



Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Московский государственный технический университет  
имени Н. Э. Баумана  
(национальный исследовательский университет)»  
(МГТУ им. Н. Э. Баумана)

---

ФАКУЛЬТЕТ «Информатика и системы управления»

---

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

---

# РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

## *К КУРСОВОЙ РАБОТЕ*

### *НА ТЕМУ:*

*«Разработка базы данных для хранения и обработки  
данных авиакомпании»*

Студент ИУ7-63Б  
(Группа)

\_\_\_\_\_  
(Подпись, дата)

ИУ7-63Б  
(И. О. Фамилия)

Руководитель курсовой работы

\_\_\_\_\_  
(Подпись, дата)

В. Марченко  
(И. О. Фамилия)

Консультант

\_\_\_\_\_  
(Подпись, дата)

Д. А. Шибанова  
(И. О. Фамилия)

*2023 г.*

## РЕФЕРАТ

Расчетно-пояснительная записка 48 с., 9 рис., 7 табл., 11 источн., 1 прил.  
АВИАБИЛЕТЫ, АВИАПЕРЕЛЕТЫ, БАЗЫ ДАННЫХ, ВЕБ-ПРИЛОЖЕНИЕ, РЕЛЯЦИОННАЯ МОДЕЛЬ ДАННЫХ, SQL

Объектом разработки является база данных и приложение к ней.

Цель работы: разработка базы данных для хранения и обработки данных авиакомпании и веб-приложения, которое будет ее использовать.

В результате выполнения работы была разработана база данных для хранения и обработки данных авиакомпании и веб-приложение, использующее эту базу данных.

Область применения результатов — дальнейшее развитие и расширение приложения для поиска и покупки авиабилетов.

# СОДЕРЖАНИЕ

<b>ВВЕДЕНИЕ</b>	<b>7</b>
<b>1 Аналитическая часть</b>	<b>9</b>
1.1 Анализ предметной области . . . . .	9
1.2 Требования к базе данных и приложению . . . . .	9
1.3 Модели баз данных . . . . .	10
1.3.1 Дореляционные базы данных . . . . .	10
1.3.2 Постреляционные базы данных . . . . .	13
1.3.3 Реляционные базы данных . . . . .	15
1.4 Информация, подлежащая хранению в базе данных . . . . .	16
1.5 ER-диаграмма сущностей базы данных . . . . .	16
1.6 Пользователи приложения . . . . .	17
1.7 Диаграмма вариантов использования . . . . .	19
<b>2 Конструкторская часть</b>	<b>22</b>
2.1 Описание сущностей базы данных . . . . .	22
2.2 Описание ограничений целостности базы данных . . . . .	25
2.3 Описание проектируемой функции на уровне базы данных . . . . .	26
2.4 Описание ролевой модели на уровне базы данных . . . . .	27
<b>3 Технологическая часть</b>	<b>28</b>
3.1 Средства реализации . . . . .	28
3.2 Реализация сущностей базы данных . . . . .	29
3.3 Реализация ограничений целостности базы данных . . . . .	31
3.4 Реализация функции на уровне базы данных . . . . .	35
3.5 Реализация ролевой модели на уровне базы данных . . . . .	36
3.6 Реализация интерфейса доступа к базе данных . . . . .	38
3.7 Тестирование разработанного функционала . . . . .	39
<b>4 Исследовательская часть</b>	<b>41</b>
4.1 Технические характеристики устройства . . . . .	41
4.2 Исследование характеристик разработанного программного обеспечения . . . . .	41

4.3	Время выполнения запросов . . . . .	42
<b>ЗАКЛЮЧЕНИЕ</b>		<b>45</b>
<b>СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ</b>		<b>47</b>
<b>ПРИЛОЖЕНИЕ А Презентация</b>		<b>48</b>

# ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей расчетно-пояснительной записке к курсовой работе применяются следующие сокращения и обозначения:

БД	База данных
ПО	Программное обеспечение
СУБД	Система управления базами данных
ER-диаграмма	Диаграмма «сущность-связь» (от англ. entity-relationship)
GPS	Система глобального позиционирования (Global Positioning System)
ID	Идентификатор
Use case диаграмма	Диаграмма вариантов использования

# ВВЕДЕНИЕ

Системы баз данных широко распространены в корпоративном мире как видимый инструмент — сотрудники часто напрямую взаимодействуют с такими системами, чтобы отправить данные или создать отчеты. Но не менее часто они используются как невидимые компоненты программных систем. Например, веб-сайт электронной коммерции, использующий базу данных на стороне сервера для хранения информации о клиентах, товарах и продажах. Или система навигации GPS, использующая встроенную базу данных для управления картами дорог. В обоих этих примерах система баз данных скрыта от пользователя; с ней взаимодействует только код приложения [1].

Наряду с веб-сайтами электронной коммерции, веб-приложения для покупки авиабилетов также используют базы данных для хранения информации о пользователях, самолетах, билетах и т. д. Существует мнение, что самолет — лучший способ передвижения. Эту точку зрения подкрепляют несколько факторов [2].

1. Самый быстрый путь к месту назначения.
2. Авиабилеты являются доступными для среднестатистического человека.
3. Полет — один из самых безопасных способов путешествовать [2].

В случае создания авиакомпании сегодня не представляется возможным обойтись без веб-приложения для продажи билетов. Соответственно, нужна база данных для хранения информации о пользователях, билетах, воздушных судах, услугах и т. д.

Целью курсовой работы является разработка базы данных для хранения и обработки данных авиакомпании, а также веб-приложения, которое будет ее использовать.

Задачами данной работы являются:

- 1) провести обзор существующих веб-приложений для покупки авиабилетов и сформулировать требования и ограничения к разрабатываемой базе данных и приложению;
- 2) спроектировать архитектуру базы данных, ограничения целостности и ролевую модель на уровне базы данных;

- 3) выбрать средства реализации и реализовать спроектированную базу данных и необходимый интерфейс для взаимодействия с ней;
- 4) исследовать характеристики разработанного программного обеспечения.

# **1 Аналитическая часть**

## **1.1 Анализ предметной области**

Одно из самых популярных средств для поиска авиабилетов — Aviasales. Однако это метапоисковик билетов, и сама компания перелеты не организует. Из довольно известных международных компаний, у которых есть собственный флот, можно выделить следующие: Qatar Airways, Emirates, United Airlines и Etihad Airways.

Веб-приложения приведенных организаций достаточно однотипные. На домашней странице пользователю сразу предлагается выбор пунктов вылета и назначения, типа билета — в одну сторону или «туда и обратно», даты вылета и обратного рейса, количества билетов и их класса. Помимо этого, доступны такие функции как онлайн регистрация, управление бронированием и проверка статуса рейса. Etihad Airways предлагает еще функцию повышения класса перелета.

