УО «Белорусский государственный технологический университет»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

КУРСОВОГО ПРОЕКТА

по дисциплине «Базы данных»

|  |  |  |  |
| --- | --- | --- | --- |
| Исполнитель |  |  |  |
| Студент 2 курса группы 1 |  |  |  |
|  | подпись, дата |  | инициалы и фамилия |
| Руководитель |  |  |  |
|  | подпись, дата |  | инициалы и фамилия |

|  |  |
| --- | --- |
| Курсовой проект защищен с оценкой \_\_\_\_\_\_\_\_\_\_\_\_ |  |

Минск 2024

**Содержание**

[Введение 6](#_Toc167314488)

[1.Обзор технических методов и программных средств разработки веб–сайта 7](#_Toc167314489)

[1.1.Обзор аналогичных решений 7](#_Toc167314490)

[1.1.1. Аналог «Booking» 7](#_Toc167314491)

[1.1.2. Аналог «Яндекс Travel» 8](#_Toc167314492)

[1.1.3 Аналог «Aviasales» 9](#_Toc167314493)

[1.2 Разработка функциональных требований 10](#_Toc167314494)

[1.3 Вывод 10](#_Toc167314495)

[2. Разработка архитектуры проекта 11](#_Toc167314496)

[2.1 Структура управлением приложения 11](#_Toc167314497)

[2.2 Диаграммы UML, взаимосвязь всех компонентов. 12](#_Toc167314498)

[2.3 Диаграмма БД, взаимосвязь компонентов. Описание объектов и ограничение целостности 13](#_Toc167314499)

[2.4 Описание информационных объектов и ограничений целостности. 14](#_Toc167314500)

[2.4.1 Passengers (Пассажиры): 14](#_Toc167314501)

[2.4.2 Tickets (Билеты): 14](#_Toc167314502)

[2.4.3 ShoppingCart (Корзина): 14](#_Toc167314503)

[2.4.4 Flights (Рейсы): 14](#_Toc167314504)

[2.4.5 Airplanes (Самолеты): 15](#_Toc167314505)

[2.4.6 Employees (Сотрудники): 15](#_Toc167314506)

[2.4.7 Crews (Экипажи): 15](#_Toc167314507)

[2.5 Вывод по разделу. 15](#_Toc167314508)

[3. Разработка модели базы данных. Создание необходимых объектов. 16](#_Toc167314509)

[3.1 Таблицы 16](#_Toc167314510)

[3.2 Процедуры 16](#_Toc167314511)

[3.2.1 Самолёты: 16](#_Toc167314512)

[3.2.2 Рейсы: 17](#_Toc167314513)

[3.2.3 Пассажиры: 17](#_Toc167314514)

[3.2.4 Билеты: 17](#_Toc167314515)

[3.2.5 Сотрудники: 18](#_Toc167314516)

[3.2.6 Экипаж: 18](#_Toc167314517)

[3.3 Функции: 18](#_Toc167314518)

[3.3.1 Самолёты: 19](#_Toc167314519)

[3.3.2 Рейсы: 19](#_Toc167314520)

[3.3.3 Пассажиры: 19](#_Toc167314521)

[3.3.4 Билеты: 19](#_Toc167314522)

[3.3.5 Сотрудники: 19](#_Toc167314523)

[3.3.6 Экипаж: 19](#_Toc167314524)

[3.4 Триггеры: 20](#_Toc167314525)

[3.4.1 Самолёты: 20](#_Toc167314526)

[3.4.2 Рейсы: 20](#_Toc167314527)

[3.4.3 Пассажиры: 20](#_Toc167314528)

[3.4.4 Билеты: 21](#_Toc167314529)

[3.4.5 Сотрудники: 21](#_Toc167314530)

[3.4.6 Экипаж: 21](#_Toc167314531)

[3.5 Индексы 21](#_Toc167314532)

[4. Описание используемой технологии. 24](#_Toc167314533)

[4.1 Измерения 26](#_Toc167314534)

[4.2 Иерархии 26](#_Toc167314535)

[4.3 Кубы 29](#_Toc167314536)

[5. Описание дополнительных функцих. 32](#_Toc167314537)

[5.1 Импорт и экспорт данных 32](#_Toc167314538)

[5.2 Создание клиент–серверной архитектуры. 34](#_Toc167314539)

[Заключение 40](#_Toc167314540)

[Список использованных источников 41](#_Toc167314541)

[Приложение А. Листинг таблиц 42](#_Toc167314543)

[Приложение В. Листинг представлений 44](#_Toc167314544)

# Введение

В современном мире сфера электронной коммерции и информационные технологии играют ключевую роль в повышении эффективности управления и обработки данных в различных отраслях. Растущий интерес к онлайн–покупкам подтверждает необходимость эффективного управления базами данных для различных интернет–сервисов.

В курсовом проекте будет разработана база данных для авиакомпании. Будут использованы передовые технологии систем управления базами данных – Microsoft SQL Server. Это решение обеспечит надежность, стабильность, масштабируемость и высокую производительность итогового продукта, необходимого для эффективной работы в сфере электронной коммерции и авиаперевозок.

Основные требования к итоговому проекту:

– База данных должна быть реализована в СУБД «Microsoft SQL Server».

­­– Реализация прав для администратора БД и клиента

– Реализовать систему хранения объектов: рейсы, пути, самолёты.

– Реализовать функционал администратора: добавление, редактирование, удаление данных, оформление билетов.

– Добавить систему сортировки данных в БД.

– Добавить систему поиска данных в БД.

– Добавить возможность покупки и возврата билета пользователем.

– Реализовать возможность создания отчётов (отчёт о кол–ве проданных билетов за промежуток времени).

– Реализовать систему аналитики данных (кол–во рейсов, средний пройденный путь, средняя цена билетов).

# Обзор технических методов и программных средств разработки веб-сайта

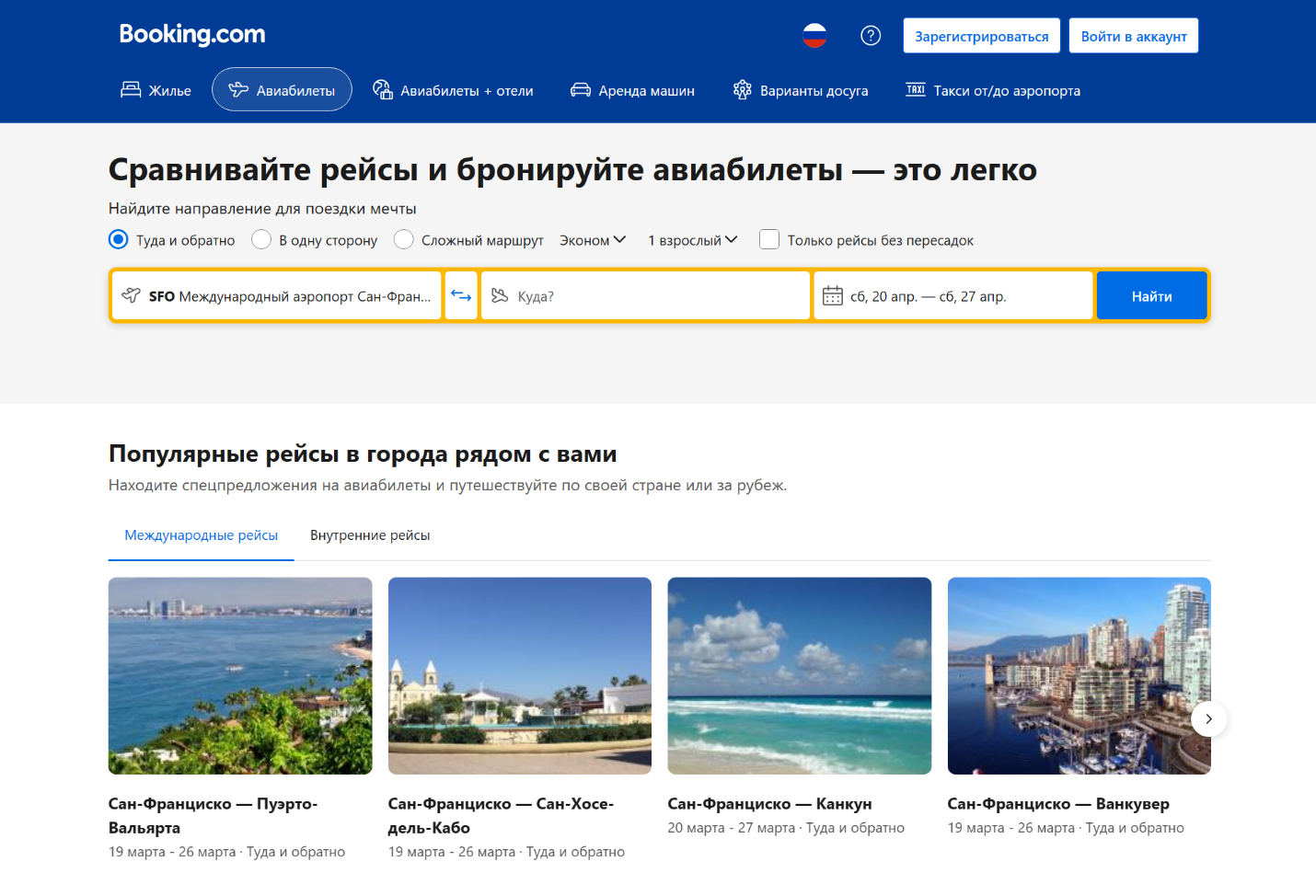
## Обзор аналогичных решений

### Аналог «Booking»

Анализ аналогичных решений является важным этапом разработки любого проекта. Он позволяет выявить ключевые функции, которые ценятся пользователями, а также определить лучшие практики для обеспечения удовлетворения их потребностей.

В данном обзоре рассмотрены три популярных сервиса для путешественников: Booking.com, Travel.yandex.ru и Aviasales.ru.

Booking.com[1]– международный сервис, предлагающий широкий выбор вариантов размещения, от отелей до апартаментов.

  
Рисунок 1.1 – Главная страница сайта booking.com

Ключевые особенности Booking.com:

– Выбор и бронирование билетов и жилья: Сайт предоставляет доступ к обширной базе данных отелей, хостелов, апартаментов и других видов размещения по всему миру.

– Информация о ценах и наличии услуг: Пользователи могут легко сравнивать цены, просматривать фотографии, читать отзывы и получать детальную информацию о каждом варианте размещения.

– Корзина: Позволяет добавлять понравившиеся варианты и бронировать их позже.

– Каталог: Структурированный каталог помогает быстро найти подходящий вариант размещения по заданным критериям (страна, город, тип жилья, ценовой диапазон).

– Личный кабинет: В личном кабинете хранится история бронирований, настройки пользователя и другая полезная информация.

### Аналог «Яндекс Travel»

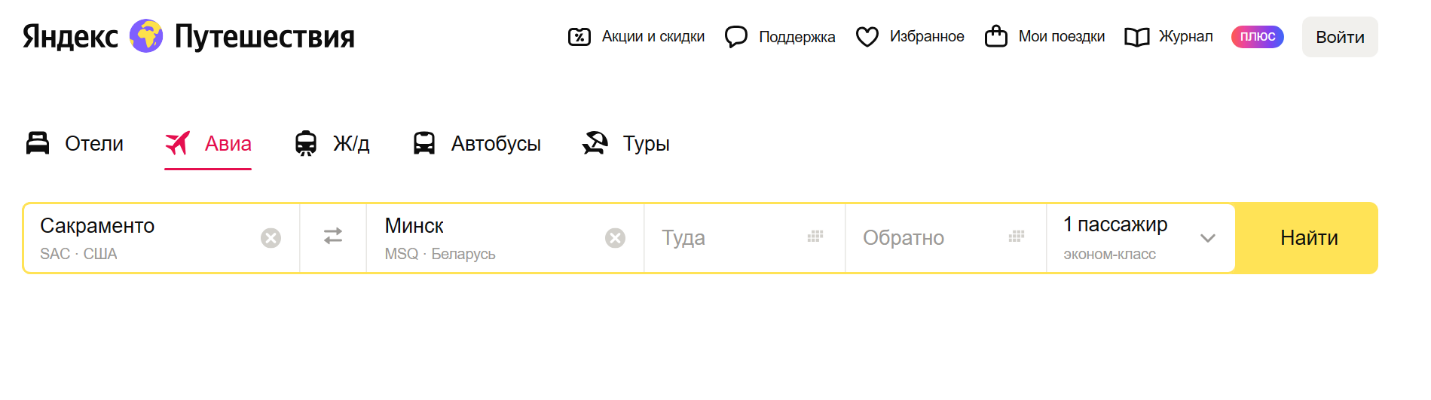
Travel.yandex.ru[2]— это российский сервис, который помогает планировать путешествия.

Рисунок 1.2 – Главная страница сайта travel.yandex.ru

Ключевые особенности Travel.yandex.ru:

– Тонкая настройка путешествия: Сервис позволяет не только найти и забронировать билеты и жилье, но и спланировать маршрут, выбрать экскурсии, арендовать автомобиль и многое другое.

– Различные фильтры и категории для удобного поиска: Пользователи могут искать билеты и отели по различным параметрам, например, по авиакомпании, классу обслуживания, типу отеля, наличию определенных удобств.

– Выбор и сравнение билетов: Сервис сравнивает цены на билеты от разных авиакомпаний и агентств, помогая найти наиболее выгодное предложение.

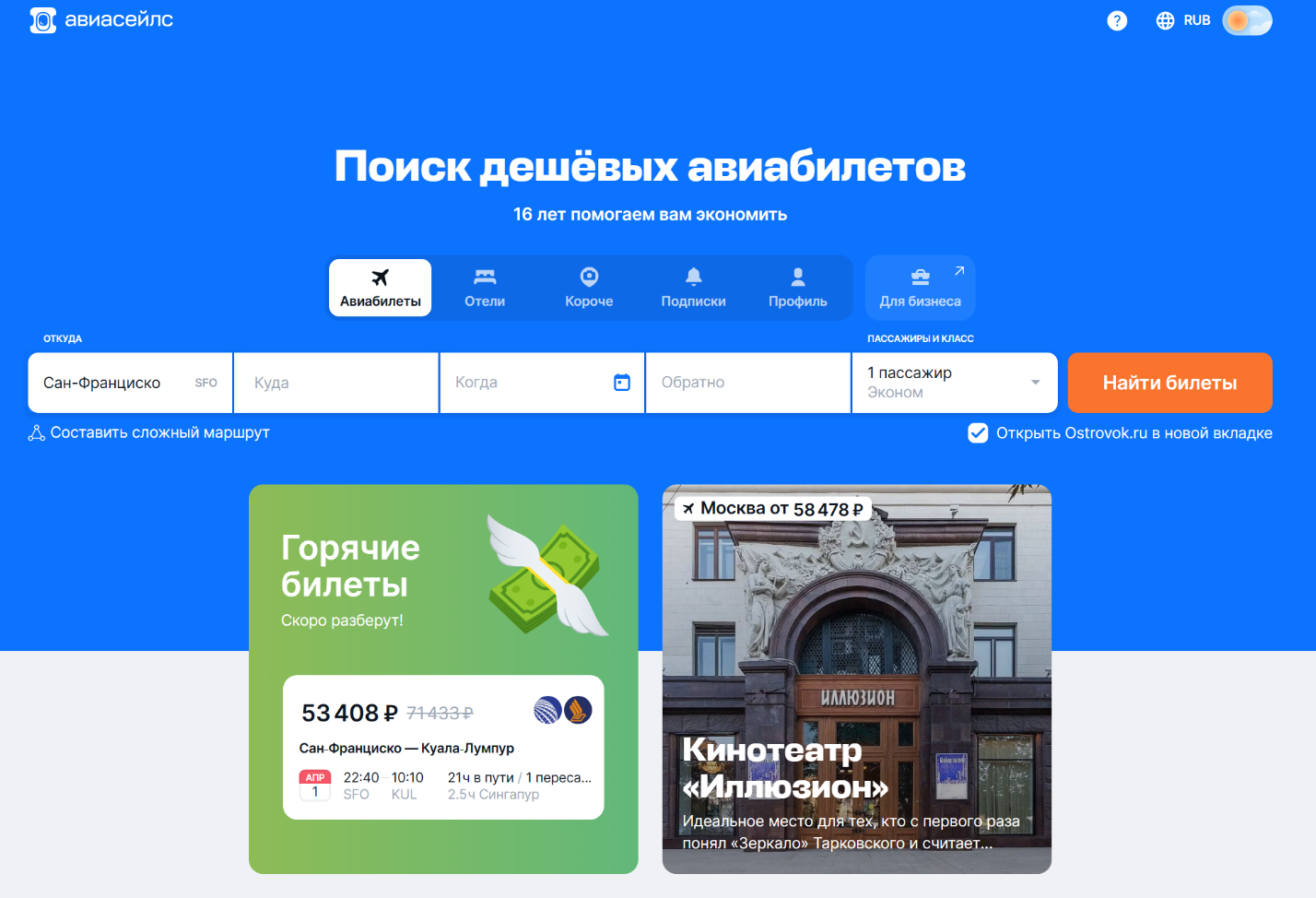
– Информация о ценах и наличии услуг: Пользователи получают актуальную информацию о стоимости билетов и наличию мест на выбранные даты.

