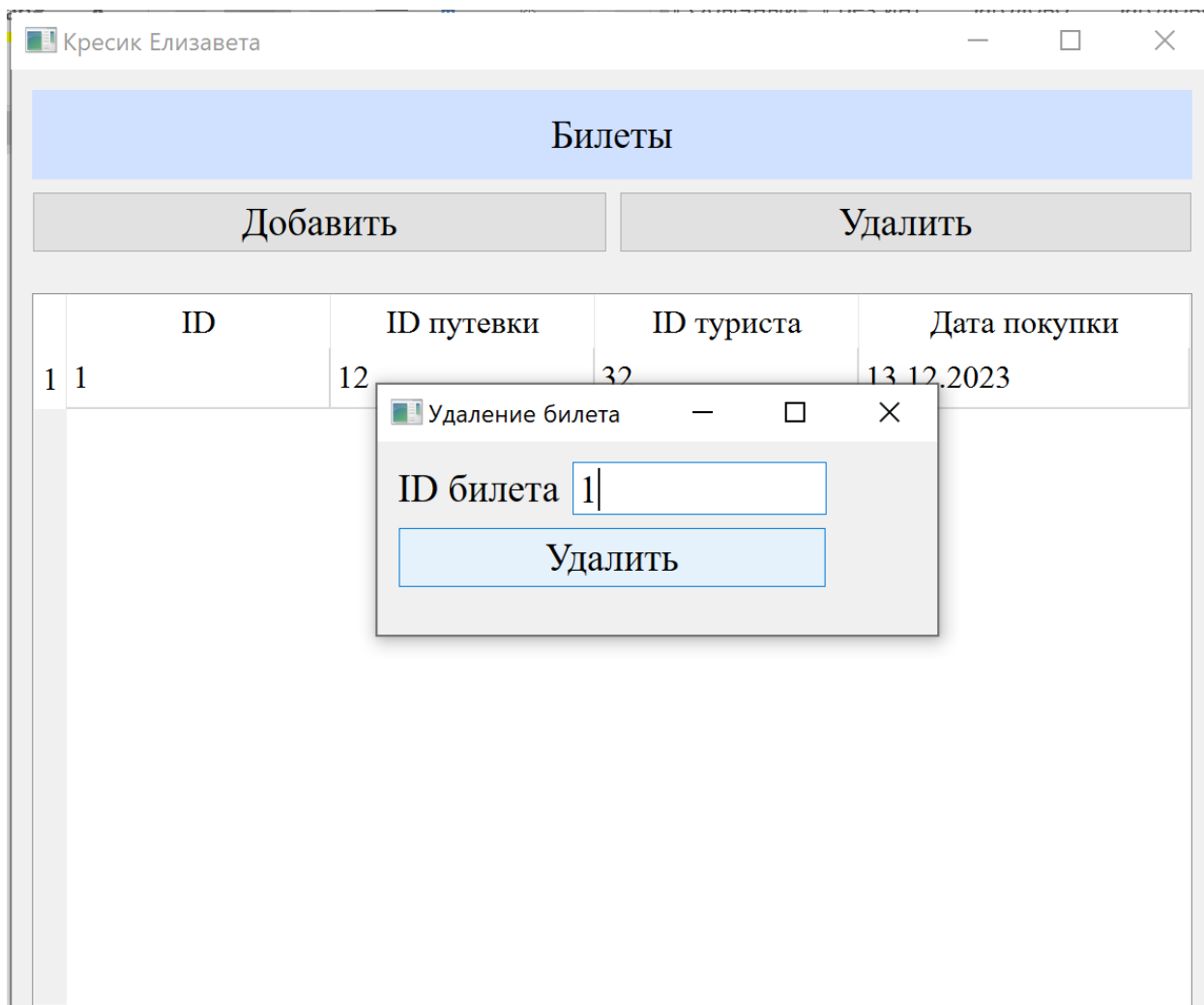


Рисунок 32 – Удаление билета



Рисунок 33 – Таблица билетов после удаления