Помимо основных функций, пользователь на сайтах этих компаний может найти и забронировать отель, экскурсию и т. д. Также стоит отметить тот факт, что эти веб-приложения доступны на большом количестве различных языков.

## **1.2 Требования к базе данных и приложению**

Приложение должно предоставлять пользователям возможность искать билеты на интересующие их рейсы. Для этого должны быть использованы следующие параметры: дата вылета, пункт вылета и пункт прибытия. Билеты могут быть нескольких классов. Помимо билетов пользователь может выбирать услуги, которые предоставляются для билетов определенных классов.

Пользователи должны иметь возможность просматривать заказ, добавлять билеты в заказ, удалять их оттуда, очищать и оплачивать заказ, добавлять и удалять услуги.

Должны быть реализованы регистрация и авторизация пользователей, внесение изменений в личные данные и возможность просмотреть информацию о самолетах авиакомпании.



## 1.3 Модели баз данных

### 1.3.1 Дореляционные базы данных

В основе любой базы данных лежит модель данных. Моделью данных называется формализованное описание структур единиц информации и операций над ними в информационной системе. Тип модели данных определяет логическую структуру базы данных и то, каким образом данные могут быть сохранены, организованы и обработаны [3].

К ранним моделям относят модели, предшествующие реляционной модели данных: иерархическую, сетевую модели и модель на основе инвертированных списков. Иерархическая модель данных была исторически первой структурой баз данных, видимо, из-за того, что древовидные иерархические структуры широко используются в повседневной человеческой деятельности [3].

Иерархическая модель данных — представление базы данных в виде древовидной (иерархической) структуры, состоящей из объектов (данных) различных уровней [3].

В этой модели данные представляются как дерево связанных записей. Имеется один корневой (родительский) тип записи — корень дерева. С ним в подчиненной связи типа 1:N находятся дочерние записи. Связи между записями выражаются в виде отношений предок-потомок, а у каждой записи есть ровно одна родительская запись. Это помогает поддерживать ссылочную целостность: когда запись удаляется из дерева, все ее потомки должны быть также удалены [3].

Иерархические базы данных имеют централизованную структуру, поэтому безопасность данных легко контролировать. Однако, определенные знания о физическом порядке хранения записей все же необходимы, так как отношения предок-потомок реализуются в виде физических указателей из одной записи на другую. Это означает, что поиск записи осуществляется методом прямого обхода дерева. Записи, расположенные в одной половине дерева, ищутся быстрее, чем в другой. Отсюда следует необходимость правильно упорядочивать записи, чтобы время их поиска было минимальным [3].

Достоинства иерархической модели.

1. Принцип построения баз данных в иерархической модели легок для

понимания. Иерархия базы данных напоминает структуру компании или генеалогическое дерево.

2. Использование отношений предок-потомок. Иерархическая модель позволяет легко представлять отношения предок-потомок, например, «А является частью В» или «А принадлежит В».
3. Быстродействие. В иерархической модели отношения предок-потомок реализуются в виде физических указателей из одной записи на другую, поэтому перемещение по базе данных происходит достаточно быстро. Поскольку структура данных в иерархической модели отличается простотой, СУБД может размещать записи предков и потомков на диске рядом друг с другом, что позволяло свести к минимуму количество операций чтения-записи [3].

Недостатки иерархической модели.

1. Операции манипулирования данными в иерархических системах ориентированы прежде всего на поиск информации сверху вниз, т. е. по данному экземпляру сегмента-отца можно найти все экземпляры сегментов-сыновей. Обратный поиск затруднен, а часто и невозможен. Например, попытка реализовать запрос типа «В скольких сборниках статей опубликовал свои статьи господин Петров?» может оказаться весьма трудной задачей.
2. Дублирование данных на логическом уровне.
3. Для представления связи M:N необходимо дублирование деревьев.
4. В иерархической модели автоматически поддерживается целостность ссылок между предками и потомками по правилу: никакой потомок не может существовать без своего родителя. Целостность по ссылкам между записями, не входящими в одну иерархию, не поддерживается. Поэтому невозможно хранение в базе данных порожденного узла без соответствующего исходного. Аналогично, удаление исходного узла влечет удаление всех порожденных узлов (деревьев), связанных с ним [3].

Сетевой подход к организации данных является расширением иерархического. Цель разработчиков сетевой модели — создание модели, позволяющей описывать связи M:N, чтобы одна запись могла участвовать в нескольких отношениях предок-потомок [3].

Сетевая модель данных базируется также на использовании представления данных в виде графа. С точки зрения теории графов сетевой модели соответствует произвольный граф: если в иерархической модели запись-потомок должна иметь в точности одного предка, то в сетевой модели данных потомок может иметь любое число предков. Вершины графа используются для интерпретации типов объектов, дуги графа используются для интерпретации типов связей между типами объектов [3].

Структура сетевой базы данных основана на следующих правилах:

- 1) база данных содержит любое количество типов записей и типов наборов;
- 2) между двумя типами записей может быть определено любое количество типов наборов;
- 3) тип записи может быть владельцем и одновременно членом нескольких типов наборов [3].

Сложность практического использования иерархических и сетевых СУБД заставляла искать иные способы представления данных. В конце 1960-х годов появились СУБД на основе инвертированных файлов, отличающиеся простотой организации и наличием весьма удобных языков манипулирования данными. Организация доступа к данным на основе инвертированных списков используется практически во всех современных реляционных СУБД, но в реляционных СУБД пользователи не имеют непосредственного доступа к инвертированным спискам (индексам) [3].

База данных на инвертированных списках похожа на реляционную базу данных, т. е. также состоит из таблиц отношений, однако ей присущи важные отличия:

- 1) допускается сложная структура атрибутов (атрибуты не обязательно атомарны);

- 2) строки таблиц (записи) упорядочены в некоторой последовательности, каждой строке присваивается уникальный номер, физическая упорядоченность строк всех таблиц может определяться и для всей базы данных;
- 3) пользователям видны и хранимые таблицы, и пути доступа к ним;
- 4) пользователь может управлять логическим порядком строк в каждой таблице с помощью специального инструмента — индексов (эти индексы автоматически поддерживаются системой и явно видны пользователям) [3].

Недостаток модели — отсутствие строгого математического аппарата, отсутствие средств для описания ограничений целостности базы данных и, как следствие — большая трудоемкость программирования запросов к базе данных. В некоторых системах поддерживаются ограничения уникальности значений некоторых полей, но в основном все возлагается на прикладную программу. Кроме этого, такие СУБД обладают рядом ограничений на количество файлов для хранения данных, количество связей между ними, длину записи и количество ее полей [3].