– Вкладка «Избранное»: Позволяет сохранять понравившиеся варианты и быстро возвращаться к ним.

– Личный кабинет: В личном кабинете хранится история заказов, настройки пользователя и другая полезная информация.

### 1.1.3 **Аналог «Aviasales»**

Aviasales.ru[3]– сервис, специализирующийся на поиске и бронировании авиабилетов.

  
Рисунок 1.3 – Главная страница сайта aviasales.ru

Ключевые особенности Aviasales.ru:

– Подбор и бронирование билетов: Сервис предоставляет доступ к базе данных авиакомпаний и агентств, предлагая широкий выбор билетов на различные направления.

– Информация о ценах и наличии услуг: Пользователи могут сравнивать цены, выбирать удобные даты и получать информацию о наличии мест на рейсах.

– Личный кабинет: В личном кабинете хранится история заказов, настройки пользователя и другая полезная информация.

Анализ аналогичных решений позволяет нам выделить ключевые функции, которые могут быть включены в наш сервис, а также выявить лучшие практики для обеспечения удовлетворения потребностей клиентов и эффективного функционирования бизнеса

## 1.2 Разработка функциональных требований

Помимо основных требований, выделенных еще при постановке задания к курсовому проекту по данной дисциплине, после тщательного анализа, можно выделить обязательный функционал для БД, который нужен будет для осуществления выбранной темы:

– Таблицы с полями, которые максимально точно будут описывать все необходимы характеристики для любой категории компонентов

– Возможность поиска данных

– Сортировка данных

– Реализация корзины и оформления билетов

– Реализация действий администратора: добавление, изменение, удаления рейсов, билетов, самолётов, пассажиров, сотрудников.

– Реализация действий клиента: просмотр, покупка, оформление, возврат билета

## 1.3 Вывод

В результате анализа альтернативных решений для поставленной задачи, была получена чёткая картина финального продукта. Так же нам это помогло расширить наш проект и понять, какие функции должны быть в нём.

# 2. Разработка архитектуры проекта

## 2.1 Структура управлением приложения

Для реализации сервиса авиакомпании с использованием Microsoft SQL Server, мы разрабатываем структуру управления приложения, которая будет эффективно взаимодействовать с базой данных и обеспечивать оптимальную работу сервиса. Важным аспектом этой структуры является использование технологии Analysis Services для анализа всех данных в таблицах, повышая эффективность бизнеса.

Для реализации нашего проекта мы выбрали СУБД Microsoft SQL Server, которая обладает широкими возможностями для проектирования, создания и управления базами данных. Мы используем SQL Server для разработки и оптимизации схемы базы данных, создания таблиц для хранения информации о товарах, клиентах, заказах и других аспектах нашего интернет–магазина.

SQL Server обеспечивает высокую производительность и надежность, что критически важно для эффективной работы сервиса и обработки больших объемов данных, связанных с полётами, клиентами и билетами.

Для обеспечения клиентов дополнительной информацией, мы реализуем механизм Analysis Services. Данный механизм будет анализировать данные в базе, что хорошо скажется на эффективности бизнеса.

## 2.2 Диаграммы UML, взаимосвязь всех компонентов.

Диаграмма UML базы данных можно увидеть на рисунке 2.1

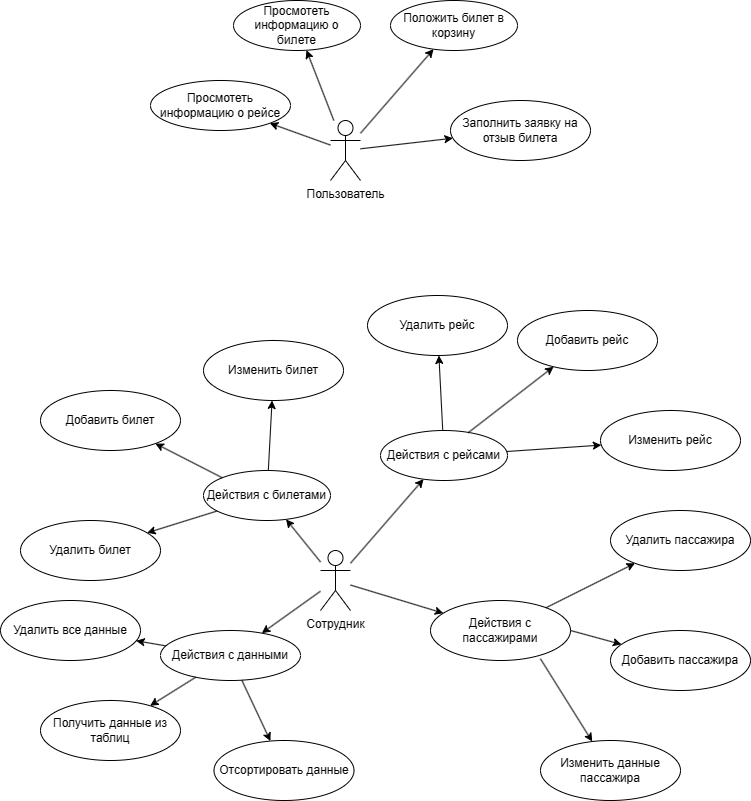


Рисунок 2.1 – Диаграмма использования

Схема базы данных ограничения целостности, связи и поля представлены на рисунке 2.2.

## 2.3 Диаграмма БД, взаимосвязь компонентов. Описание объектов и ограничение целостности

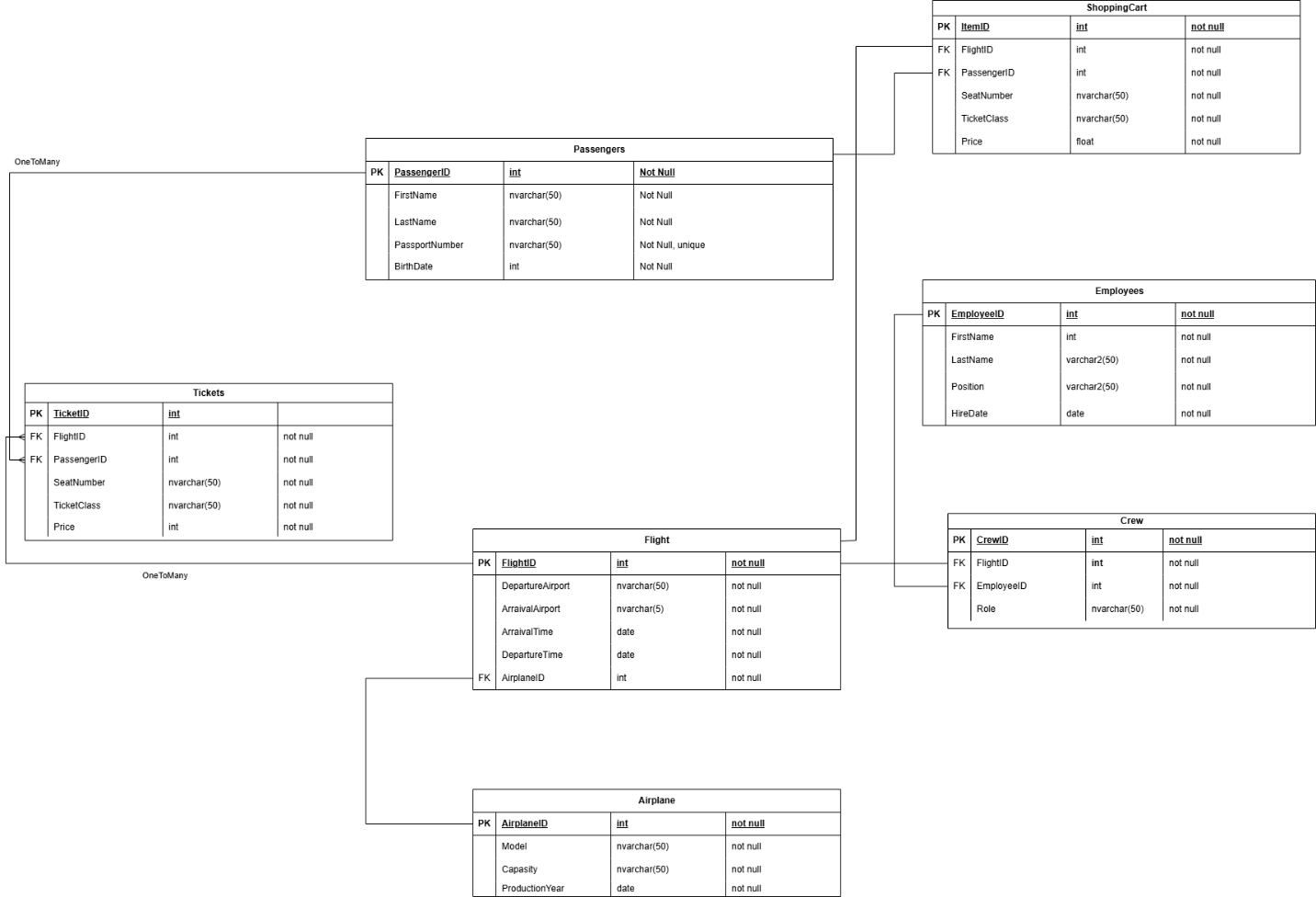


Рисунок 2.2 – Схема БД проекта

Принимая во внимание то, нужно обеспечить продукт всем необходимым функционалом и сделать это максимально эффективно была выбрана схема базы данных, показанная на рисунке 2.2.

## **2.4 Описание информационных объектов и ограничений целостности.**

### 2.4.1 Passengers (Пассажиры):

– PassengerID (Идентификатор пассажира): Уникальный идентификатор для каждого пассажира.

– FirstName (Имя): Имя пассажира.

– Lastname (Фамилия): Фамилия пассажира.

– PassportNumber (Номер паспорта): Номер паспорта пассажира для идентификации и оформления билетов.

– Birthdate (Дата рождения): Дата рождения пассажира, может использоваться для проверки возраста и предоставления скидок.

### 2.4.2 Tickets (Билеты):

– TicketID (Идентификатор билета): Уникальный идентификатор для каждого билета.

– FlightID (Идентификатор рейса): Связь с таблицей Flights для определения рейса, на который куплен билет.

– PassengerID (Идентификатор пассажира): Связь с таблицей Passengers для определения пассажира, которому принадлежит билет.

– SeatNumber (Номер места): Номер места пассажира в самолете.

– TicketClass (Класс билета): Класс билета (эконом, бизнес и т.д.), определяет уровень комфорта и услуг.

– Price (Цена): Цена билета.

### 2.4.3 ShoppingCart (Корзина):

– ItemID (Идентификатор товара): Уникальный идентификатор для каждого элемента в корзине.

– FlightID (Идентификатор рейса): Связь с таблицей Flights для определения рейса, на который выбирается билет.

– PassengerID (Идентификатор пассажира): Связь с таблицей Passengers для определения пассажира, которому принадлежит корзина.

– SeatNumber (Номер места): Номер места, которое пассажир выбрал в самолете.

– TicketClass (Класс билета): Класс билета (эконом, бизнес и т.д.), который выбирает пассажир.

– Price (Цена): Цена выбранного билета.

### 2.4.4 Flights (Рейсы):

– FlightID (Идентификатор рейса): Уникальный идентификатор для каждого рейса.

– DepartureAirport (Аэропорт вылета): Аэропорт, из которого вылетает рейс.

– ArrivalAirport (Аэропорт прибытия): Аэропорт, в который прибывает рейс.

– DepartureTime (Время вылета): Время вылета рейса.

– ArrivalTime (Время прибытия): Время прибытия рейса.

– AirplaneID (Идентификатор самолета): Связь с таблицей Airplanes для определения самолета, который выполняет рейс.

### 2.4.5 Airplanes (Самолеты):

– AirplaneID (Идентификатор самолета): Уникальный идентификатор для каждого самолета.

– Model (Модель): Модель самолета.

– Capacity (Вместимость): Количество пассажиров, которое может вместить самолет.

– ProductionYear (Год выпуска): Год выпуска самолета.

### 2.4.6 Employees (Сотрудники):

– EmployeeID (Идентификатор сотрудника): Уникальный идентификатор для каждого сотрудника.

– FirstName (Имя): Имя сотрудника.

– Lastname (Фамилия): Фамилия сотрудника.

– Position (Должность): Должность сотрудника в компании.

– HireDate (Дата найма): Дата, когда сотрудник был принят на работу.

### 2.4.7 Crews (Экипажи):

– CrewID (Идентификатор экипажа): Уникальный идентификатор для каждого экипажа.

– FlightID (Идентификатор рейса): Связь с таблицей Flights для определения рейса, который обслуживает экипаж.

– EmployeeID (Идентификатор сотрудника): Связь с таблицей Employees для определения сотрудников, которые входят в экипаж.

– Role (Роль): Роль сотрудника в экипаже (пилот, бортпроводник и т.д.).

## **2.5 Вывод по разделу.**

В данном разделе была разработана архитектура проекта, составлена обобщенная структура управления приложением и диаграмма UML. Были описаны все таблицы базы данных и ограничения целостности к строкам таблицы.

## **3. Разработка модели базы данных.**Создание необходимых объектов.

## 3.1 Таблицы

Таблица — это совокупность связанных данных, хранящихся в структурированном виде в базе данных. Она состоит из столбцов и строк.

База данных данного курсового проекта содержит 7 таблиц, которые описаны в главе 2.

## 3.2 Процедуры

Процедуры — это именованные блоки кода на языке PL/pgSQL, которые хранятся в базе данных и могут быть вызваны при необходимости. Они используются для группировки и переиспользования кода, который должен выполняться несколько раз, а также для уменьшения нагрузки на сеть при обращении к базе данных. Процедуры могут содержать в себе SQL–запросы, циклы, условные операторы, переменные и многое другое.

Пример создания хранимой процедуры GetFlightsByDate, которая выводит список рейсов в определённое время, представлена в листинге 3.1

|  |
| --- |
| CREATE PROCEDURE GetFlightsByDate  @Date DATE  AS  BEGIN  SELECT \* FROM Flights  WHERE CAST(DepartureTime AS DATE) = @Date;  END;  GO |

Листинг 3.1 – Процедура GetFlightsByDate

### 3.2.1 Самолёты:

– AddAirplane: Добавляет новый самолет в базу данных.

– UpdateAirplane: Обновляет информацию о существующем самолете.

– DeleteAirplane: Удаляет самолет из базы данных.

– GetAirplaneByID: Получает информацию о самолете по его ID.

– GetAllAirplanes: Получает список всех самолетов.

– GetAirplanesByModel: Получает список самолетов определенной модели.

– AddRandomAirplane: Добавляет случайный самолет с генерируемыми характеристиками.

– DeleteAllAirplanes: Удаляет все самолеты из базы данных.

### 3.2.2 Рейсы:

– AddFlight: Добавляет новый рейс в базу данных.

– UpdateFlight: Обновляет информацию о существующем рейсе.

– DeleteFlight: Удаляет рейс из базы данных.

– GetFlightByID: Получает информацию о рейсе по его ID.

– GetAllFlights: Получает список всех рейсов.

– GetFlightsByAirport: Получает список рейсов, связанных с определенным аэропортом (вылет или прилет).

– GetFlightsByDate: Получает список рейсов на определенную дату.

– AddRandomFlight: Добавляет случайный рейс с генерируемыми характеристиками.

– DeleteAllFlights: Удаляет все рейсы из базы данных.

– AddToShoppingCart: Добавляет выбранный билет в корзину.

– ProcessTicket: Оформляет билет, перемещая данные из корзины в таблицу билетов.

### 3.2.3 Пассажиры:

– AddPassenger: Добавляет нового пассажира в базу данных.

– UpdatePassenger: Обновляет информацию о существующем пассажире.

– DeletePassenger: Удаляет пассажира из базы данных.

– GetPassengerByID: Получает информацию о пассажире по его ID.

– GetAllPassengers: Получает список всех пассажиров.

– GetPassengerByPassport: Получает информацию о пассажире по номеру паспорта.

– AddRandomPassenger: Добавляет случайного пассажира с генерируемыми характеристиками.

– DeleteAllPassengers: Удаляет всех пассажиров из базы данных.

### 3.2.4 Билеты:

– AddTicket: Добавляет новый билет в базу данных.

– UpdateTicket: Обновляет информацию о существующем билете.

– DeleteTicket: Удаляет билет из базы данных.

– GetTicketByID: Получает информацию о билете по его ID.