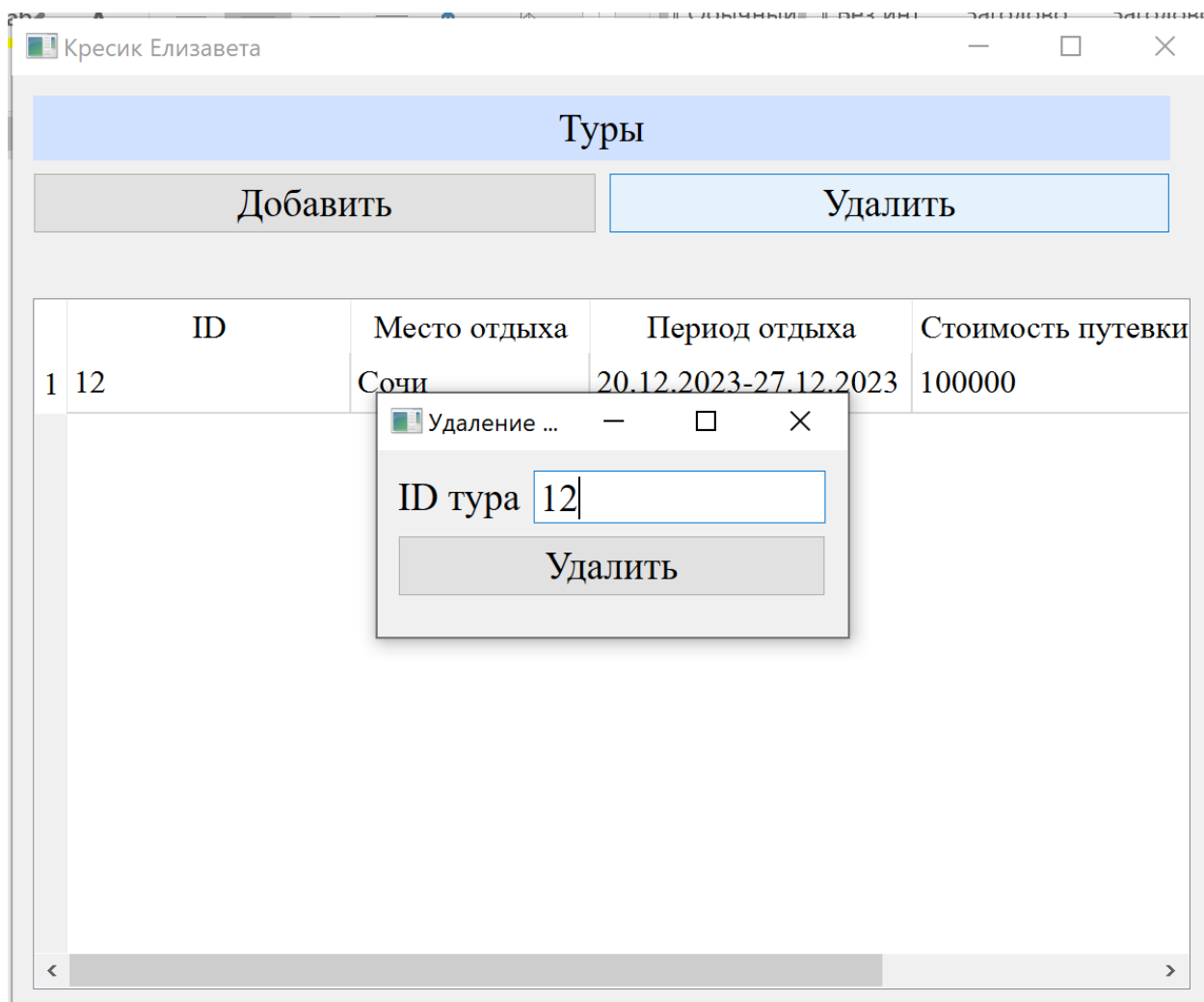


Рисунок 34 – Удаление тура

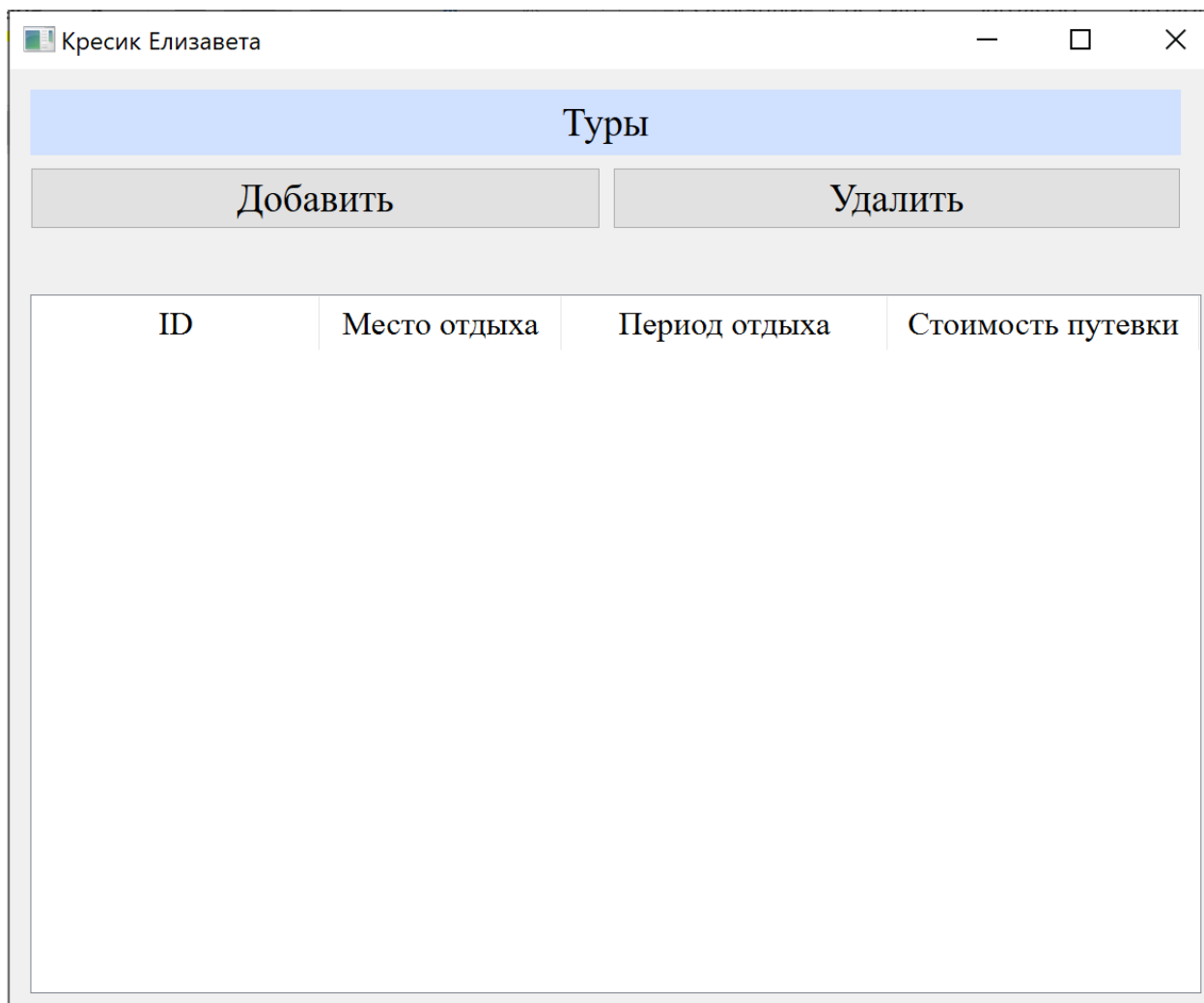


Рисунок 35 – Таблица туров после удаления