### 1.3.2 Постреляционные базы данных

Третье поколение баз данных, называемых постреляционные, начало развиваться с 90-годов XX века. Тогда появились объектные, объектно-реляционные и полуструктурированные базы данных, которые расширяют возможности реляционных баз данных и позволяют хранить и обрабатывать как атомарные значения, так и объекты со сложной структурой. Объектные базы данных, основанные на объектно-ориентированной парадигме, — альтернатива реляционному подходу. Объектно-реляционные базы данных поддерживают обратную совместимость с реляционными базами и расширяют их возможности. Полуструктурированные базы данных развиваются параллельно на основе сетевых и иерархических баз данных и позволяют работать с частично структурированными данными [4].

Объектный подход к созданию баз данных ориентирован на объединение возможностей объектно-ориентированного языка программирования и базы данных для совместной работы. Он предполагает постоянное хранение

объектов программы в базе данных и формирование запросов к базе данных на языке написания программы [4].

Объектная СУБД обладает следующими возможностями:

- 1) поддерживает сложную систему типов, в том числе атомарные, структуры, коллекции и ссылки и предоставляет средства для их описания;
- 2) разделяет работу с литералами (неизменяемыми значениями любых типов) и объектами классов;
- 3) обеспечивает идентификацию объектов;
- 4) позволяет определять иерархии классов и интерфейсов [4].

Объектно-реляционная модель БД предоставляет пользователю следующие возможности:

- 1) хранить и обрабатывать сложные типы данных (структуры и коллекции);
- 2) наряду с предопределенными типами данных использовать пользовательские типы данных;
- 3) включать в пользовательские типы данных атрибуты и методы;
- 4) поддерживать уникальные идентификаторы кортежей и ссылки на них;
- 5) определять иерархию (наследование) типов данных [4].

Поддерживает следующие типы данных:

- 1) атомарные;
- 2) структуры, содержащие набор именованных полей;
- 3) массивы, хранящие упорядоченный набор элементов;
- 4) множества и мультимножества, хранящие неупорядоченный набор элементов;
- 5) пользовательские типы данных;

- 6) ссылки на объекты (кортежи) пользовательских типов данных [4].

Преимущество баз данных с фиксированной схемой, например, реляционных, — высокая эффективность обработки и хранения больших объемов данных. Основным преимуществом полуструктурированных баз данных является их гибкость. Модель полуструктурированных данных позволяет хранить внутри данных не только значения, но и их структуру и изменять ее со временем. Полуструктурированные данные изображают с помощью графа. Вершины графа — информационные элементы, а дуги — связи между ними [4].

Полуструктурированные данные применяют:

- 1) для описания схожей информации в БД с отличающимися схемами данных при их интеграции;
- 2) для представления документов (по аналогии с языком XML для веб-сайтов);
- 3) для работы с унаследованными БД при укрупнении, поглощении компаний (описывают обобщенную структуру исходных БД) [4].

### **1.3.3 Реляционные базы данных**

Реляционные базы данных относят ко второму поколению баз данных. Они появились в начале 70-х годов XX века и активно использовались до конца 90-х годов XX века. Считается, что их теорию сформулировал Эдгар Кодд. Особенность реляционных баз данных состоит в том, что все данные и связи между ними хранятся в таблицах. Для определения структуры данных и манипулирования их значениями используют язык SQL (Structured Query Language — структурированный язык запросов) [4].

Реляционная модель БД — логическая модель, базирующаяся на концептуальной модели, например, модели сущность–связь, и используемая для построения схемы (структуры) реляционной БД. Управление реляционной БД осуществляется реляционной СУБД [4].

Факторы, обеспечившие быстрое распространение реляционной модели:

- 1) с прагматической точки зрения база данных представляется в виде двумерных таблиц (отношений), обработка в которых не зависит от организации хранения данных в памяти;

- 2) с математической точки зрения реляционная база данных — конечный набор отношений различной арифности, являющихся областью приложений математической логики, теории множеств и общей алгебры, замкнутость реляционной модели (операции над отношениями дают отношение) обеспечивает основу для интерпретации выводимости, избыточности и непротиворечивости данных;
- 3) наличие небольшого набора абстракций, которые позволяют сравнительно просто моделировать большую часть распространенных предметных областей и допускают точные формальные определения [3].

## **1.4 Информация, подлежащая хранению в базе данных**

В базе данных нужно будет хранить информацию о шести сущностях: пользователь, заказ, билет, рейс, самолет и услуга.

1. Пользователь. Информация о зарегистрированных пользователях: роль, адрес электронной почты, пароль в зашифрованном виде, имя, фамилия и дата регистрации.
2. Заказ. Информация о заказах зарегистрированных пользователей: заказчик билета и статус заказа.
3. Рейс. Пункт вылета, пункт прибытия, дата и время вылета, дата и время прибытия.
4. Билет. Рейс, на который продается билет, возможность бронирования, ряд, место, класс, возможность возврата и стоимость.
5. Самолет. Название компании-производителя, модель, количество мест эконом класса, бизнес-класса и первого класса.
6. Услуга. Наименование, стоимость, возможность получения данной услуги для билета эконом класса, бизнес-класса и первого класса.

## **1.5 ER-диаграмма сущностей базы данных**

На рисунке 1.1 показана диаграмма «сущность-связь» проектируемой базы данных в нотации Чена. Представлены шесть сущностей и их свойства.