– GetAllTickets: Получает список всех билетов.

– GetTicketsByFlight: Получает список билетов на определенный рейс.

– GetTicketsByPassenger: Получает список билетов, купленных определенным пассажиром.

– AddRandomTicket: Добавляет случайный билет с генерируемыми характеристиками.

### 3.2.5 Сотрудники:

– AddEmployee: Добавляет нового сотрудника в базу данных.

– UpdateEmployee: Обновляет информацию о существующем сотруднике.

– DeleteEmployee: Удаляет сотрудника из базы данных.

– GetEmployeeByID: Получает информацию о сотруднике по его ID.

– GetAllEmployees: Получает список всех сотрудников.

– GetEmployeesByPosition: Получает список сотрудников определенной должности.

– AddRandomEmployee: Добавляет случайного сотрудника с генерируемыми характеристиками.

### 3.2.6 Экипаж:

– AddCrewMember: Добавляет члена экипажа к рейсу.

– UpdateCrewMember: Обновляет роль члена экипажа.

– DeleteCrewMember: Удаляет члена экипажа из рейса.

GetCrewByFlight: Получает список экипажа для рейса.

– AddRandomCrew: Добавляет случайный экипаж к рейсу.

## 3.3 Функции:

Пример одной из разработанных функций GetFlightsDuration, которая вычисляет время полёта, представлена в листинге 3.2

|  |
| --- |
| CREATE FUNCTION GetFlightDuration (@FlightID INT)  RETURNS DECIMAL(5, 2)  AS  BEGIN  DECLARE @Duration DECIMAL(5, 2);  SELECT @Duration = DATEDIFF(MINUTE, DepartureTime, ArrivalTime) / 60.0  FROM Flights  WHERE FlightID = @FlightID;  RETURN @Duration;  END;  GO |

Листинг 3.2 – Процедура GetFlightsDuration

### 3.3.1 Самолёты:

– GetAirplaneAge: Вычисляет возраст самолета в годах.

– IsAirplaneAvailable: Проверяет доступность самолета в заданный период времени.

– GetTotalCapacity: Вычисляет общую вместимость всех самолетов.

– GetAverageAirplaneAge: Вычисляет средний возраст всех самолетов.

### 3.3.2 Рейсы:

– GetFlightDuration: Вычисляет продолжительность рейса в часах.

– AreSeatsAvailable: Проверяет наличие свободных мест на рейсе в заданном классе.

### 3.3.3 Пассажиры:

– GetPassengerFullName: Получает полное имя пассажира (имя и фамилия).

– IsPassengerAdult: Проверяет, является ли пассажир совершеннолетним.

### 3.3.4 Билеты:

– GetTotalTicketsSold: Получает общее количество проданных билетов.

– GetRevenueByFlight: Получает общую выручку от продажи билетов на определенный рейс.

– GetMostPopularDestination: Получает аэропорт назначения, в который было продано больше всего билетов.

– GetAverageTicketPrice: Получает среднюю цену билета.

### 3.3.5 Сотрудники:

– GetEmployeeExperience: Получает стаж работы сотрудника в годах.

– GetTotalEmployeesByPosition: Получает количество сотрудников определенной должности.

### 3.3.6 Экипаж:

– IsCrewComplete: Проверяет, укомплектован ли экипаж для рейса.

Общие:

– SortTable: Сортирует выбранную таблицу по выбранному столбцу в нужном порядке

– TotalPurge: Удаляет все данные из всех таблиц

## 3.4 Триггеры:

Пример одного разработанного триггера tr\_Airplanes\_Insert, который выводит сообщение о добавлении самолёта, представлен в листинге 3.3

|  |
| --- |
| CREATE TRIGGER tr\_Airplanes\_Insert  ON Airplanes  AFTER INSERT  AS  BEGIN  PRINT 'New airplane added to the database.';    END;  GO |

Листинг 3.3 – Процедура GetFlightsDuration

### 3.4.1 Самолёты:

– tr\_Airplanes\_Insert: Выводит сообщение при добавлении нового самолета в базу данных.

– tr\_Airplanes\_Update: Выводит сообщение при обновлении информации о самолете.

– tr\_Airplanes\_Delete: Выводит сообщение при удалении самолета из базы данных.

### 3.4.2 Рейсы:

– tr\_Flights\_Insert: Выводит сообщение при добавлении нового рейса в расписание.

– tr\_Flights\_Update: Выводит сообщение при обновлении информации о рейсе.

– tr\_Flights\_Delete: Выводит сообщение при удалении рейса из расписания.

### 3.4.3 Пассажиры:

– tr\_Passengers\_Insert: Выводит сообщение при регистрации нового пассажира.

– tr\_Passengers\_Update: Выводит сообщение при обновлении информации о пассажире.

– tr\_Passengers\_Delete: Выводит сообщение при удалении данных пассажира.

### 3.4.4 Билеты:

– tr\_Tickets\_Insert: Выводит сообщение при покупке нового билета.

– tr\_Tickets\_Update: Выводит сообщение при обновлении информации о билете.

– tr\_Tickets\_Delete: Выводит сообщение при отмене билета.

### 3.4.5 Сотрудники:

– tr\_Employees\_Insert: Выводит сообщение при найме нового сотрудника.

– tr\_Employees\_Update: Выводит сообщение при обновлении информации о сотруднике.

– tr\_Employees\_Delete: Выводит сообщение при удалении данных сотрудника.

### 3.4.6 Экипаж:

– tr\_Crews\_Insert: Выводит сообщение при назначении члена экипажа на рейс.

– tr\_Crews\_Update: Выводит сообщение при обновлении роли члена экипажа.

– tr\_Crews\_Delete: Выводит сообщение при удалении члена экипажа из рейса.

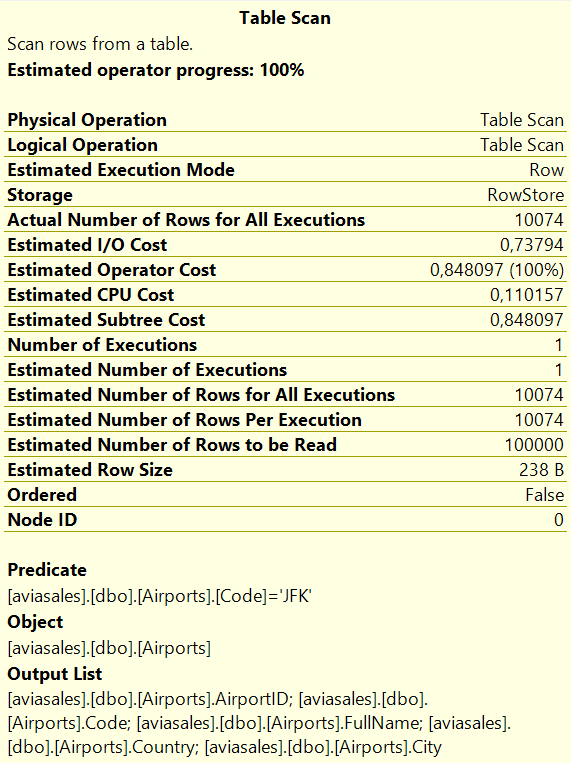
## 3.5 Индексы

Был разработан индекс для ускорения поиска по коду аэропортов, индекс представлен в листинге 3.4

|  |
| --- |
| CREATE CLUSTERED INDEX IX\_Airports\_Code ON dbo.Airports (Code); |

Листинг 3.4 – Процедура GetFlightsDuration

Индекс применяется в таблице, в которой находится 100 000 строк, попробуем сделать поиск по коду аэропорта без индекса:

Рисунок 3.1 – Стоимость запроса без индекса

Попробуем сделать тот же запрос, создав индекс:

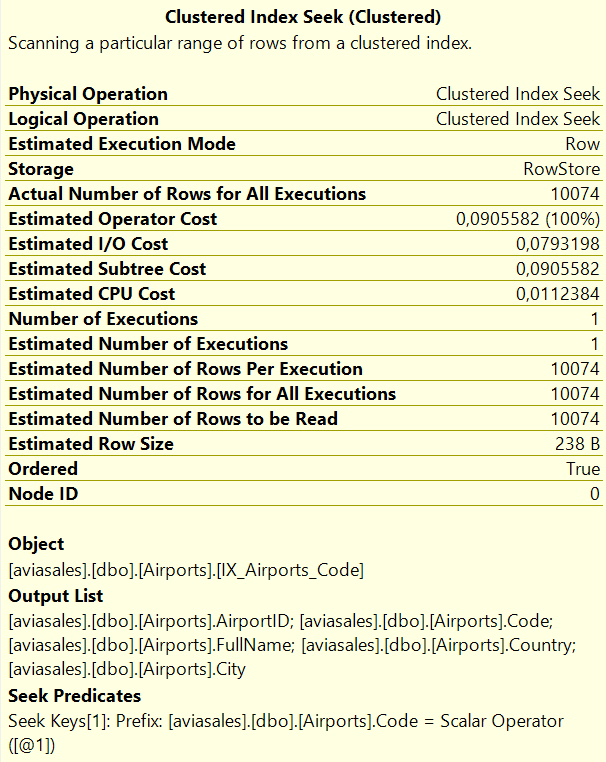


Рисунок 3.2 – Стоимость запроса с индексом

Как мы можем видеть, цена запроса уменьшилась практически в 10 раз.

# 4. Установка, настройка и использование Oracle.

## Microsoft SQL Server — это реляционная система управления базами данных (СУБД), разработанная и продвигаемая компанией Microsoft. Она также является одной из самых популярных и мощных баз данных на рынке. SQL Server предоставляет широкий набор функций для хранения, организации и обработки данных, а также для управления доступом к ним.

## 4.1. Создание ролей для разграничения доступа.

Роли в базе данных MSSQL представляют собой набор привилегий, которые могут быть присвоены пользователям или другим ролям. Используя роли, можно управлять доступом к объектам базы данных и определять различные уровни привилегий для пользователей. Применение ролей упрощает администрирование базы данных, так как позволяет определить права доступа один раз и затем присваивать их группам пользователей. Это делает процесс управления привилегиями более гибким и эффективным.

Основная цель ролей - обеспечение безопасности данных. Они позволяют строго контролировать доступ к чувствительной информации и предотвращать несанкционированный доступ.

В рамках курсового проекта было разработано две роли:

1.dbowner - это роль с полным доступом к базе данных aviasales. Она может выполнять все действия, включая создание, изменение и удаление объектов, а также управление пользователями.

2.customer - это роль с ограниченным доступом, которая может выполнять следующие действия:

1.AddToShoppingCart: Добавить билет в корзину пользователя.

2.GetAllCartData: Получить все данные о билетах в корзине пользователя.

3.ProcessTicket: Обработать (оплатить) билет, перемещая его из корзины в таблицу Tickets.

4.GetTicketByID: Получить информацию о билете по его идентификатору.

5.GetFlightByID: Получить информацию о рейсе по его идентификатору.

Важно отметить, что:

1.customer не может добавлять новых рейсов, самолетов, пассажиров или сотрудников.

2.customer не может удалять или изменять данные в таблицах, кроме своей корзины.

customer не может управлять пользователями.

Пример создания роли продемонстрирован в листинге 4.1.

|  |
| --- |
| CREATE ROLE customer;  GRANT EXECUTE ON AddToShoppingCart TO customer;  GRANT EXECUTE ON GetAllCartData TO customer;  GRANT EXECUTE ON ProcessTicket TO customer;  GRANT EXECUTE ON GetTicketByID TO customer;  GRANT EXECUTE ON GetFlightByID TO customer;  GRANT EXECUTE ON AddToShoppingCart TO customer;  GRANT EXECUTE ON GetAllCartData TO customer; |

Листинг 4.1 – Пример создания роли

Эта роль наделяется правами на просмотр своих данных, разграничивая пользователей от ненужных им данных.

## 4.2. Описание процедур общего назначения.

Для облегчения использования базы данных были добавлены некоторые процедуры общего назначения

|  |
| --- |
| CREATE PROCEDURE TotalPurge  AS  BEGIN  DBCC CHECKIDENT ('Tickets', RESEED, 0);  DELETE FROM Tickets  DBCC CHECKIDENT ('Crews', RESEED, 0);  DELETE FROM Crews  DBCC CHECKIDENT ('Flights', RESEED, 0);  DELETE FROM Flights  DBCC CHECKIDENT ('Employees', RESEED, 0);  DELETE FROM Employees  DBCC CHECKIDENT ('ShoppingCart', RESEED, 0);  DELETE FROM ShoppingCart  DBCC CHECKIDENT ('Airplanes', RESEED, 0);  DELETE FROM Airplanes  DBCC CHECKIDENT ('Passengers', RESEED, 0);  DELETE FROM Passengers;  END; |

Листинг 4.1 – Пример процедуры удаления данных из бд

Процедурa SortTable нужна для сортировки таблиц. Процедура принимает в себя 3 параметра: имя таблицы, столбец для сортировки и порядок сортировки. Таблица в результирующем запросе будет отсортирована по заданным критериям.

|  |
| --- |
| CREATE PROCEDURE SortTable  (  @TableName NVARCHAR(128), -- Имя таблицы  @ColumnName NVARCHAR(128), -- Имя столбца для сортировки  @SortOrder NVARCHAR(4) = 'ASC' -- Порядок сортировки (ASC или DESC)  )  AS  BEGIN  -- Проверка наличия таблицы  IF NOT EXISTS (SELECT \* FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_NAME = @TableName)  BEGIN  PRINT 'Ошибка: Таблица "' + @TableName + '" не найдена.';  RETURN;  END;  -- Проверка наличия столбца  IF NOT EXISTS (SELECT \* FROM INFORMATION\_SCHEMA.COLUMNS WHERE TABLE\_NAME = @TableName AND COLUMN\_NAME = @ColumnName)  BEGIN  PRINT 'Ошибка: Столбец "' + @ColumnName + '" не найден в таблице "' + @TableName + '".';  RETURN;  END;  -- Проверка порядка сортировки  IF @SortOrder NOT IN ('ASC', 'DESC')  BEGIN  PRINT 'Ошибка: Неверный порядок сортировки. Допустимые значения: ASC, DESC.';  RETURN;  END;  -- Построение динамического SQL-запроса  DECLARE @sql NVARCHAR(MAX);  SET @sql = N'SELECT \* FROM ' + QUOTENAME(@TableName) + N' ORDER BY ' + QUOTENAME(@ColumnName) + N' ' + @SortOrder;  -- Выполнение динамического SQL-запроса  EXEC sp\_executesql @sql;  END; |

Листинг 4.2 – Пример создания процедуры экспорта

## 4.3. Тестирование производительности базы данных.

Для тестирования нагрузки базы данных была создана таблица Airports, куда было добавлено 100 000 строк.

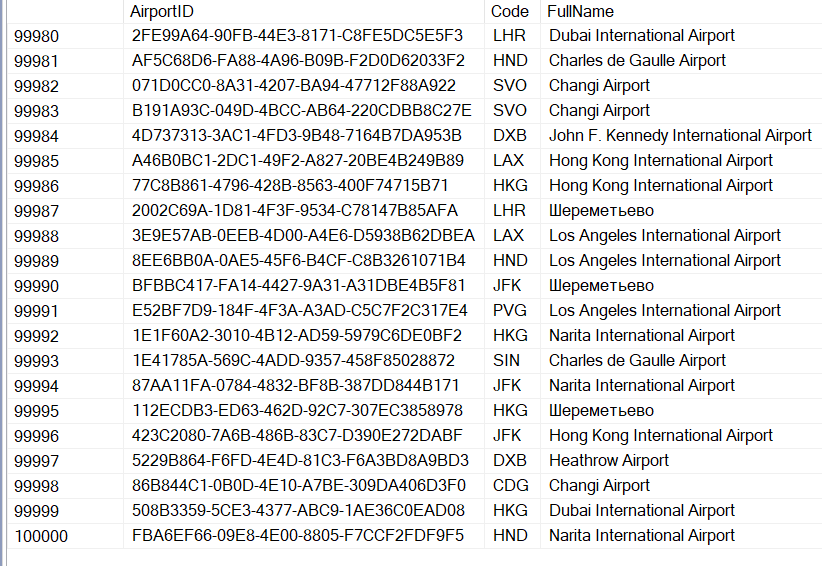


Рисунок 4.1 – 100 000 строк в таблице

Так же в разделе 3.5 было подробно расписана работа индекса и разница между стоимостью запроса.

## 4.4. Вывод.

В этом разделе была описана выбранная СУБД, рассказано о разработанных процедурах, триггерах, функциях и индексах. Также была проверена работоспособность разработанной БД посредством добавления в одну из таблиц 100000 строк.

# 5. Описание используемой технологии.

Технология Analysis Services представляет собой аналитический движок для комплексной обработки многомерных данных в реляционных и многомерных хранилищах данных. Это средство предоставляет бизнес–аналитику, интеллектуальные функции, а также возможность создавать многомерные структуры, содержащие агрегированные данные из реляционных источников для быстрого извлечения и анализа.