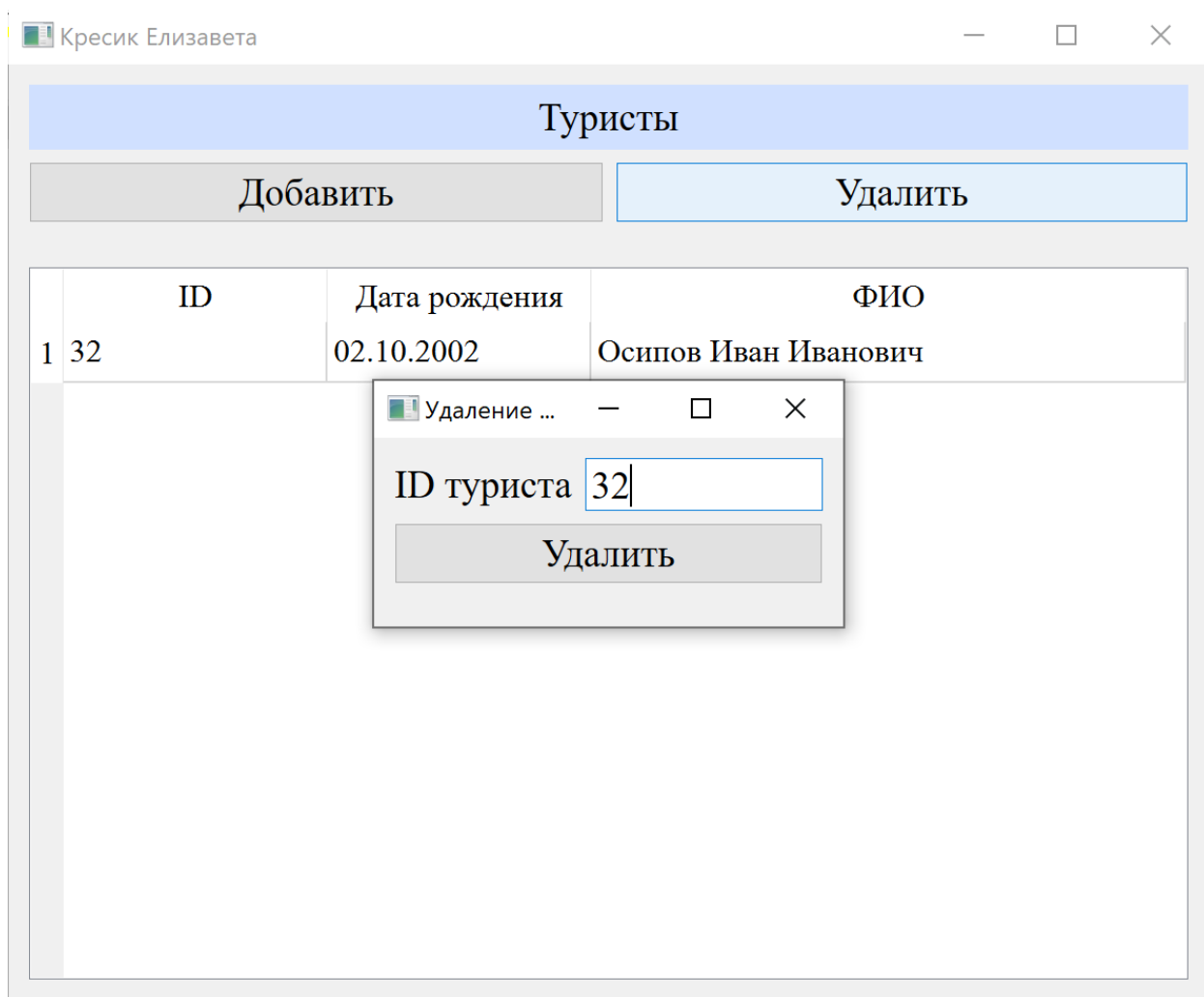


Рисунок 36 – Удаление туриста

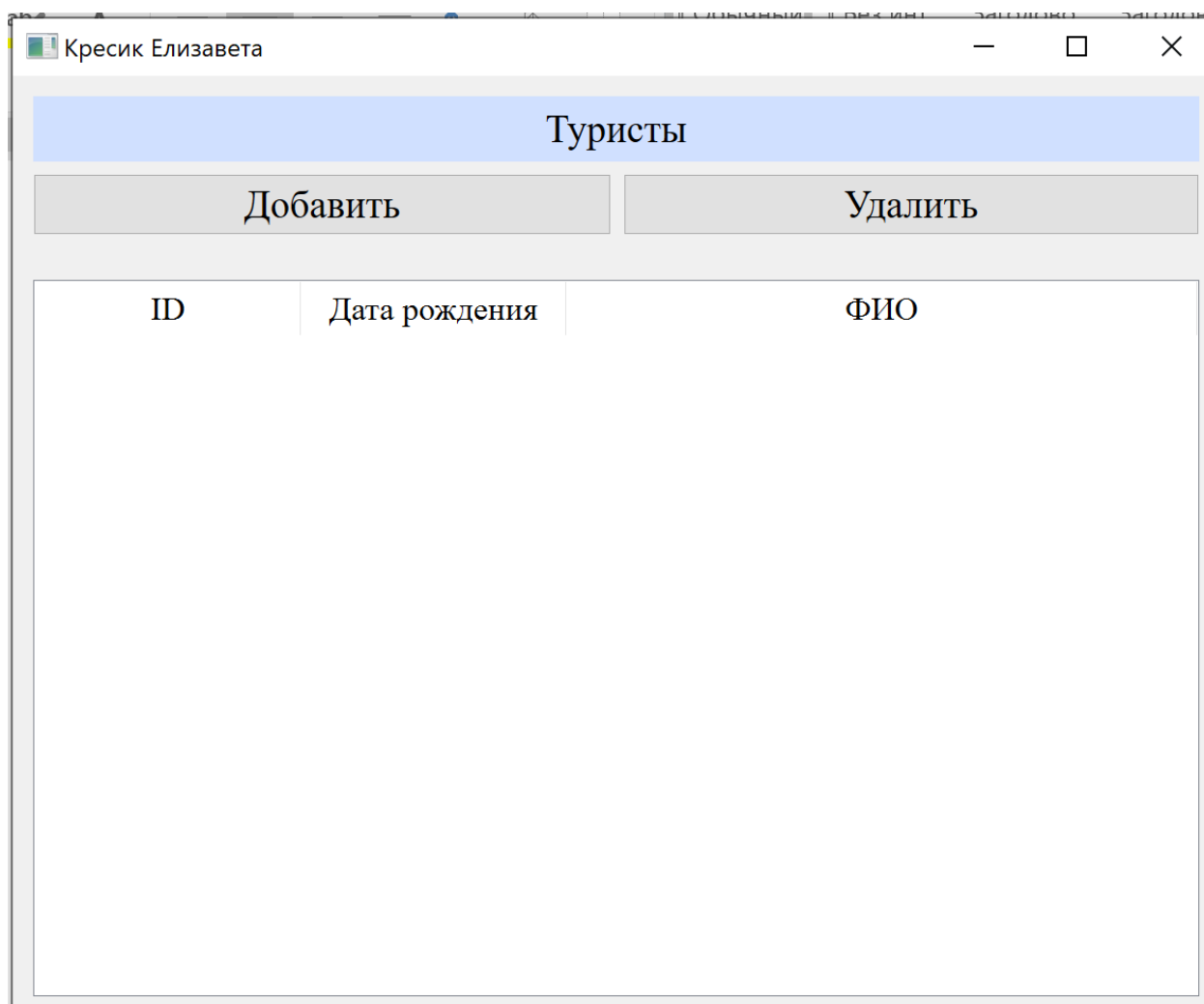


Рисунок 37 – Таблица туристов после удаления

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
- stackoverflow.com

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.\nМы 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.\nМы 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()));
    }

```