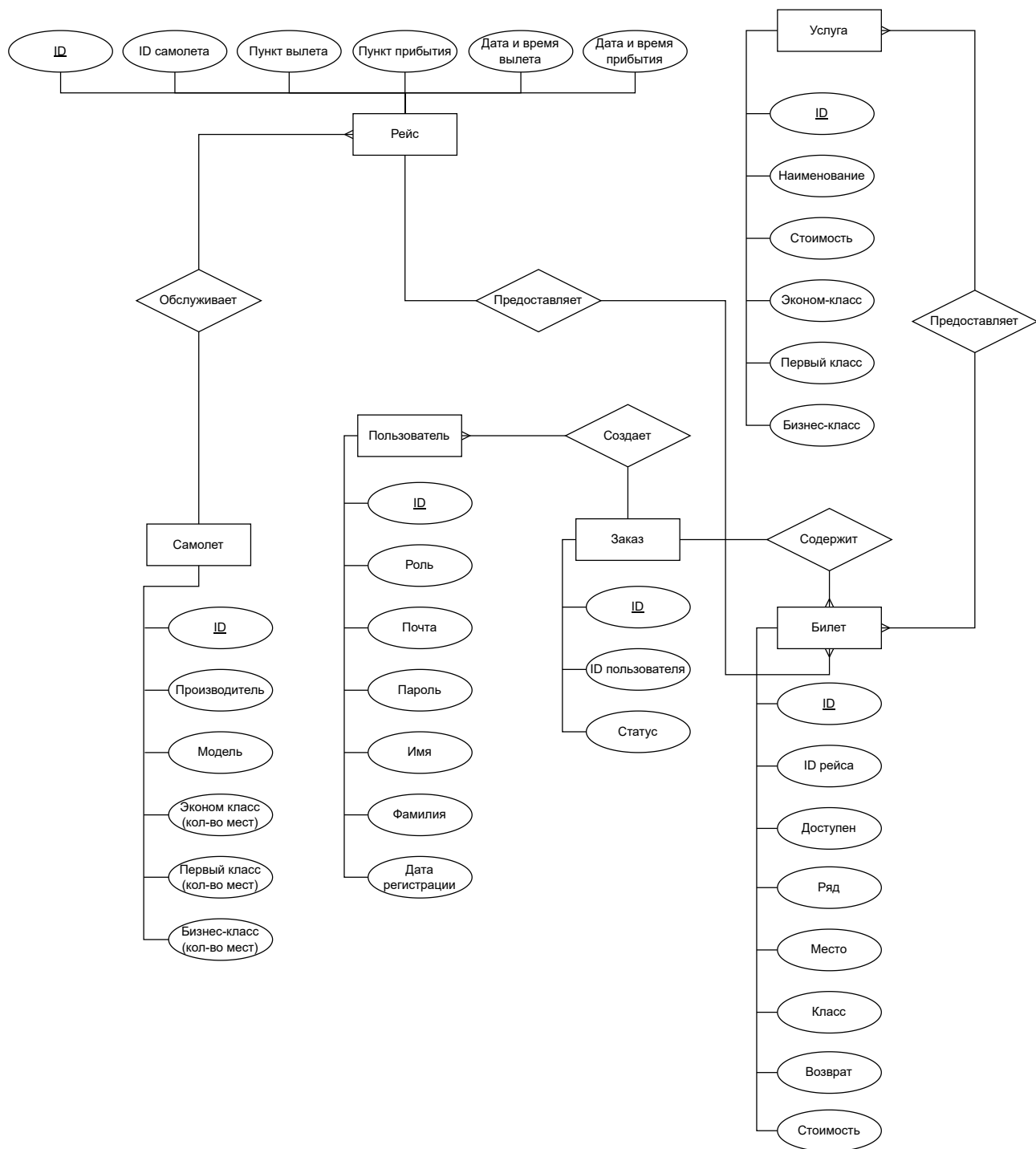


Рисунок 1.1 – ER-диаграмма

## 1.6 Пользователи приложения

Пользователи приложения могут быть четырех типов: посетители, клиенты, модераторы и администраторы.

Посетитель — неавторизованный пользователь. Он может только искать рейсы и билеты, просматривать самолеты компании, регистрироваться и входить в личный кабинет.



Клиент — зарегистрированный и авторизованный пользователь. Он может взаимодействовать с заказом: может его просмотреть, добавить и удалить билеты, добавить и удалить услуги, очистить и оплатить заказ. Также у него должна быть возможность внести изменения в свои данные — адрес электронной почты, имя и фамилию.

Модератор — зарегистрированный и авторизованный пользователь, который может добавлять в базу данных новые рейсы и билеты и удалять их оттуда.

Администратор — модератор, у которого есть полный доступ к базе данных. Он должен иметь возможность удалять учетные записи пользователей в приложении. Помимо этого, администратор может напрямую работать с базой данных, в том числе, создавать и удалять таблицы.

## 1.7 Диаграмма вариантов использования

На рисунках 1.2–1.5 показаны use case диаграммы для разных пользователей.