Установка Analysis Services:

1. Analysis Services является компонентом Microsoft SQL Server, поэтому необходимо в настройках установки SQL Server включить Analysis Service

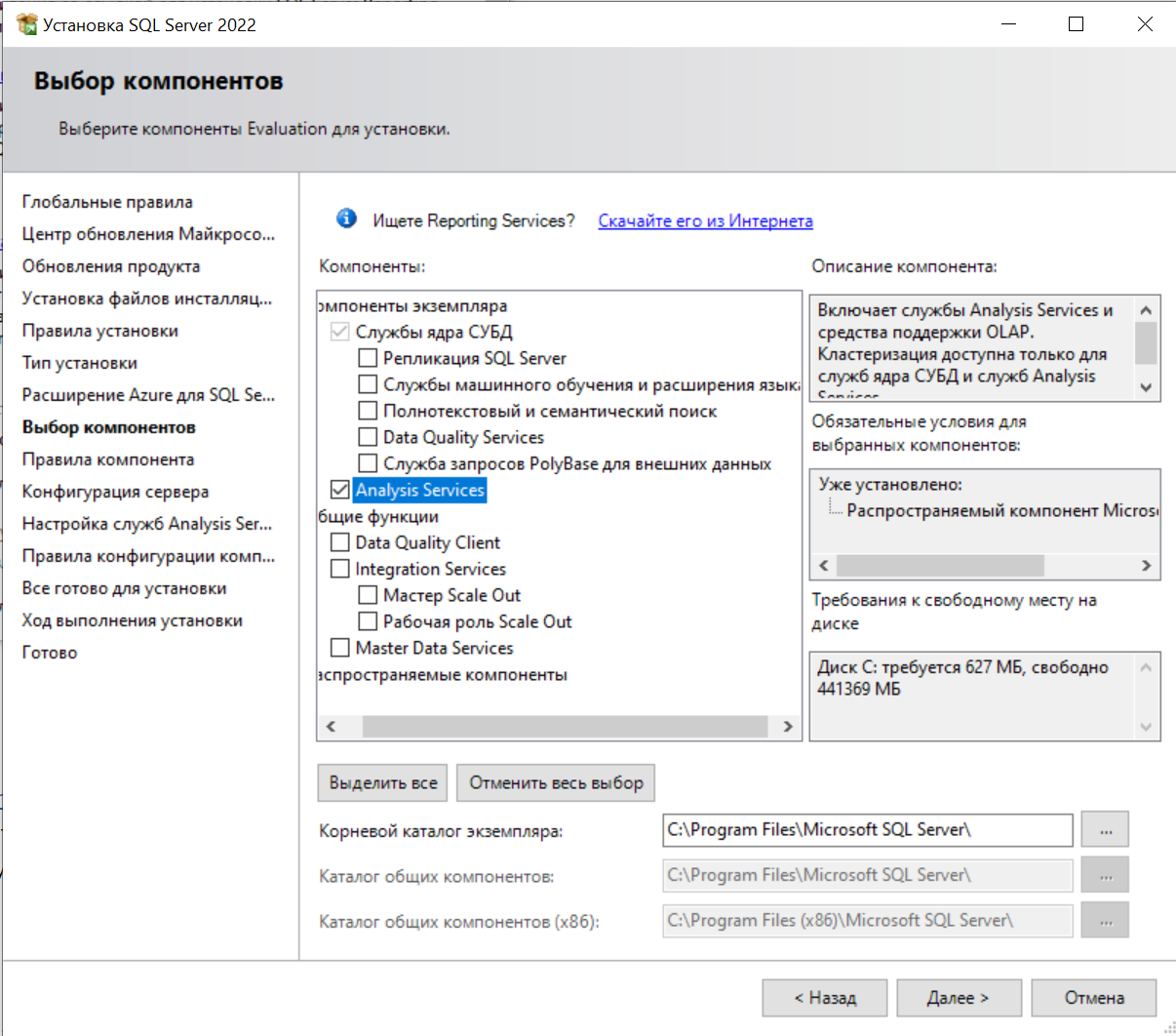


Рисунок 5.1 – Установка Analysis Service

2. После завершения установки Analysis Services будет доступен в составе SQL Server.

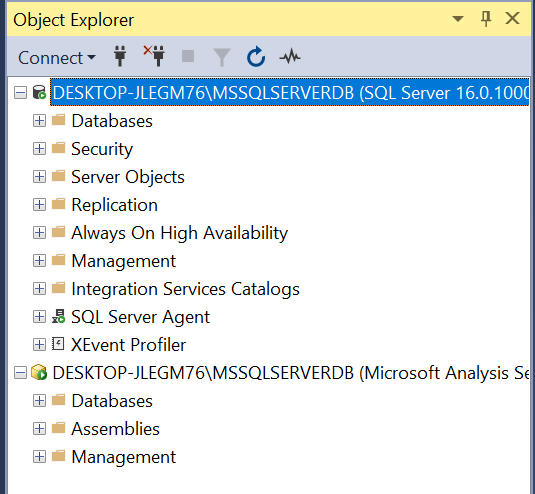


Рисунок 5.2 –Analysis Service

Работа с Analysis Services:

Для управления Analysis Services будет использоваться Visual Studio с установленными расширениями

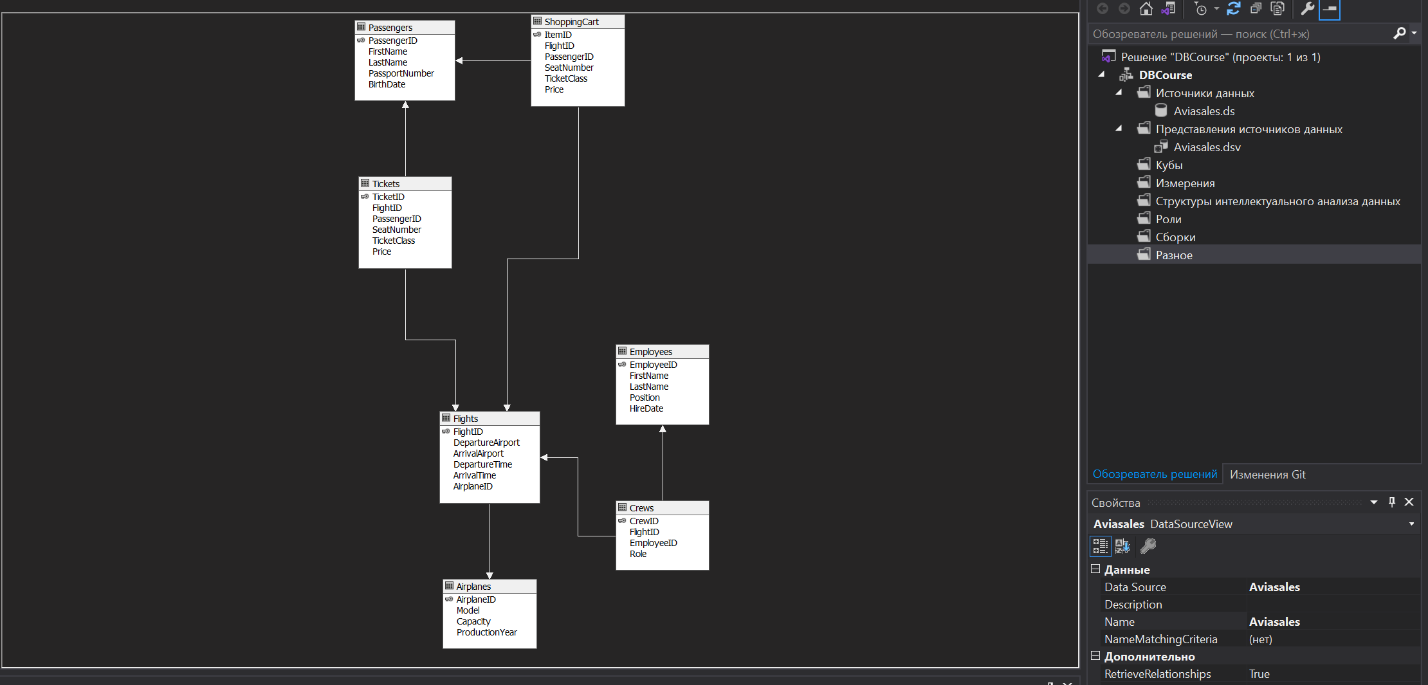


Рисунок 5.3 –Analysis Service в Visual Studio

### 5.1 Измерения

Измерения представляют собой иерархически организованные категории или атрибуты данных, используемые для фильтрации и анализа данных. Каждое измерение состоит из одного или нескольких уровней, которые образуют структуру иерархии.

Измерения позволяют пользователям просматривать данные с разных углов зрения, изменяя уровни детализации. Они могут анализировать показатели на высоком уровне, например по годам, или переходить к более детальному анализу по месяцам или дням. Измерения в Analysis Services тесно связаны с концепцией иерархий и являются её неотъемлемой частью.

### 5.2 Иерархии

Иерархия состоит из нескольких уровней, организованных в порядке от более общих к более детальным. Каждый уровень представляет собой консолидированное представление данных уровня ниже.

Иерархии обеспечивают логическую структуру для просмотра и анализа данных с разными уровнями детализации. Пользователи могут легко переключаться между уровнями для получения нужной степени обобщения или детализации.

При разработке многомерных моделей в Analysis Services определение правильных иерархий для каждого измерения имеет важное значение для обеспечения полезного и интуитивно понятного анализа данных в бизнес–приложениях.

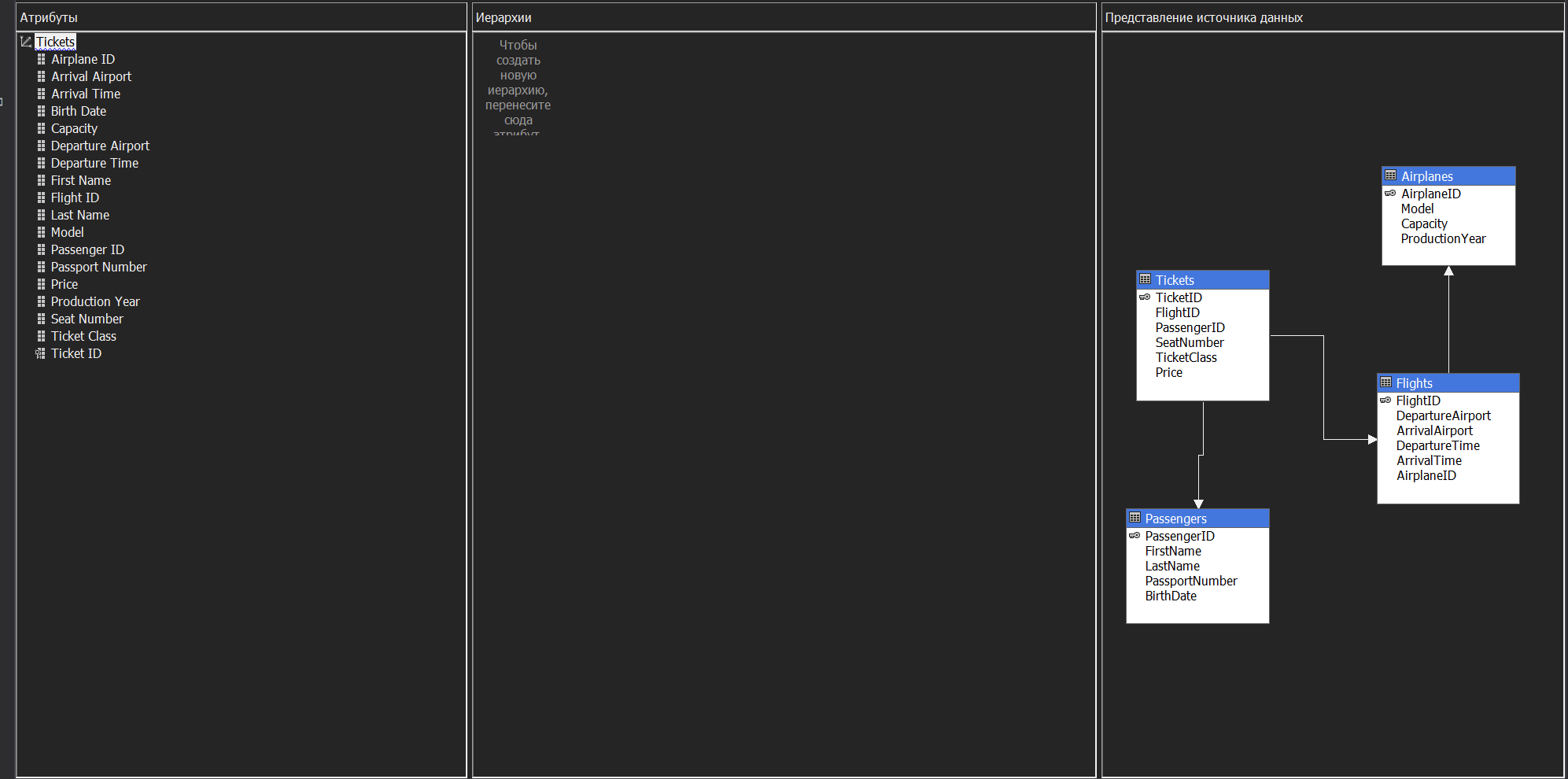


Рисунок 5.4 – Представления и иерархии

После создания представления можно создать иерархии:

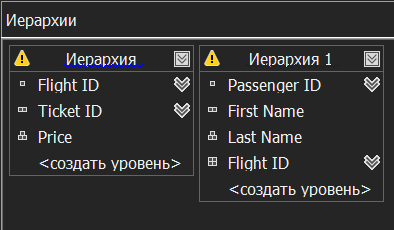


Рисунок 5.5 – Иерархии

С помощью данных иерархий можно облегчить поиск или сортировку данных, посмотрим как это будет выглядеть после обработки строк и развёртывания в встроенном браузере:

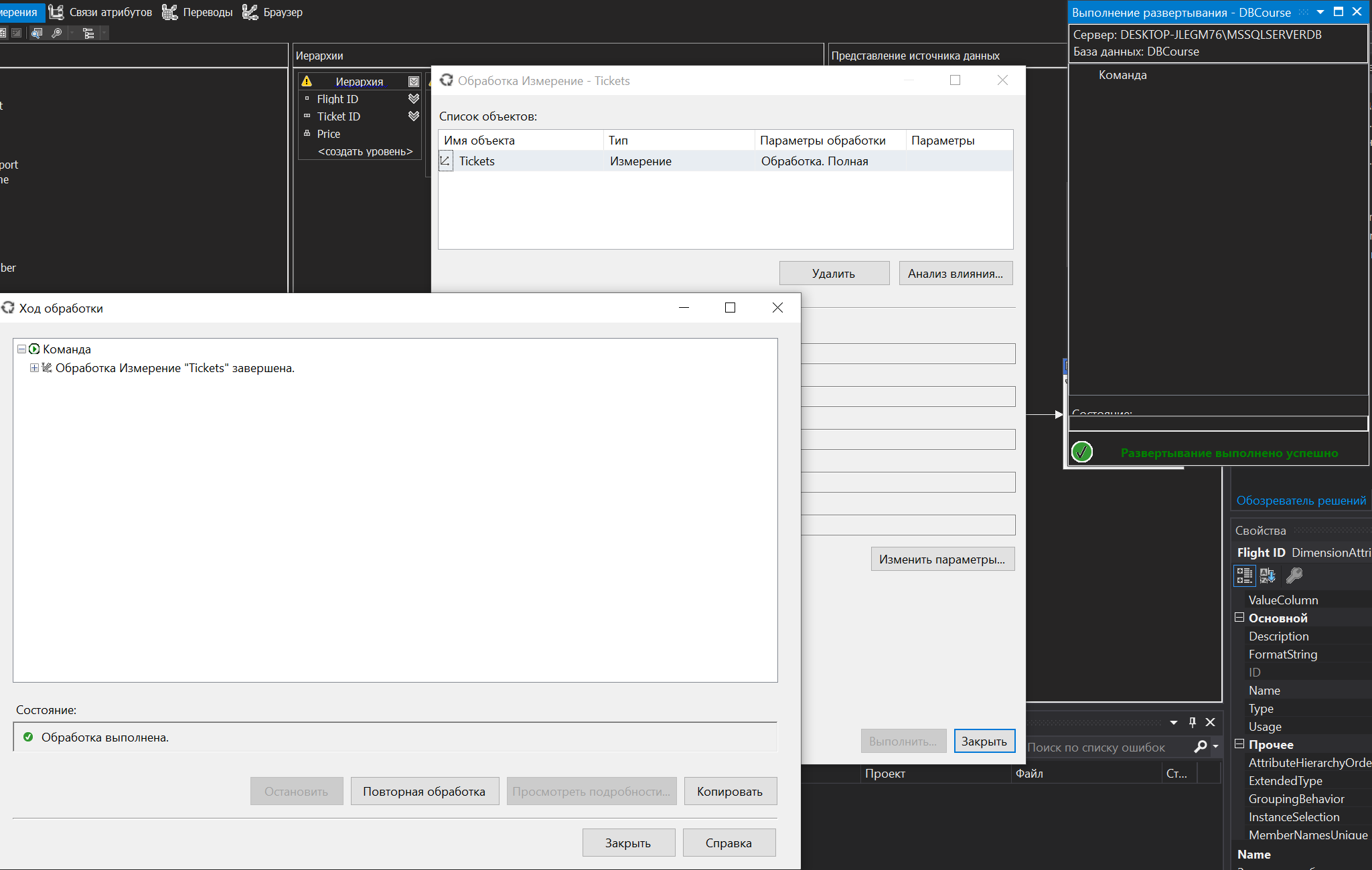


Рисунок 5.6 – Развёртывание и публикация

Мы получили иерархию, в которой можно посмотреть ID рейса, увидеть все связанные с ним билеты и посмотреть их цену:

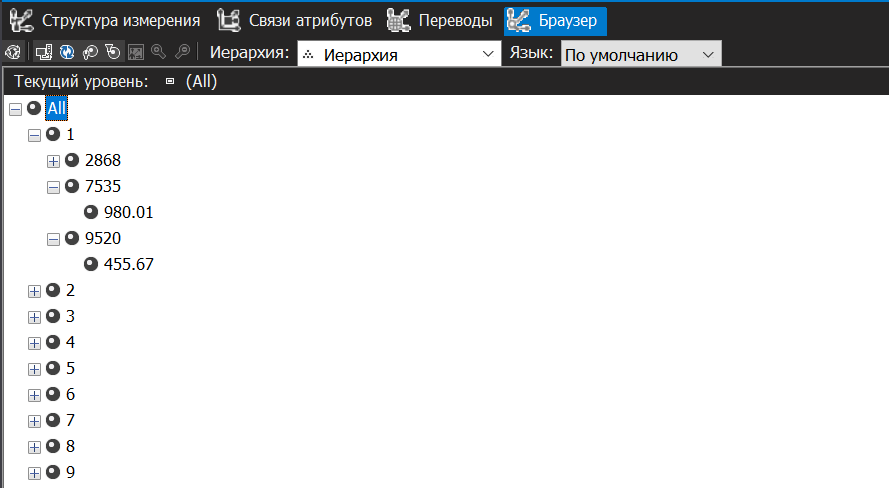


Рисунок 5.7 – Иерархия

### 5.3 Кубы

После создания и настройки представлений источников данных они используются для построения многомерного куба или табличной модели. Куб представляет собой многомерную базу данных, состоящую из измерений, групп мер и партиций.

При создании куба вы указываете представления источников данных, из которых будут извлекаться данные для измерений и мер. Таким образом, куб становится логическим представлением данных на основе нескольких реляционных таблиц, объединенных и преобразованных через представления источников данных.

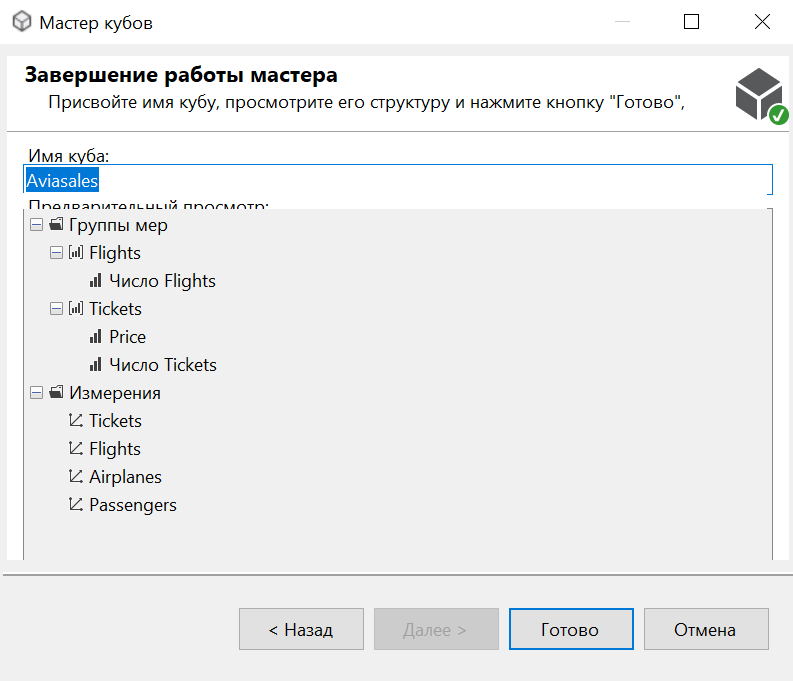
Преимущество использования представлений источников данных заключается в том, что они предоставляют единое централизованное представление данных для многомерного куба, абстрагируя его от сложной реляционной структуры базы данных. Это упрощает создание и управление кубами, поскольку логика объединения, фильтрации и преобразования данных инкапсулирована в представлениях.

Рисунок 5.8 – Создание куба

После создания и выбора таблиц и измерений Analysis Service автоматически посчитает значения в строках с числами:

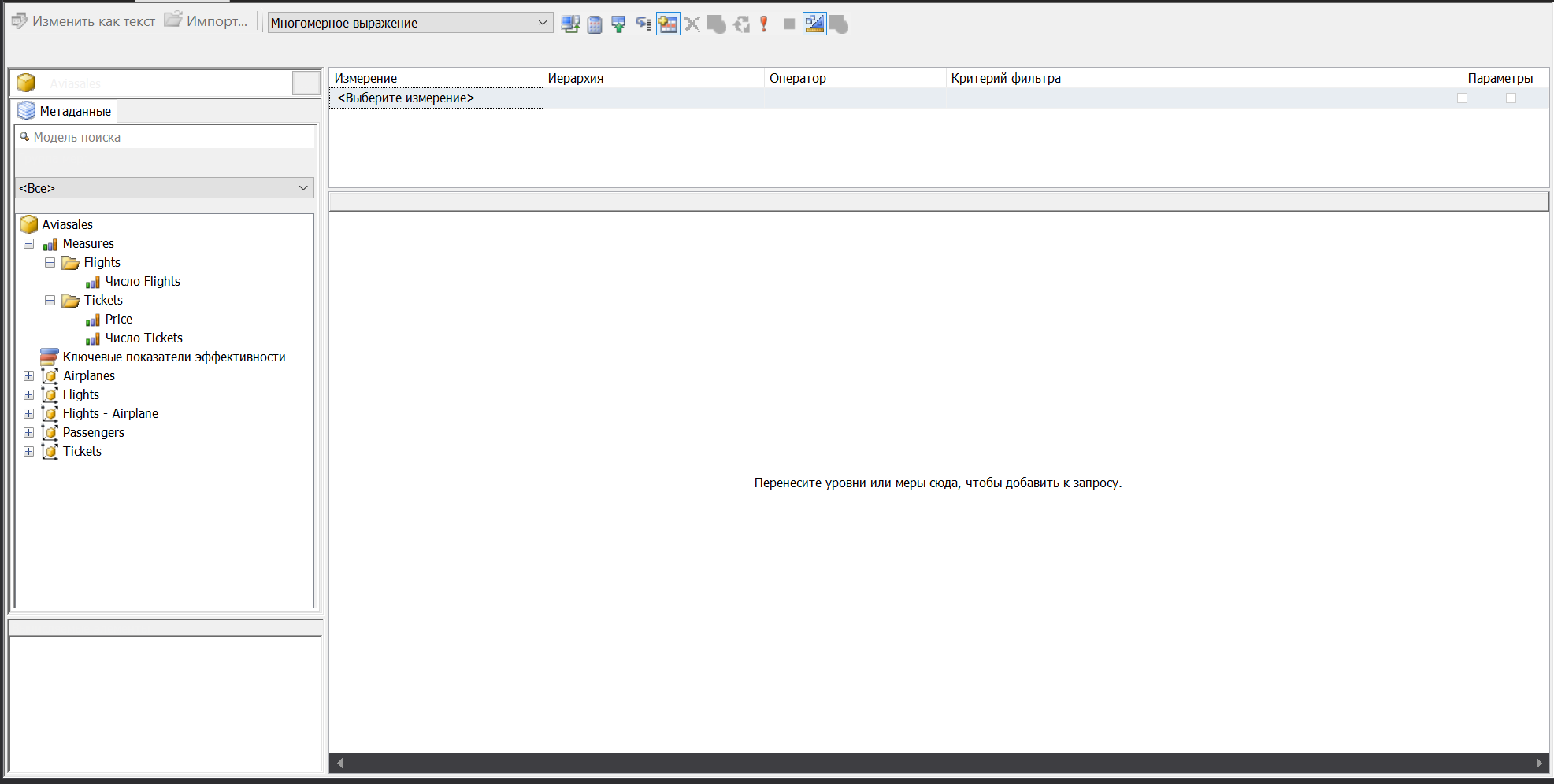


Рисунок 5.9 – Окно настройки куба

С помощью кубов можно анализировать информацию, создавая удобные условия сортировки. Например, разместив рядом столбцы AirplaneID и Price, куб покажет доход каждого самолёта:

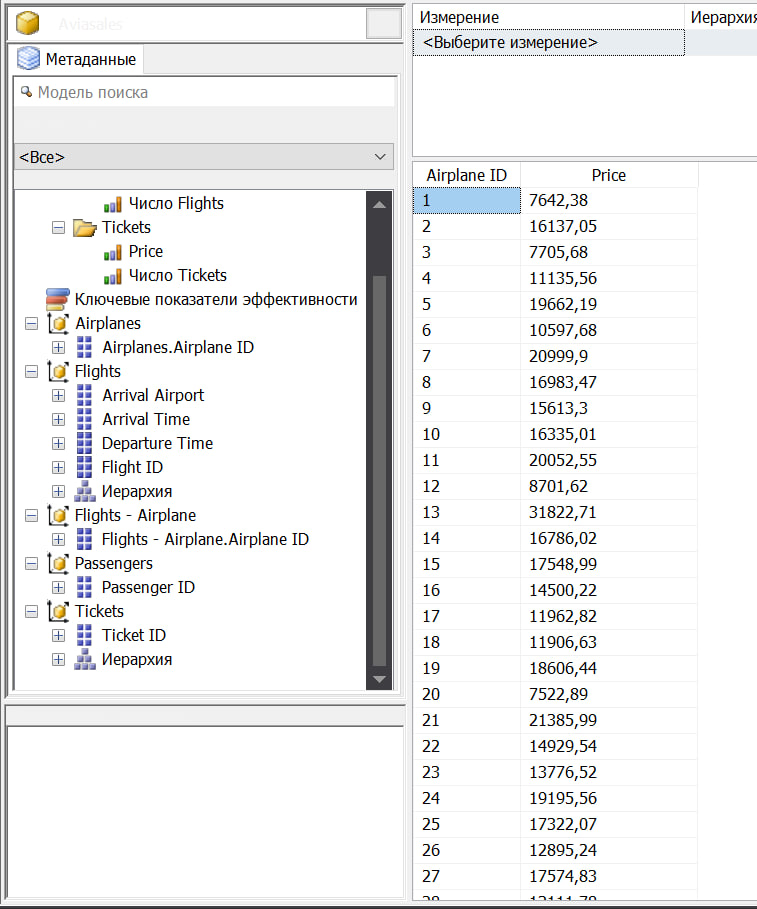


Рисунок 5.10 – Пример использования куба

Analysis Services предоставляет мощные возможности для анализа и извлечения данных из многомерных структур. Его интеграция с SQL Server обеспечивает бесперебойную работу с реляционными базами данных, а клиентские приложения могут использовать эту технологию для получения ценной бизнес–аналитики.

# 6. Описание дополнительных функцих.

### 6.1 Импорт и экспорт данных

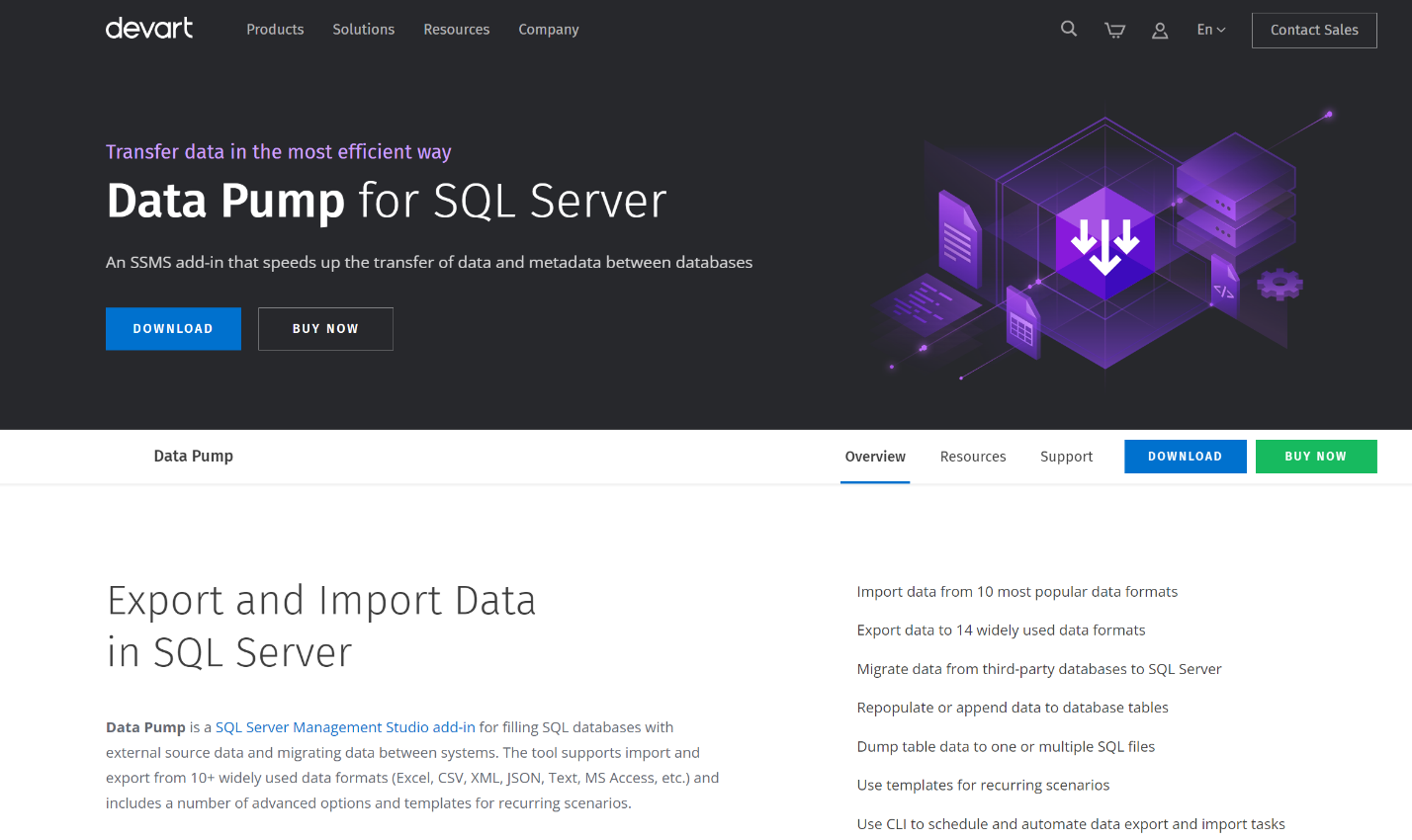
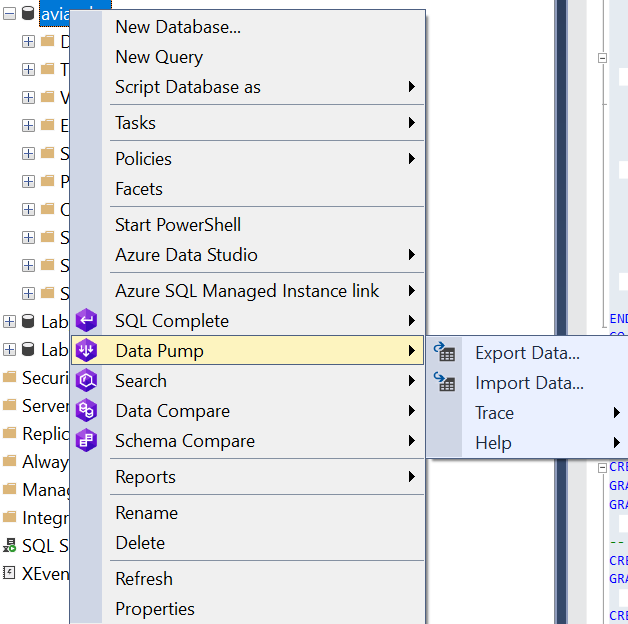
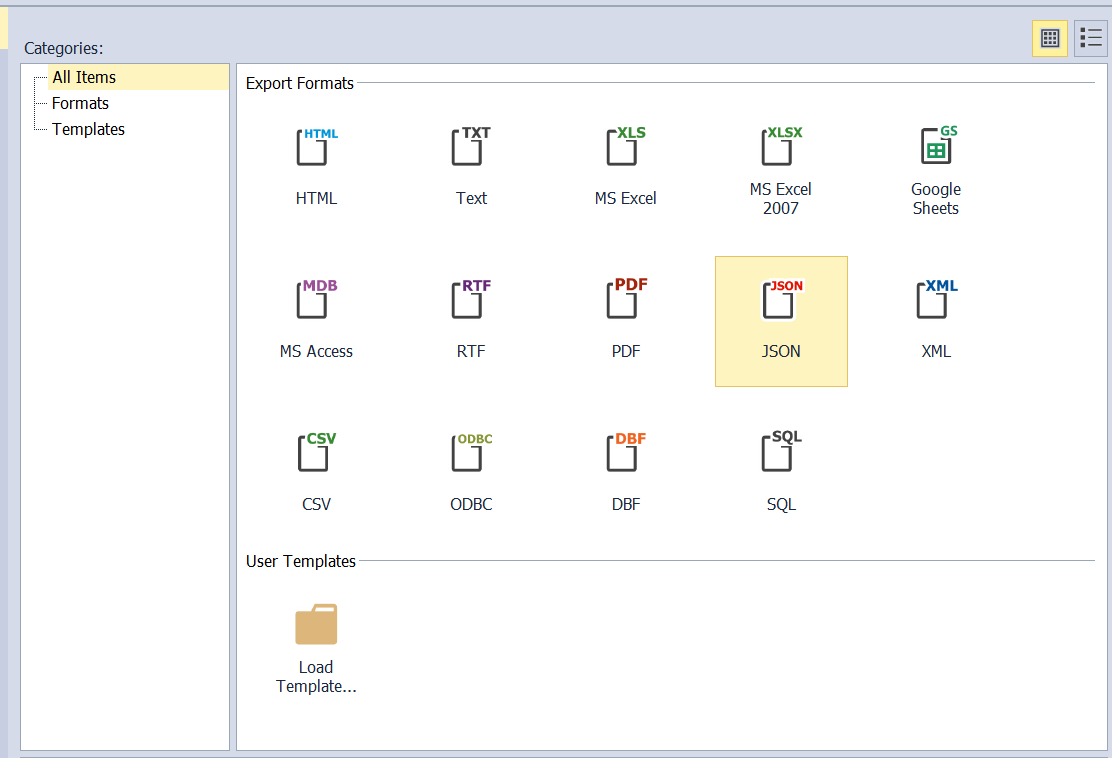
Для добавления возможности импорта и экспорта данных в SSMS было принято решение об установке дополнений Devart DBforge, в частности – DataPump.