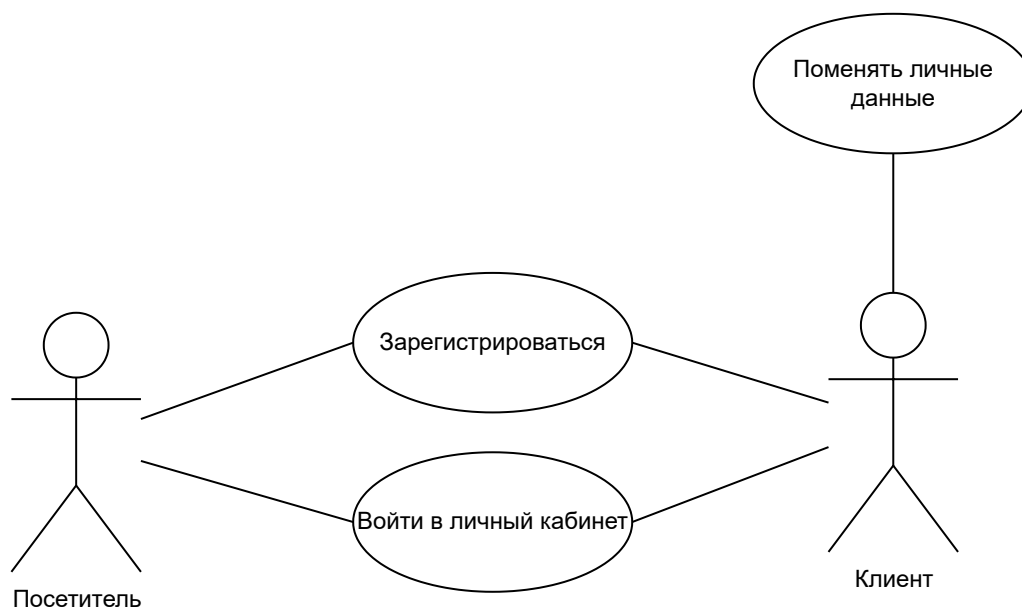


Рисунок 1.2 – Взаимодействие пользователя с личными данными

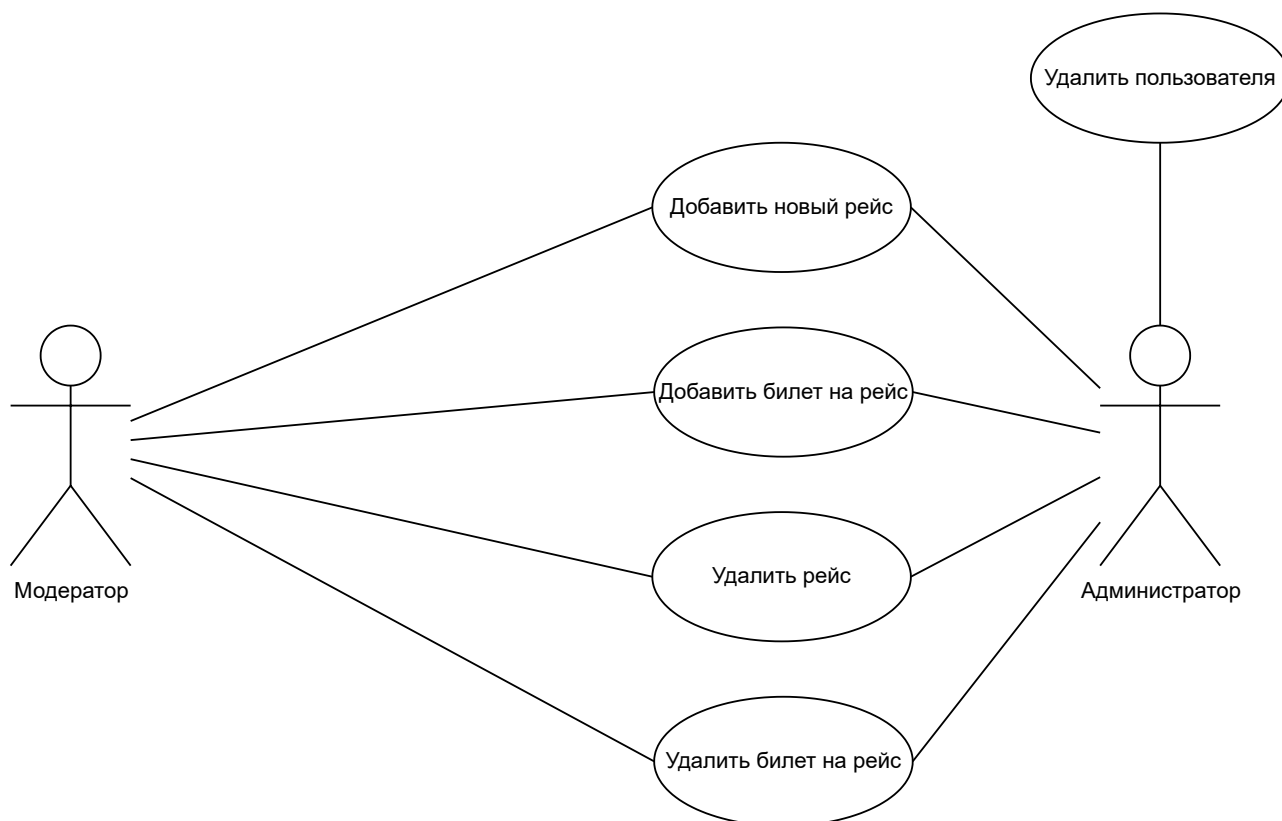


Рисунок 1.3 – Администрирование

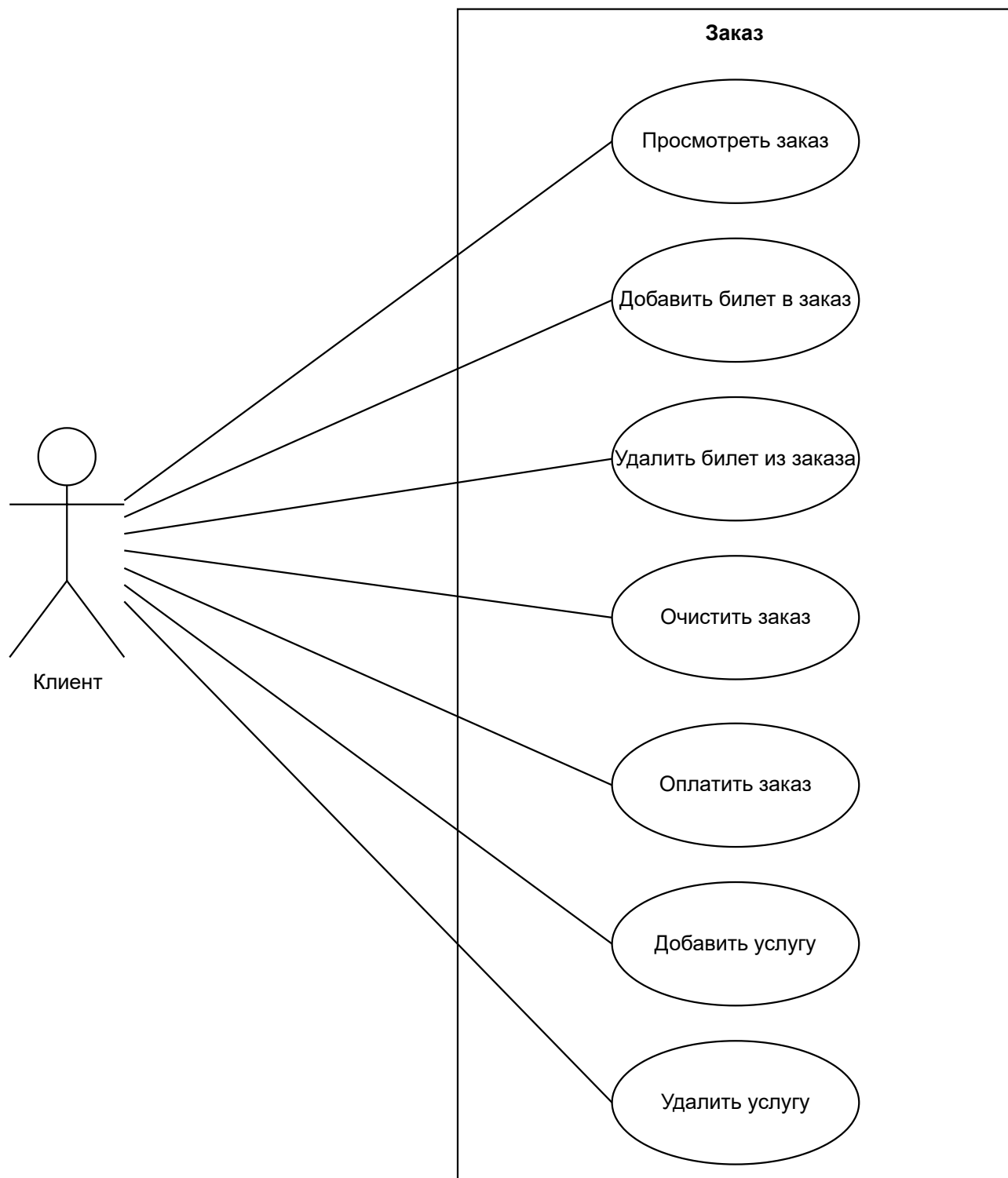


Рисунок 1.4 – Взаимодействие пользователя с заказом

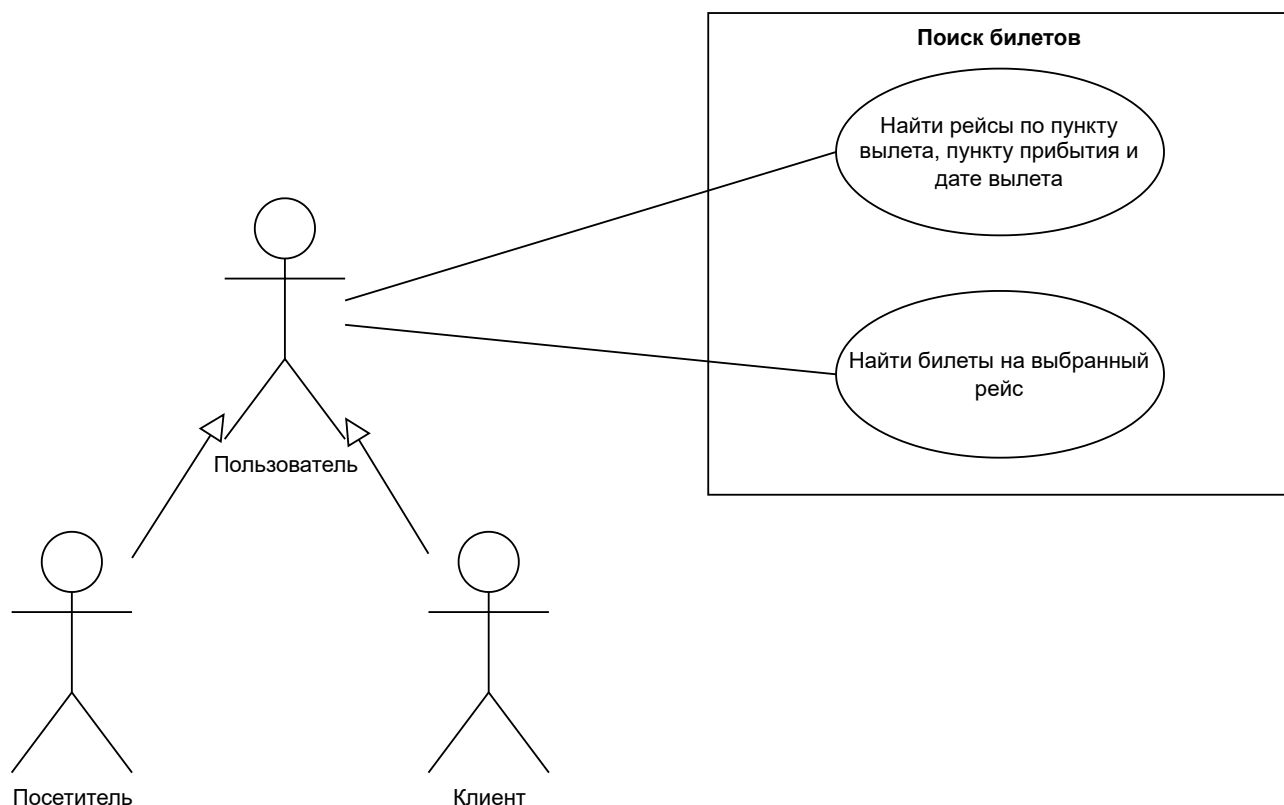


Рисунок 1.5 – Поиск рейсов и билетов

## Вывод из аналитической части

В ходе выполнения аналитической части курсовой работы был проведен анализ предметной области; сформулированы требования к базе данных и приложению; выделены шесть сущностей базы данных — пользователь, заказ, билет, рейс, услуга и самолет; проведен анализ существующих баз данных на основе формализации данных и выбрана реляционная модель, описаны пользователи приложения (посетители, клиенты, модераторы и администраторы) и пользовательские сценарии в виде use case диаграмм.

## 2 Конструкторская часть

### 2.1 Описание сущностей базы данных

В аналитической части были выделены шесть сущностей базы данных: пользователь, рейс, билет, самолет, услуга и заказ. На рисунке 2.1 представлена диаграмма базы данных.

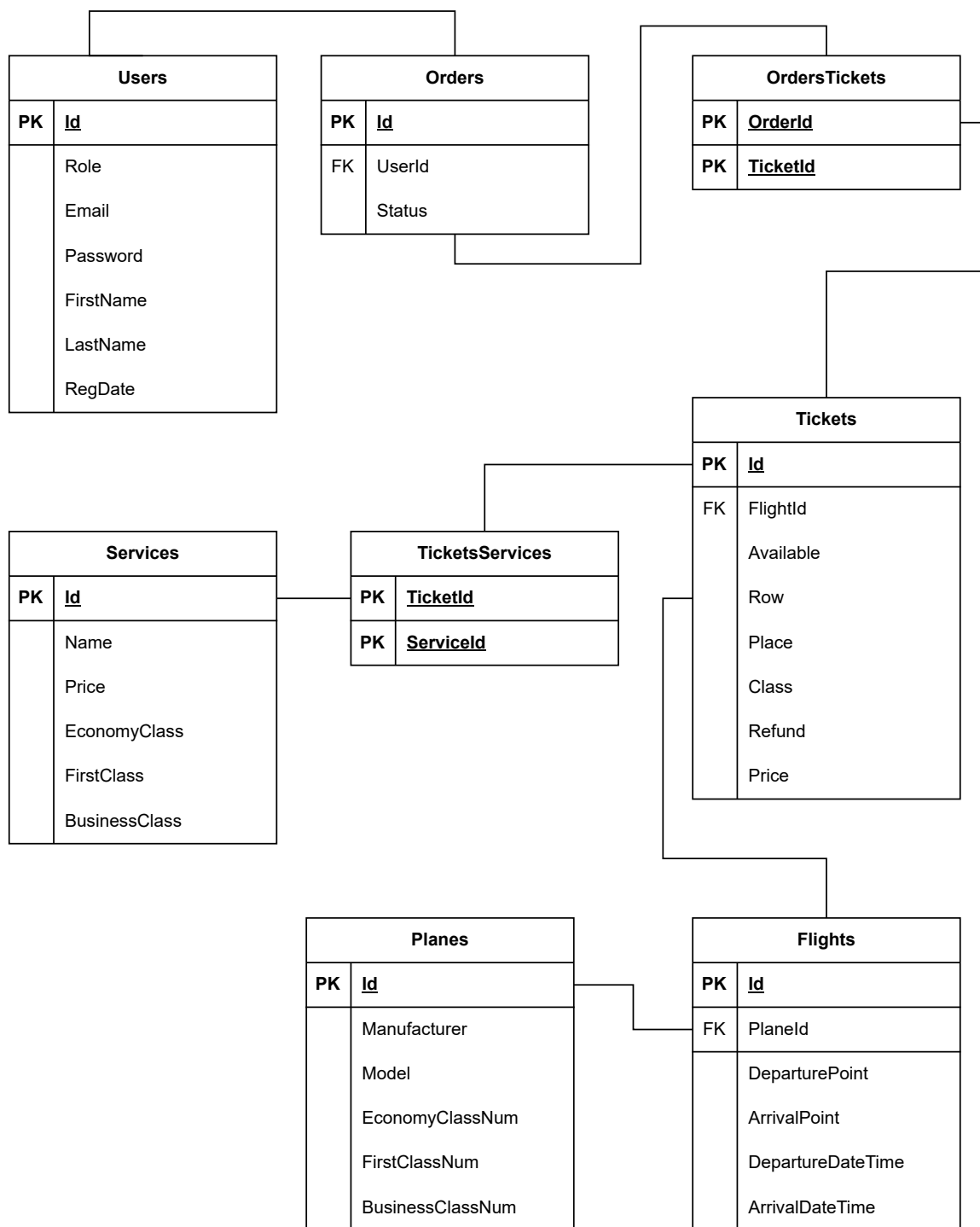


Рисунок 2.1 – Диаграмма базы данных

В таблицах 2.1–2.6 описаны атрибуты таблиц базы данных.