Рисунок 6.1 – Главная страница DataPump

После установки в СУБД появится дополнительное контекстное окно, в котором можно будет произвести импорт/экспорт данных во всех основных форматах хранения данных.

Рисунок 6.2 – пример использования Data Pump

Рисунок 6.3 – пример использования Data Pump

После этого файл с данными будет создан по указанному пути.

### 6.2 Создание клиент–серверной архитектуры.

Для демонстрации работы клиента с базой данный был разработан сервер с возможностью покупки билетов. Для сервера был использован веб–фреймворк Express JS.

индекс представлен в листинге 3.4

|  |
| --- |
| const express = require('express');  const bodyParser = require('body-parser');  const path = require('path');  const sql = require('mssql');  const cors = require('cors');  const app = express();  app.use(cors());  // Middleware для обработки JSON и urlencoded данных  app.use(bodyParser.json());  app.use(bodyParser.urlencoded({ extended: true }));  // Настройки подключения к базе данных  const config = {  server: 'localhost',  database: 'aviasales',  user: 'sa',  password: '12345',  port: 1433,  options: {  encrypt: false  }  }; |

Листинг 6.1 – Создание сервера и подключение к БД

Со стороны клиента нужно реализовать запросы, для примера возьмем регистрацию:

|  |
| --- |
| // Функционал регистрации  registrationForm.addEventListener('submit', function(event) {  event.preventDefault();    const formData = new FormData(this);  const json = JSON.stringify(Object.fromEntries(formData));  fetch('http://localhost:3000/register', {  method: 'POST',  headers: {  'Content-Type': 'application/json'  },  body: json  })  .then(response => response.json())  .then(data => {  console.log(data);  if (data.success) {  alert('Регистрация прошла успешно!');  this.reset();  registrationFormContainer.style.display = 'none';  } else {  alert('Ошибка при регистрации: ' + data.error);  }  })  .catch(error => {  console.error('Ошибка:', error);  alert('Что-то пошло не так. Попробуйте еще раз.');  });  }); |

Листинг 6.2 – Регистрация клиента

Реализуем систему обработки запроса на стороне сервера:

|  |
| --- |
| // Обработчик для статических файлов (CSS и JavaScript)  app.use(express.static(path.join(\_\_dirname, 'public')));  // Метод для обработки запроса на регистрацию  app.post('/register', async (req, res) => {  const { firstName, lastName, passportNumber, birthDate } = req.body;  console.log('Получены данные регистрации:', req.body); // Выводим полученные данные в консоль  try {  // Подключаемся к базе данных  await sql.connect(config);    // Выполняем запрос к базе данных  const result = await sql.query`INSERT INTO Passengers (FirstName, LastName, PassportNumber, BirthDate) VALUES (${firstName}, ${lastName}, ${passportNumber}, ${birthDate})`;    // Отправляем ответ клиенту  res.json({ success: true });  } catch (error) {  // Если произошла ошибка, отправляем соответствующий ответ  console.error(error);  res.status(500).json({ success: false, error: error.message });  } finally {  // Закрываем соединение с базой данных  await sql.close();  }  }); |

Листинг 6.3 – Запрос на стороне сервера

При входе клиент будет перенаправлен в личный кабинет, где у него есть возможность поиска рейсов по данным, который в данный момент существуют в БД.

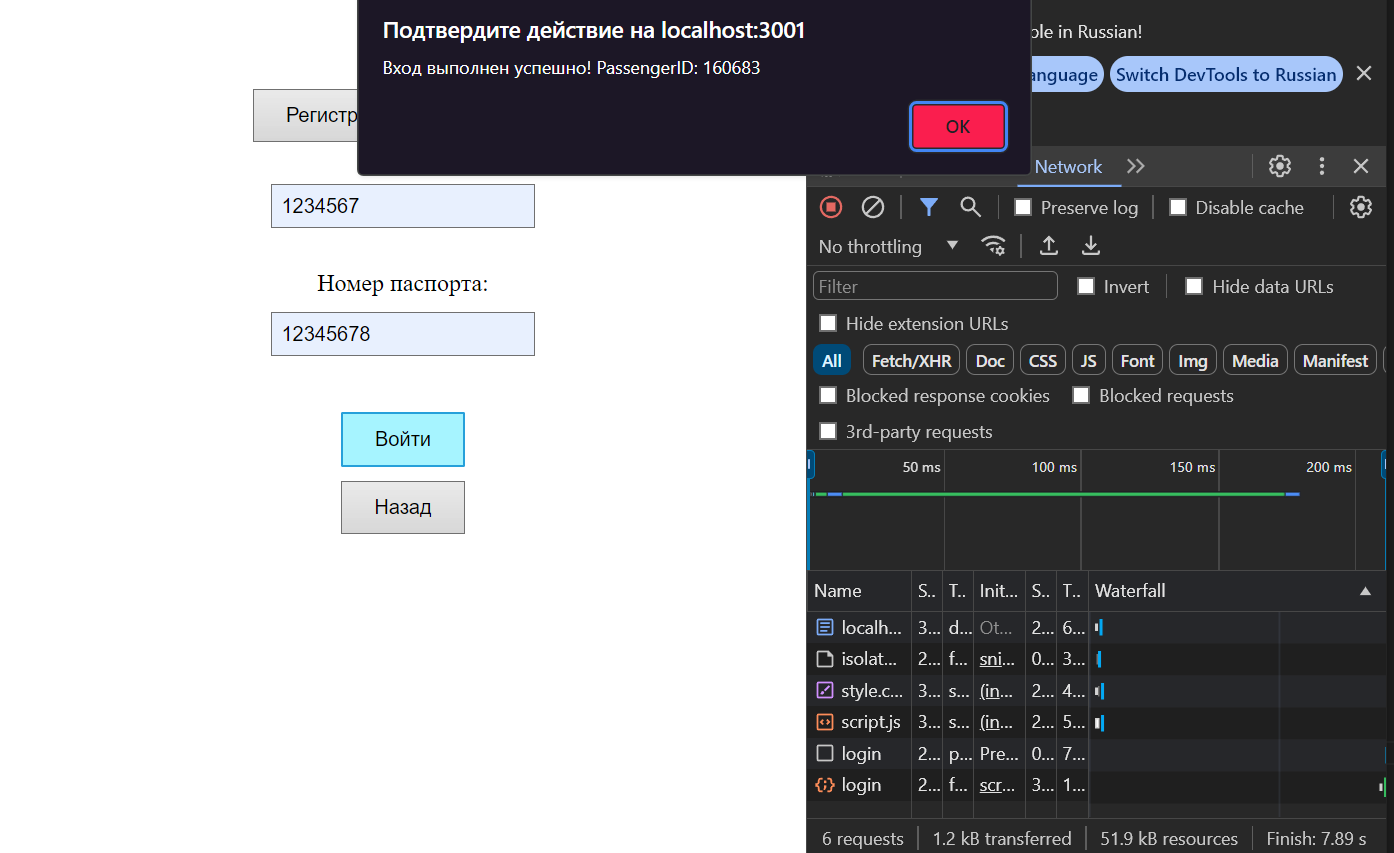


Рисунок 6.4 – Вход в личный кабинет

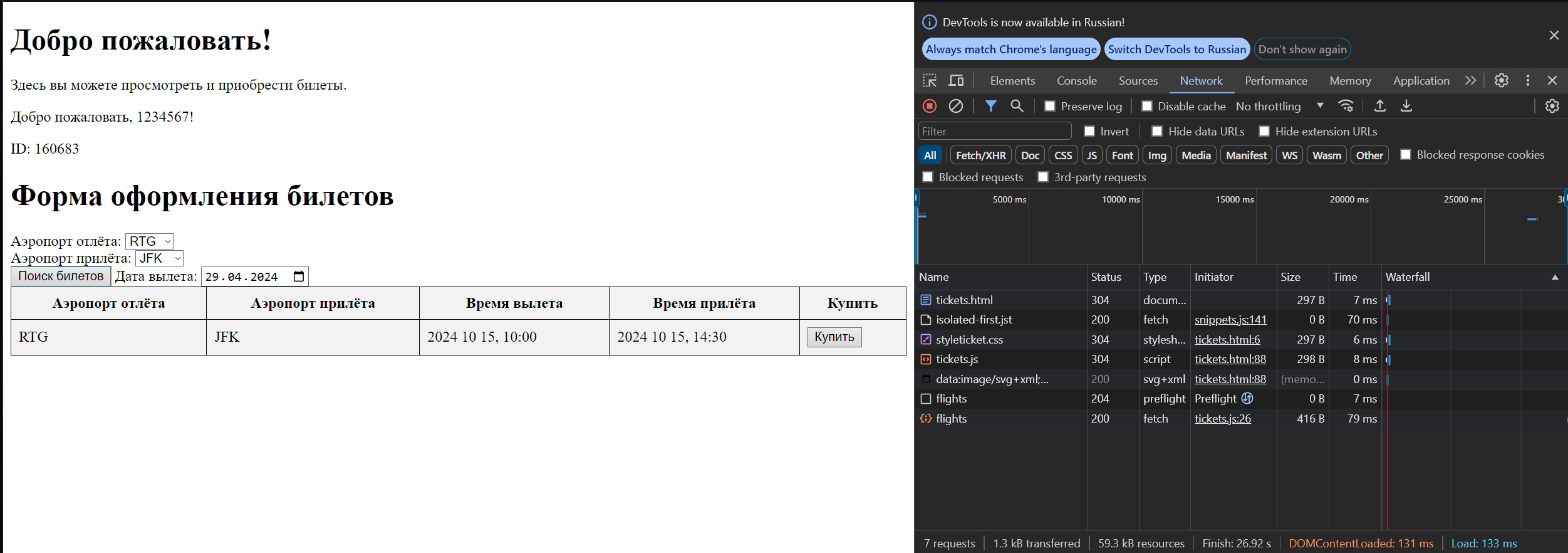


Рисунок 6.5 – Выбор билетов

После нажатия кнопки «Купить» в корзину помещается информация, собранная во время всей сессии пользователя.

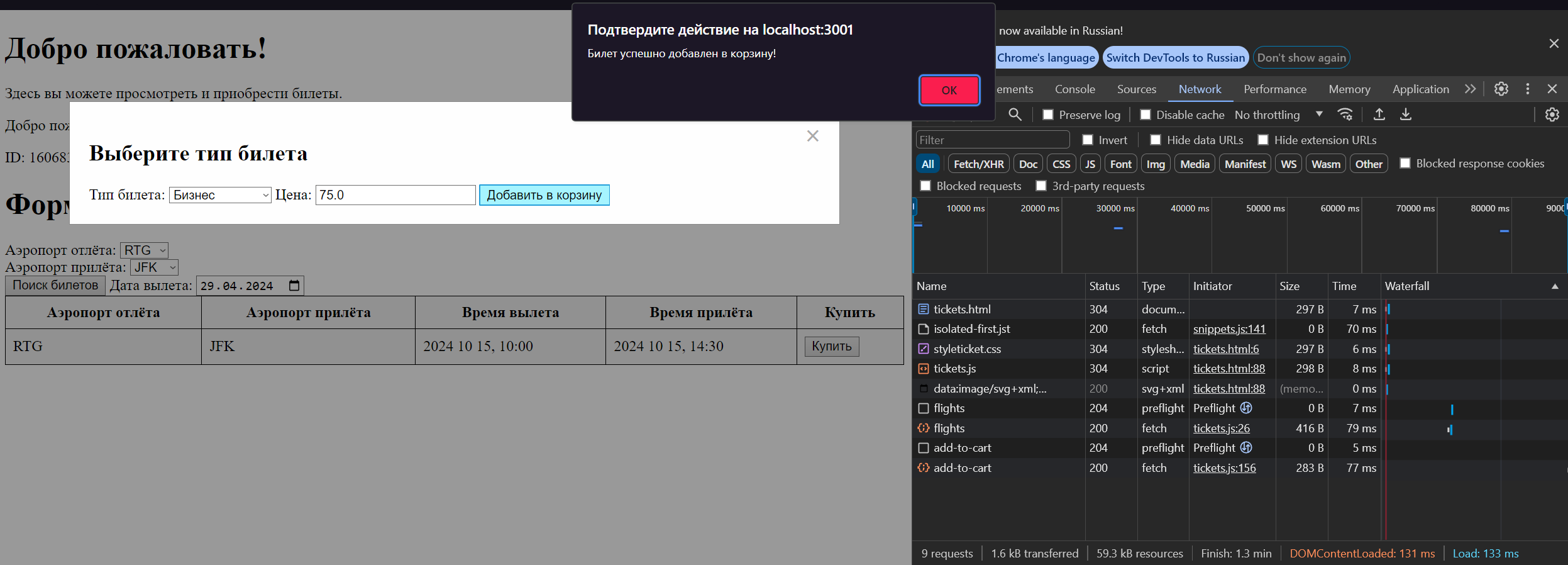


Рисунок 6.6 – Добавление билета в корзину

Была реализована корзина, в которой каждый пользователь видит свои билеты.

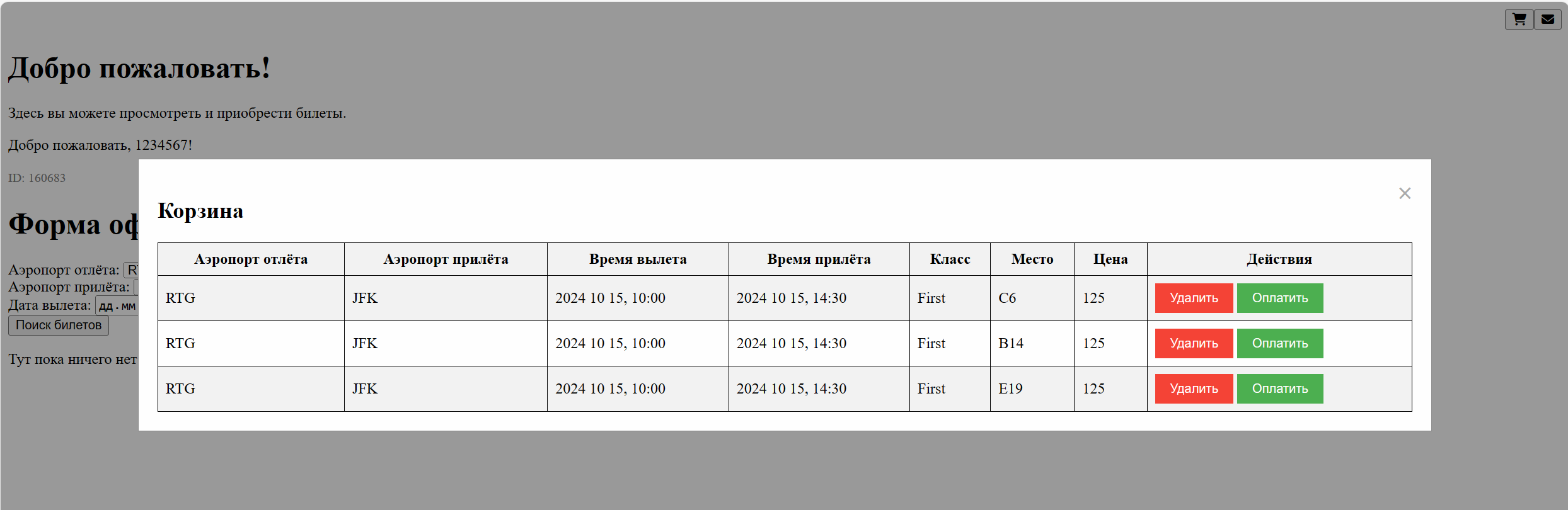


Рисунок 6.6 – Корзина

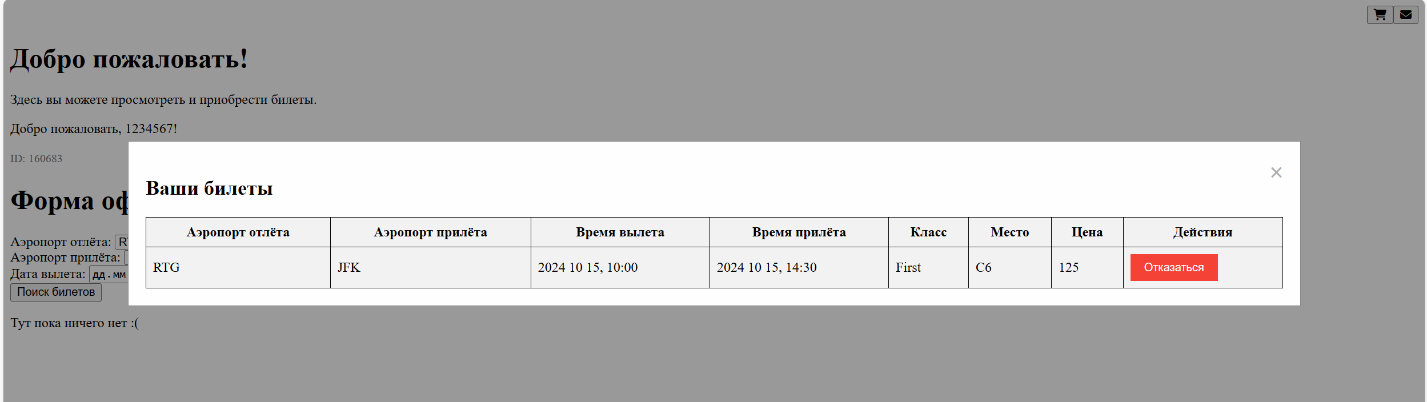
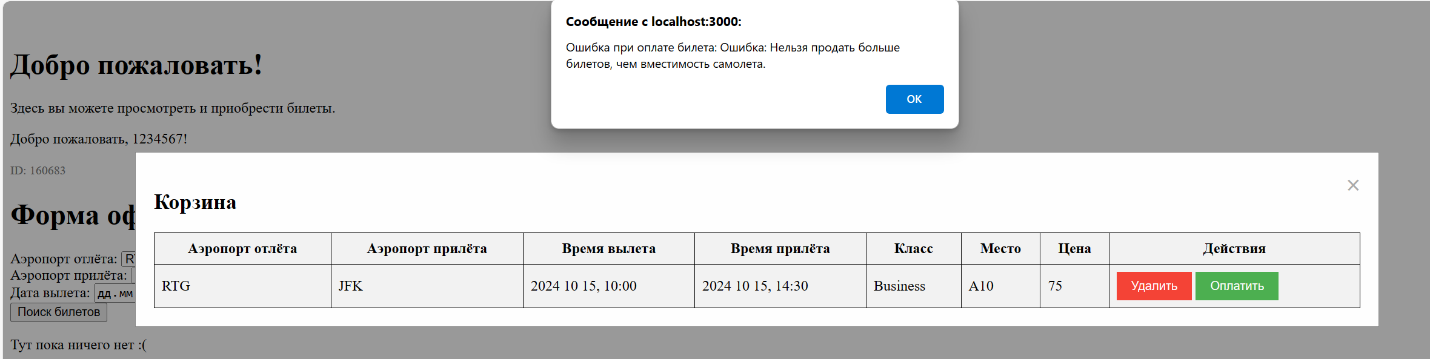
Были реализованы 2 возможности: удаление из корзины и оплата билета. После оплаты билет переходит в разряд купленных, от него также можно отказаться.

Рисунок 6.6 – Билеты

Рисунок 6.7 – Ошибка при покупке

Реализован триггер, который не позволяет самолёту переполняться: если будет куплено слишком много билетов, то это приведёт к ошибке

# Заключение

В ходе данного курсового проекта: была разработана модель и объекты базы данных для сервиса авиакомпании; были описаны и применены на практике процедуры экспорта и импорта данных; было проведено тестирование производительности базы данных, в ходе которого были разработаны индексы, ускорившие получение данных; разработаны различные процедуры для взаимодействия с базой данных.

Функционально были выполнены следующие задачи:

– Реализовать систему хранения объектов: рейсы, пути, самолёты.

– Реализовать функционал администратора: добавление, редактирование, удаление данных, оформление билетов.

– Добавить систему сортировки данных в БД.

– Добавить систему поиска данных в БД.

– Реализовать функционал пользователя: добавление в корзину,

оформление билета, возврат билета.

## Список использованных источников

1. Booking [Электронный ресурс] – https://www.booking.com/ – Дата доступа: 22.04.2024
2. Yandex Travel [Электронный ресурс] – Режим доступа: https://www.travel.yandex.ru.com / – Дата доступа: 22.04.2024.
3. Aviasales [Электронный ресурс] – https://aviasales.com/ – Дата доступа: 22.04.2024
4. Devart [Электронный ресурс] / Режим доступа: https://www.devart.com/dbforge/sql/studio/studio–sql.html – Дата доступа: 29.04.2024
5. Леоненков А. В. Самоучитель UML / Леоненков Александр Васильевич – 2–е изд. – СПб.: БХВ–Петербург, 2004. – 432 с.
6. Proven Strategies to Improve Database Performance [Электронный ресурс] – Режим доступа: https://www.metisdata.io/blog/8–proven–strategies–to–improve–database–performance – Дата доступа: 27.04.2024.

# Приложение А. Листинг таблиц

|  |
| --- |
| CREATE TABLE Airplanes (  AirplaneID INT PRIMARY KEY IDENTITY,  Model NVARCHAR(50) NOT NULL,  Capacity INT NOT NULL,  ProductionYear DATE NOT NULL  ); |

Листинг А.1 – Таблица самолётов

|  |
| --- |
| CREATE TABLE Flights (  FlightID INT PRIMARY KEY IDENTITY,  DepartureAirport NVARCHAR(3) NOT NULL,  ArrivalAirport NVARCHAR(3) NOT NULL,  DepartureTime DATETIME NOT NULL,  ArrivalTime DATETIME NOT NULL,  AirplaneID INT REFERENCES Airplanes(AirplaneID)  ); |

Листинг А.2 – Таблица рейсов

|  |
| --- |
| CREATE TABLE Passengers (  PassengerID INT PRIMARY KEY IDENTITY,  FirstName NVARCHAR(50) NOT NULL,  LastName NVARCHAR(50) NOT NULL,  PassportNumber NVARCHAR(20) NOT NULL UNIQUE,  BirthDate DATE NOT NULL  ); |

Листинг А.3 – Таблица пассажиров

|  |
| --- |
| CREATE TABLE Tickets (  TicketID INT PRIMARY KEY IDENTITY,  FlightID INT REFERENCES Flights(FlightID),  PassengerID INT REFERENCES Passengers(PassengerID),  SeatNumber NVARCHAR(4) NOT NULL,  TicketClass NVARCHAR(20) NOT NULL,  Price DECIMAL(10, 2) NOT NULL  ); |

Листинг А.4 – Таблица билетов

|  |
| --- |
| CREATE TABLE Employees (  EmployeeID INT PRIMARY KEY IDENTITY,  FirstName NVARCHAR(50) NOT NULL,  LastName NVARCHAR(50) NOT NULL,  Position NVARCHAR(50) NOT NULL,  HireDate DATE NOT NULL  ); |

Листинг А.5 – Таблица сотрудников

|  |
| --- |
| CREATE TABLE Crews (  CrewID INT PRIMARY KEY IDENTITY,  FlightID INT REFERENCES Flights(FlightID),  EmployeeID INT REFERENCES Employees(EmployeeID),  Role NVARCHAR(50) NOT NULL  ); |

Листинг А.6 – Таблица экипажа

|  |
| --- |
| CREATE TABLE ShoppingCart (  ItemID INT PRIMARY KEY IDENTITY,  FlightID INT REFERENCES Flights(FlightID),  PassengerID INT REFERENCES Passengers(PassengerID),  SeatNumber NVARCHAR(4) NOT NULL,  TicketClass NVARCHAR(20) NOT NULL,  Price DECIMAL(10, 2) NOT NULL  ); |

Листинг А.7 – Таблица бронирований

# Приложение В. Листинг представлений

|  |
| --- |
| –– Процедура добавления нового самолета  CREATE PROCEDURE AddAirplane  @Model NVARCHAR(50),  @Capacity INT,  @ProductionYear DATE  AS  BEGIN  INSERT INTO Airplanes (Model, Capacity, ProductionYear)  VALUES (@Model, @Capacity, @ProductionYear);  END;  GO  –– Процедура обновления информации о самолете  CREATE PROCEDURE UpdateAirplane  @AirplaneID INT,  @Model NVARCHAR(50),  @Capacity INT,  @ProductionYear DATE  AS  BEGIN  UPDATE Airplanes  SET Model = @Model, Capacity = @Capacity, ProductionYear = @ProductionYear  WHERE AirplaneID = @AirplaneID;  END;  GO  –– Процедура удаления самолета  CREATE PROCEDURE DeleteAirplane  @AirplaneID INT  AS  BEGIN  DELETE FROM Airplanes  WHERE AirplaneID = @AirplaneID;  END;  GO  –– Процедура получения информации о самолете по ID  CREATE PROCEDURE GetAirplaneByID  @AirplaneID INT  AS  BEGIN  SELECT \* FROM Airplanes  WHERE AirplaneID = @AirplaneID;  END;  GO  –– Процедура получения списка всех самолетов  CREATE PROCEDURE GetAllAirplanes  AS  BEGIN  SELECT \* FROM Airplanes;  END;  GO  –– Процедура получения списка самолетов по модели  CREATE PROCEDURE GetAirplanesByModel  @Model NVARCHAR(50)  AS  BEGIN  SELECT \* FROM Airplanes  WHERE Model = @Model;  END;  GO  –– Процедура добавления случайного самолета  CREATE PROCEDURE AddRandomAirplane  AS  BEGIN  DECLARE @RandomModel NVARCHAR(50);  DECLARE @RandomCapacity INT;  DECLARE @RandomYear DATE;  –– Список возможных моделей самолетов  DECLARE @Models TABLE (Model NVARCHAR(50));  INSERT INTO @Models VALUES  ('Airbus A318'), ('Airbus A319'), ('Airbus A320'), ('Airbus A321'), ('Airbus A330'),  ('Airbus A340'), ('Airbus A350'), ('Airbus A380'), ('Boeing 737'), ('Boeing 747'),  ('Boeing 757'), ('Boeing 767'), ('Boeing 777'), ('Boeing 787'), ('Embraer E170'),  ('Embraer E175'), ('Embraer E190'), ('Embraer E195'), ('Bombardier CRJ700'), ('Bombardier CRJ900'),  ('Bombardier CRJ1000'), ('ATR 42'), ('ATR 72'), ('Sukhoi Superjet 100'), ('Mitsubishi Regional Jet');  SELECT TOP 1 @RandomModel = Model FROM @Models ORDER BY NEWID();  –– Генерация случайной вместимости (150–400)  SELECT @RandomCapacity = ROUND((350 – 150) \* RAND() + 150, 0);  –– Генерация случайного года производства (2000–2023)  SELECT @RandomYear = DATEADD(YEAR, ROUND((23 – 0) \* RAND() + 0, 0), '20000101');  –– Добавление сгенерированного самолета  INSERT INTO Airplanes (Model, Capacity, ProductionYear)  VALUES (@RandomModel, @RandomCapacity, @RandomYear);  END;  GO  –– Процедура удаления всех самолетов  CREATE PROCEDURE DeleteAllAirplanes  AS  BEGIN  DELETE FROM Airplanes;  END;  GO |

Листинг В.1 – Процедуры для самолётов

|  |
| --- |
| –– Процедура удаления всех рейсов  CREATE PROCEDURE DeleteAllFlights  AS  BEGIN  DELETE FROM Flights;  END;  GO  –– Процедура добавления нового рейса  CREATE PROCEDURE AddFlight  @DepartureAirport NVARCHAR(3),  @ArrivalAirport NVARCHAR(3),  @DepartureTime DATETIME,  @ArrivalTime DATETIME,  @AirplaneID INT  AS  BEGIN  INSERT INTO Flights (DepartureAirport, ArrivalAirport, DepartureTime, ArrivalTime, AirplaneID)  VALUES (@DepartureAirport, @ArrivalAirport, @DepartureTime, @ArrivalTime, @AirplaneID);  END;  GO  –– Процедура обновления информации о рейсе  CREATE PROCEDURE UpdateFlight  @FlightID INT,  @DepartureAirport NVARCHAR(3),  @ArrivalAirport NVARCHAR(3),  @DepartureTime DATETIME,  @ArrivalTime DATETIME,  @AirplaneID INT  AS  BEGIN  UPDATE Flights  SET DepartureAirport = @DepartureAirport, ArrivalAirport = @ArrivalAirport,  DepartureTime = @DepartureTime, ArrivalTime = @ArrivalTime, AirplaneID = @AirplaneID  WHERE FlightID = @FlightID;  END;  GO  –– Процедура удаления рейса  CREATE PROCEDURE DeleteFlight  @FlightID INT  AS  BEGIN  DELETE FROM Flights  WHERE FlightID = @FlightID;  END;  GO  –– Процедура получения информации о рейсе по ID  CREATE PROCEDURE GetFlightByID  @FlightID INT  AS  BEGIN  SELECT \* FROM Flights  WHERE FlightID = @FlightID;  END;  GO  –– Процедура получения списка всех рейсов  CREATE PROCEDURE GetAllFlights  AS  BEGIN  SELECT \* FROM Flights;  END;  GO  –– Процедура получения списка рейсов по аэропорту  CREATE PROCEDURE GetFlightsByAirport  @Airport NVARCHAR(3)  AS  BEGIN  SELECT \* FROM Flights  WHERE DepartureAirport = @Airport OR ArrivalAirport = @Airport;  END;  GO  –– Процедура получения списка рейсов на определенную дату  CREATE PROCEDURE GetFlightsByDate  @Date DATE  AS  BEGIN  SELECT \* FROM Flights  WHERE CAST(DepartureTime AS DATE) = @Date;  END;  GO  –– Процедура добавления случайного рейса  CREATE PROCEDURE AddRandomFlight  AS  BEGIN  DECLARE @RandomDepartureAirport NVARCHAR(3);  DECLARE @RandomArrivalAirport NVARCHAR(3);  DECLARE @RandomDepartureTime DATETIME;  DECLARE @RandomArrivalTime DATETIME;  DECLARE @RandomAirplaneID INT;  –– Список возможных аэропортов  DECLARE @Airports TABLE (AirportCode NVARCHAR(3));  INSERT INTO @Airports VALUES ('RTG'), ('JFK'), ('LAX'), ('LHR'), ('CDG'), ('HND');  –– Выбор случайных аэропортов отправления и прибытия  SELECT TOP 1 @RandomDepartureAirport = AirportCode FROM @Airports ORDER BY NEWID();  SELECT TOP 1 @RandomArrivalAirport = AirportCode FROM @Airports WHERE AirportCode != @RandomDepartureAirport ORDER BY NEWID();  –– Генерация случайной даты вылета в диапазоне ближайших 30 дней  SELECT @RandomDepartureTime = DATEADD(DAY, ROUND((29 – 0) \* RAND() + 0, 0), GETDATE());  SET @RandomDepartureTime = DATEADD(HOUR, ROUND((23 – 0) \* RAND() + 0, 0), @RandomDepartureTime); –– Случайный час  –– Генерация случайной даты прилета (на 1–5 часов позже вылета)  SELECT @RandomArrivalTime = DATEADD(HOUR, ROUND((5 – 1) \* RAND() + 1, 0), @RandomDepartureTime);  –– Выбор случайного самолета  SELECT TOP 1 @RandomAirplaneID = AirplaneID FROM Airplanes ORDER BY NEWID();  –– Добавление сгенерированного рейса  INSERT INTO Flights (DepartureAirport, ArrivalAirport, DepartureTime, ArrivalTime, AirplaneID)  VALUES (@RandomDepartureAirport, @RandomArrivalAirport, @RandomDepartureTime, @RandomArrivalTime, @RandomAirplaneID);  END;  GO |