Таблица 2.1 – Атрибуты таблицы «Пользователи»

Имя атрибута	Тип атрибута	Описание
Id	bigint	Идентификатор — первичный ключ
Role	varchar	Роль пользователя
Email	varchar	Адрес электронной почты
Password	varchar	Зашифрованный пароль
FirstName	varchar	Имя пользователя
LastName	varchar	Фамилия пользователя
RegDate	date	Дата регистрации

Таблица 2.2 – Атрибуты таблицы «Заказы»

Имя атрибута	Тип атрибута	Описание
Id	bigint	Идентификатор — первичный ключ
UserId	bigint	Идентификатор пользователя — внешний ключ
Status	varchar	Статус заказа

Таблица 2.3 – Атрибуты таблицы «Билеты»

Имя атрибута	Тип атрибута	Описание
Id	bigint	Идентификатор — первичный ключ
FlightId	bigint	Идентификатор рейса — внешний ключ
Available	bit	Доступен ли билет для заказа
Row	int	Ряд в самолете
Place	char	Место в ряду
Class	varchar	Класс билета
Refund	bit	Возможен ли возврат билета
Price	money	Стоимость

Таблица 2.4 – Атрибуты таблицы «Рейсы»

Имя атрибута	Тип атрибута	Описание
Id	bigint	Идентификатор — первичный ключ
PlaneId	bigint	Идентификатор самолета — внешний ключ
DeparturePoint	varchar	Пункт вылета
ArrivalPoint	varchar	Пункт прибытия
DepartureDateTime	datetime	Дата и время вылета
ArrivalDateTime	datetime	Дата и время прибытия

Таблица 2.5 – Атрибуты таблицы «Самолеты»

Имя атрибута	Тип атрибута	Описание
Id	bigint	Идентификатор — первичный ключ
Manufacturer	varchar	Название компании-производителя
Model	varchar	Название модели самолета
EconomyClassNum	int	Количество мест эконом класса
BusinessClassNum	int	Количество мест бизнес-класса
FirstClassNum	int	Количество мест первого класса

Таблица 2.6 – Атрибуты таблицы «Услуги»

Имя атрибута	Тип атрибута	Описание
Id	bigint	Идентификатор — первичный ключ
Name	varchar	Название услуги
Price	money	Стоимость услуги
EconomyClass	bit	Доступна ли услуга для билета эконом класса
BusinessClass	bit	Доступна ли услуга для билета бизнес-класса
FirstClass	bit	Доступна ли услуга для билета первого класса

## 2.2 Описание ограничений целостности базы данных

На данные, хранящиеся в базе, должны быть наложены определенные ограничения для обеспечения ее целостности.

Все идентификаторы должны быть положительными целыми числами.

Ограничения для таблицы «Пользователи». Роль пользователя не может отсутствовать. Адрес электронной почты не может быть равен NULL, должен состоять из двух частей, разделенных символом «@», а также должен быть уникальным. Пароль не может отсутствовать, должен состоять не менее чем из 6-ти символов и обязательно должен храниться в зашифрованном виде. Дата регистрации пользователя не должна отсутствовать.

Ограничения для таблицы «Заказы». Статус — строка «создан» или «оплачен», не должен отсутствовать.

Ограничения для таблицы «Рейсы». Пункт вылета и прибытия — строки, не должны отсутствовать. Дата и время вылета и прибытия не должны отсутствовать.

Ограничения для таблицы «Билеты». Билет доступен для бронирования — логический тип (истина или ложь). Ряд в самолете — положительное число, не должен отсутствовать. Место в ряду — буква от А до К, не должно отсутствовать. Класс — строка «эконом», «первый» или «бизнес», не должен отсутствовать. Возможность возврата — логический тип (истина или ложь). Стоимость — положительное число, не должна отсутствовать.

Ограничения для таблицы «Самолеты». Производитель и модель — строки, не должны отсутствовать. Количество мест разных классов — неотрицательные числа, не должны отсутствовать.

Ограничения для таблицы «Услуги». Наименование услуги — строка, не должна отсутствовать. Стоимость — положительное число, не должна отсутствовать. Доступность для мест разных классов — логический тип (истина или ложь).



## 2.3 Описание проектируемой функции на уровне базы данных

Пользователь приложения может заказывать билеты и услуги. Так как они находятся в разных таблицах, нужно выполнять два запроса для того, чтобы посчитать стоимость всех билетов и услуг, заказанных пользователем. Для этой цели удобно реализовать функцию, которая по идентификатору заказа будет возвращать его стоимость. На рисунке 2.2 представлена схема проектируемой функции.