Листинг В.2 – Процедуры для рейсов

|  |
| --- |
| –– Процедура добавления нового пассажира  CREATE PROCEDURE AddPassenger  @FirstName NVARCHAR(50),  @LastName NVARCHAR(50),  @PassportNumber NVARCHAR(20),  @BirthDate DATE  AS  BEGIN  INSERT INTO Passengers (FirstName, LastName, PassportNumber, BirthDate)  VALUES (@FirstName, @LastName, @PassportNumber, @BirthDate);  END;  GO  –– Процедура обновления информации о пассажире  CREATE PROCEDURE UpdatePassenger  @PassengerID INT,  @FirstName NVARCHAR(50),  @LastName NVARCHAR(50),  @PassportNumber NVARCHAR(20),  @BirthDate DATE  AS  BEGIN  UPDATE Passengers  SET FirstName = @FirstName, LastName = @LastName,  PassportNumber = @PassportNumber, BirthDate = @BirthDate  WHERE PassengerID = @PassengerID;  END;  GO  –– Процедура удаления пассажира  CREATE PROCEDURE DeletePassenger  @PassengerID INT  AS  BEGIN  DELETE FROM Passengers  WHERE PassengerID = @PassengerID;  END;  GO  –– Процедура получения информации о пассажире по ID  CREATE PROCEDURE GetPassengerByID  @PassengerID INT  AS  BEGIN  SELECT \* FROM Passengers  WHERE PassengerID = @PassengerID;  END;  GO  –– Процедура получения списка всех пассажиров  CREATE PROCEDURE GetAllPassengers  AS  BEGIN  SELECT \* FROM Passengers;  END;  GO  –– Процедура получения информации о пассажире по номеру паспорта  CREATE PROCEDURE GetPassengerByPassport  @PassportNumber NVARCHAR(20)  AS  BEGIN  SELECT \* FROM Passengers  WHERE PassportNumber = @PassportNumber;  END;  GO  –– Процедура добавления случайного пассажира  CREATE PROCEDURE AddRandomPassenger  AS  BEGIN  DECLARE @RandomFirstName NVARCHAR(50);  DECLARE @RandomLastName NVARCHAR(50);  DECLARE @RandomPassportNumber NVARCHAR(20);  DECLARE @RandomBirthDate DATE;  DECLARE @FirstNames TABLE (FirstName NVARCHAR(50));  INSERT INTO @FirstNames VALUES  ('John'), ('Mary'), ('Alex'), ('Helen'), ('Dmitry'), ('Anna'), ('Sergey'), ('Olga'),  ('Andrew'), ('Tatiana'), ('Vladimir'), ('Natalie'), ('Michael'), ('Catherine'), ('Alexander'), ('Julia'),  ('Maxim'), ('Svetlana'), ('Nikita'), ('Irina');  DECLARE @LastNames TABLE (LastName NVARCHAR(50));  INSERT INTO @LastNames VALUES  ('Johnson'), ('Smith'), ('Brown'), ('Taylor'), ('Williams'), ('Jones'), ('Davis'), ('Miller'),  ('Wilson'), ('Moore'), ('Taylor'), ('Anderson'), ('Thomas'), ('Jackson'), ('White'), ('Harris'),  ('Martin'), ('Thompson');  –– Выбор случайного имени и фамилии  SELECT TOP 1 @RandomFirstName = FirstName FROM @FirstNames ORDER BY NEWID();  SELECT TOP 1 @RandomLastName = LastName FROM @LastNames ORDER BY NEWID();  –– Генерация случайного номера паспорта (10 цифр)  SELECT @RandomPassportNumber = CAST(RAND() \* 100000000 AS bigint);  –– Генерация случайной даты рождения (1950–2005)  SELECT @RandomBirthDate = DATEADD(YEAR, ROUND((55 – 0) \* RAND() + 0, 0), '19500101');  –– Добавление сгенерированного пассажира  INSERT INTO Passengers (FirstName, LastName, PassportNumber, BirthDate)  VALUES (@RandomFirstName, @RandomLastName, @RandomPassportNumber, @RandomBirthDate);  END;  GO  –– Процедура удаления всех рейсов  CREATE PROCEDURE DeleteAllPassengers  AS  BEGIN  DELETE FROM Passengers;  END;  GO |

Листинг В.3 – Процедуры для пассажиров

|  |
| --- |
| CREATE PROCEDURE AddToShoppingCart  @FlightID INT,  @PassengerID INT,  @SeatNumber NVARCHAR(4),  @TicketClass NVARCHAR(20),  @Price DECIMAL(10, 2)  AS  BEGIN  INSERT INTO ShoppingCart (FlightID, PassengerID, SeatNumber, TicketClass, Price)  VALUES (@FlightID, @PassengerID, @SeatNumber, @TicketClass, @Price);  END;  CREATE PROCEDURE ProcessTicket  @TicketID INT  AS  BEGIN  DECLARE @FlightID INT;  DECLARE @PassengerID INT;  DECLARE @SeatNumber NVARCHAR(4);  DECLARE @TicketClass NVARCHAR(20);  DECLARE @Price DECIMAL(10, 2);  –– Получение данных билета из корзины по ID  SELECT  @FlightID = FlightID,  @PassengerID = PassengerID,  @SeatNumber = SeatNumber,  @TicketClass = TicketClass,  @Price = Price  FROM ShoppingCart  WHERE ItemID = @TicketID;  –– Вставка данных билета в таблицу Tickets  INSERT INTO Tickets (FlightID, PassengerID, SeatNumber, TicketClass, Price)  VALUES (@FlightID, @PassengerID, @SeatNumber, @TicketClass, @Price);  –– Удаление данных билета из корзины  DELETE FROM ShoppingCart WHERE ItemID = @TicketID;  END;  CREATE PROCEDURE GetAllCartData  AS  BEGIN  SELECT \* FROM ShoppingCart;  END;  –– Процедура добавления нового билета  CREATE PROCEDURE AddTicket  @FlightID INT,  @PassengerID INT,  @SeatNumber NVARCHAR(4),  @TicketClass NVARCHAR(20),  @Price DECIMAL(10, 2)  AS  BEGIN  INSERT INTO Tickets (FlightID, PassengerID, SeatNumber, TicketClass, Price)  VALUES (@FlightID, @PassengerID, @SeatNumber, @TicketClass, @Price);  END;  GO  –– Процедура обновления информации о билете  CREATE PROCEDURE UpdateTicket  @TicketID INT,  @FlightID INT,  @PassengerID INT,  @SeatNumber NVARCHAR(4),  @TicketClass NVARCHAR(20),  @Price DECIMAL(10, 2)  AS  BEGIN  UPDATE Tickets  SET FlightID = @FlightID, PassengerID = @PassengerID,  SeatNumber = @SeatNumber, TicketClass = @TicketClass, Price = @Price  WHERE TicketID = @TicketID;  END;  GO  –– Процедура удаления билета  CREATE PROCEDURE DeleteTicket  @TicketID INT  AS  BEGIN  DELETE FROM Tickets  WHERE TicketID = @TicketID;  END;  GO  EXEC AddRandomFlight;  SELECT TOP 1 \*  FROM Flights  ORDER BY FlightID DESC;  –– Процедура получения информации о билете по ID  CREATE PROCEDURE GetTicketByID  @TicketID INT  AS  BEGIN  SELECT \* FROM Tickets  WHERE TicketID = @TicketID;  END;  GO  –– Процедура получения списка всех билетов  CREATE PROCEDURE GetAllTickets  AS  BEGIN  SELECT \* FROM Tickets;  END;  GO  –– Процедура получения списка билетов на определенный рейс  CREATE PROCEDURE GetTicketsByFlight  @FlightID INT  AS  BEGIN  SELECT \* FROM Tickets  WHERE FlightID = @FlightID;  END;  GO  –– Процедура получения списка билетов, купленных определенным пассажиром  CREATE PROCEDURE GetTicketsByPassenger  @PassengerID INT  AS  BEGIN  SELECT \* FROM Tickets  WHERE PassengerID = @PassengerID;  END;  GO  –– Процедура добавления случайного билета  CREATE PROCEDURE AddRandomTicket  AS  BEGIN  DECLARE @RandomFlightID INT;  DECLARE @RandomPassengerID INT;  DECLARE @RandomSeatNumber NVARCHAR(4);  DECLARE @RandomTicketClass NVARCHAR(20);  DECLARE @RandomPrice DECIMAL(10, 2);  –– Списки возможных классов билетов  DECLARE @TicketClasses TABLE (TicketClass NVARCHAR(20));  INSERT INTO @TicketClasses VALUES ('Economy'), ('Business'), ('First');  –– Выбор случайного рейса и пассажира  SELECT TOP 1 @RandomFlightID = FlightID FROM Flights ORDER BY NEWID();  SELECT TOP 1 @RandomPassengerID = PassengerID FROM Passengers ORDER BY NEWID();  –– Генерация случайного номера места (ряд 1–30, место A–F)  SELECT @RandomSeatNumber = CAST(RAND() \* 30 + 1 AS NVARCHAR(20)) + CHAR(CAST(RAND() \* 6 + 65 AS INT));  –– Выбор случайного класса билета  SELECT TOP 1 @RandomTicketClass = TicketClass FROM @TicketClasses ORDER BY NEWID();  –– Генерация случайной цены (100–1000)  SELECT @RandomPrice = ROUND((1000 – 100) \* RAND() + 100, 2);  –– Генерация случайного номера места (ряд 1–30, место A–F)  DECLARE @RandomRow INT;  DECLARE @RandomSeat CHAR(1);  SELECT @RandomRow = CAST(RAND() \* 30 + 1 AS INT);  SELECT @RandomSeat = CHAR(CAST(RAND() \* 6 + 65 AS INT));  SET @RandomSeatNumber = CAST(@RandomRow AS NVARCHAR(20)) + @RandomSeat;  –– Добавление сгенерированного билета  INSERT INTO Tickets (FlightID, PassengerID, SeatNumber, TicketClass, Price)  VALUES (@RandomFlightID, @RandomPassengerID, @RandomSeatNumber, @RandomTicketClass, @RandomPrice);  END;  GO |

Листинг В.4 – Процедуры для билетов

|  |
| --- |
| –– Процедура добавления нового сотрудника  CREATE PROCEDURE AddEmployee  @FirstName NVARCHAR(50),  @LastName NVARCHAR(50),  @Position NVARCHAR(50),  @HireDate DATE  AS  BEGIN  INSERT INTO Employees (FirstName, LastName, Position, HireDate)  VALUES (@FirstName, @LastName, @Position, @HireDate);  END;  GO  –– Процедура обновления информации о сотруднике  CREATE PROCEDURE UpdateEmployee  @EmployeeID INT,  @FirstName NVARCHAR(50),  @LastName NVARCHAR(50),  @Position NVARCHAR(50),  @HireDate DATE  AS  BEGIN  UPDATE Employees  SET FirstName = @FirstName, LastName = @LastName,  Position = @Position, HireDate = @HireDate  WHERE EmployeeID = @EmployeeID;  END;  GO  –– Процедура удаления сотрудника  CREATE PROCEDURE DeleteEmployee  @EmployeeID INT  AS  BEGIN  DELETE FROM Employees  WHERE EmployeeID = @EmployeeID;  END;  GO  –– Процедура получения информации о сотруднике по ID  CREATE PROCEDURE GetEmployeeByID  @EmployeeID INT  AS  BEGIN  SELECT \* FROM Employees  WHERE EmployeeID = @EmployeeID;  END;  GO  –– Процедура получения списка всех сотрудников  CREATE PROCEDURE GetAllEmployees  AS  BEGIN  SELECT \* FROM Employees;  END;  GO  –– Процедура получения списка сотрудников по должности  CREATE PROCEDURE GetEmployeesByPosition  @Position NVARCHAR(50)  AS  BEGIN  SELECT \* FROM Employees  WHERE Position = @Position;  END;  GO |

Листинг В.5 – Процедуры для сотрудников

|  |
| --- |
| –– Процедура добавления члена экипажа к рейсу  CREATE PROCEDURE AddCrewMember  @FlightID INT,  @EmployeeID INT,  @Role NVARCHAR(50)  AS  BEGIN  INSERT INTO Crews (FlightID, EmployeeID, Role)  VALUES (@FlightID, @EmployeeID, @Role);  END;  GO  –– Процедура обновления роли члена экипажа  CREATE PROCEDURE UpdateCrewMember  @CrewID INT,  @Role NVARCHAR(50)  AS  BEGIN  UPDATE Crews  SET Role = @Role  WHERE CrewID = @CrewID;  END;  GO  –– Процедура удаления члена экипажа из рейса  CREATE PROCEDURE DeleteCrewMember  @CrewID INT  AS  BEGIN  DELETE FROM Crews  WHERE CrewID = @CrewID;  END;  GO  –– Процедура получения списка экипажа для рейса  CREATE PROCEDURE GetCrewByFlight  @FlightID INT  AS  BEGIN  SELECT c.CrewID, e.FirstName, e.LastName, c.Role  FROM Crews c  JOIN Employees e ON c.EmployeeID = e.EmployeeID  WHERE c.FlightID = @FlightID;  END;  GO  CREATE PROCEDURE AddRandomCrew  @FlightID INT  AS  BEGIN  –– Temporary table to store available employees (excluding pilots and copilots later)  DECLARE @AvailableCrew TABLE (EmployeeID INT, Position NVARCHAR(50));  –– Select employees with appropriate positions  INSERT INTO @AvailableCrew  SELECT EmployeeID, Position FROM Employees  WHERE Position IN ('Pilot', 'Copilot', 'Flight attendant');  –– Assign pilot (ensure uniqueness)  DECLARE @SelectedPilotID INT;  SELECT TOP 1 @SelectedPilotID = EmployeeID FROM @AvailableCrew WHERE Position = 'Pilot' ORDER BY NEWID();  EXEC AddCrewMember @FlightID, @SelectedPilotID, 'Pilot';  –– Assign copilot (exclude already selected pilot)  DECLARE @SelectedCopilotID INT;  SELECT TOP 1 @SelectedCopilotID = EmployeeID FROM @AvailableCrew WHERE Position = 'Copilot' AND EmployeeID <> @SelectedPilotID ORDER BY NEWID();  EXEC AddCrewMember @FlightID, @SelectedCopilotID, 'Copilot';  –– Create a temporary table excluding the selected pilot and copilot  DECLARE @AvailableAttendants TABLE (EmployeeID INT);  INSERT INTO @AvailableAttendants  SELECT EmployeeID FROM @AvailableCrew  WHERE EmployeeID NOT IN (@SelectedPilotID, @SelectedCopilotID) AND Position = 'Flight attendant';  –– Assign flight attendants (ensure uniqueness and exclude pilot/copilot)  DECLARE @NumAttendants INT = CAST(RAND() \* 3 + 3 AS INT);  WHILE @NumAttendants > 0  BEGIN  DECLARE @AttendantID INT;  SELECT TOP 1 @AttendantID = EmployeeID FROM @AvailableAttendants ORDER BY NEWID();  EXEC AddCrewMember @FlightID, @AttendantID, 'Flight attendant';    –– Remove assigned attendant from the pool  DELETE FROM @AvailableAttendants WHERE EmployeeID = @AttendantID;  SET @NumAttendants = @NumAttendants – 1;  END;  END;  GO  CREATE PROCEDURE GetAllCrew  @FlightID INT  AS  BEGIN  SELECT \* FROM CREWS WHERE FlightID = @FlightID;  END;  GO |

Листинг В.6 – Процедуры для экипажа