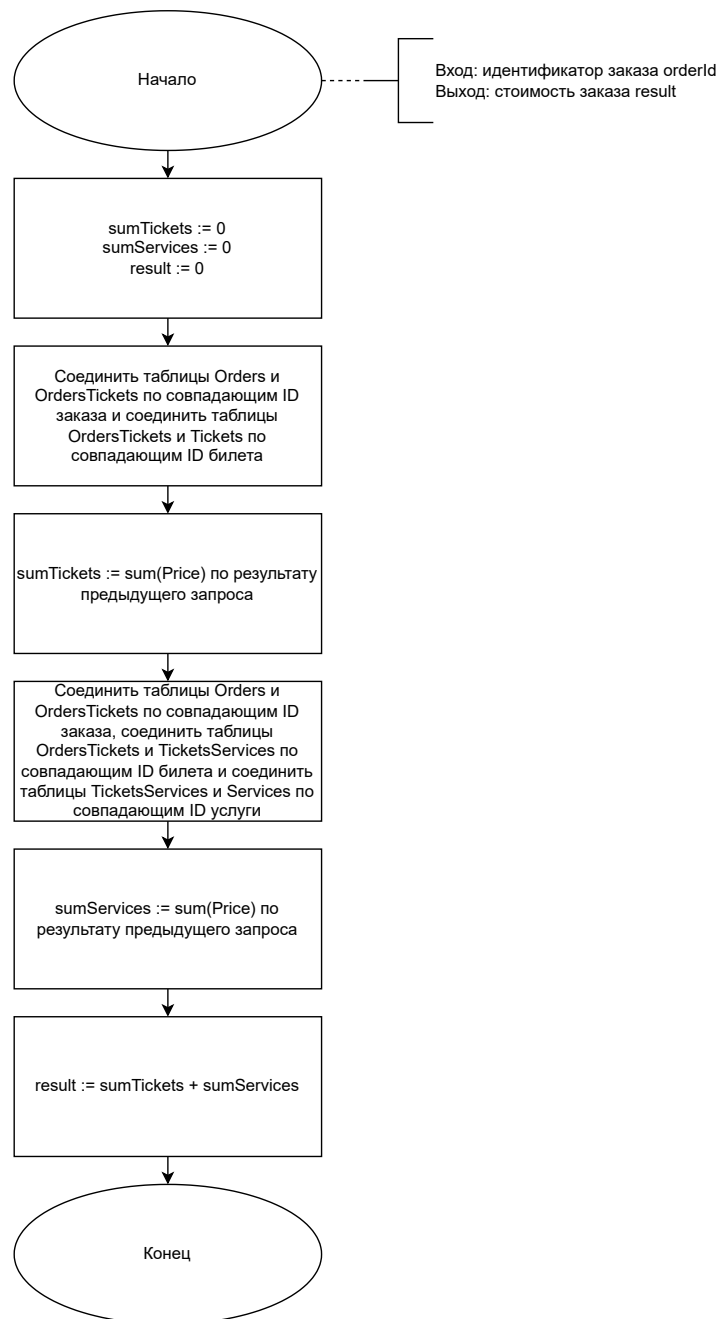


Рисунок 2.2 – Функция, возвращающая стоимость заказа

## **2.4 Описание ролевой модели на уровне базы данных**

В аналитической части было указано, что в приложении есть четыре типа пользователей: посетители, клиенты, модераторы и администраторы. Взаимодействием пользователей с базой данных должно происходить не напрямую, а с помощью определенного промежуточного слоя. Таким образом, для посетителя можно вообще не создавать роль на уровне базы данных. При попытке зазвать билет, пользователь будет переадресован на страницу регистрации. Для трех других типов пользователей нужно создавать роли на уровне базы данных.

Доступны три роли на уровне базы данных.

1. Клиент. Может получать информацию из таблиц «Пользователи», «Заказы», «Билеты», «Рейсы», «Самолеты» и «Услуги». Может изменять статус билета (можно бронировать/нельзя бронировать), добавлять билеты в таблицу «Заказы», а также может изменять информацию о себе в таблице «Пользователи».
2. Модератор. Обладает правами клиента. Не может вносить изменения в таблицу «Пользователи», кроме информации о себе. Может добавлять и удалять данные в таблицах «Рейсы» и «Билеты».
3. Администратор. Обладает правами модератора. Может удалять данные из таблицы «Пользователи».

### **Вывод из конструкторской части**

В ходе выполнения конструкторской части курсовой работы была создана диграмма базы данных; описаны сущности базы данных; описаны проектируемые ограничения целостности базы данных; описана функция на уровне базы данных, которая считает стоимость заказа, в виде схемы алгоритма и описана проектируемая ролевая модель на уровне базы данных, которая состоит из трех ролей — клиент, модератор и администратор.

## 3 Технологическая часть

### 3.1 Средства реализации

Веб-приложения бывают одностраничными (SPA) и многостраничными (MPA). Оба подхода имеют свои преимущества и недостатки. В таблице 3.1 перечислены преимущества и недостатки одностраничных приложений.

Таблица 3.1 – Преимущества и недостатки одностраничных приложений [5]

Преимущества	Недостатки
Быстрая загрузка и обновление страниц	Тяжелые фреймворки
Удобство использования	Интерпретируемые языки
Изолирование фронтенда и бэкенда	Поисковая оптимизация

Одностраничные приложения пишутся на языке JavaScript. Для создания многостраничного приложения можно обойтись знанием языка гипертекстовой разметки (HTML) и языка таблиц стилей (CSS) для написания клиентской части, а для написания серверной — Python, Ruby, Java, C#, Go, PHP и т. д. [5]

Наиболее популярные реляционные системы управления базами данных: Oracle, MySQL, Microsoft SQL Server, PostgreSQL и SQLite [6][7].

Для реализации программного обеспечения был выбран язык программирования C# и СУБД Microsoft SQL Server ввиду следующих причин:

- 1) на языке программирования C# можно создавать веб-приложения с помощью кроссплатформенного фреймворка ASP.NET Core;
- 2) C# поддерживает СУБД Microsoft SQL Server;
- 3) фреймворк ASP.NET Identity представляет встроенную систему аутентификации и авторизации;
- 4) в стандартной библиотеке C# реализованы необходимые структуры данных для работы с БД (такие как `IEnumerable<T>`, `List<T>` и `SortedSet<T>`).

Таким образом, с помощью языка C# и СУБД Microsoft SQL Server можно реализовать программное обеспечение, которое соответствует перечисленным выше требованиям.

## 3.2 Реализация сущностей базы данных

В листингах 3.1–3.6 показана реализация сущностей базы данных.

Листинг 3.1 – Сущность «Пользователь»

```
public class User
{
    private IUserRepository<User> _db;

    [Key]
    public Int64 Id { get; set; }
    public string Role { get; set; }
    public string Email { get; set; }
    public string Password { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public DateTime RegDate { get; set; }
}
```

Листинг 3.2 – Сущность «Рейс»

```
public class Flight
{
    private IFlightRepository<Flight> _db;
    [Key]
    public Int64 Id { get; set; }
    public string DeparturePoint { get; set; }
    public string ArrivalPoint { get; set; }
    public DateTime DepartureDateTime { get; set; }
    public DateTime ArrivalDateTime { get; set; }
}
```

### Листинг 3.3 – Сущность «Заказ»

```
public class Order
{
    private IOrderRepository<Order> _db;

    [Key]
    public Int64 Id { get; set; }
    public Int64 UserId { get; set; }
    public string Status { get; set; }
}
```

### Листинг 3.4 – Сущность «Билет»

```
public class Ticket
{
    private ITicketRepository<Ticket> _db;

    [Key]
    public Int64 Id { get; set; }
    public Int64 FlightId { get; set; }
    public bool Available { get; set; }
    public int Row { get; set; }
    public char Place { get; set; }
    public string Class { get; set; }
    public bool Refund { get; set; }
    public Decimal Price { get; set; }
}
```

### Листинг 3.5 – Сущность «Самолет»

```
public class Plane
{
    private IPlaneRepository<Plane> _db;

    [Key]
    public Int64 Id { get; set; }
    public string Manufacturer { get; set; }
    public string Model { get; set; }
    public uint EconomyClassNum { get; set; }
    public uint FirstClassNum { get; set; }
    public uint BusinessClassNum { get; set; }
}
```

### Листинг 3.6 – Сущность «Услуга»

```
public class Service
{
    private IServiceRepository<Service> _db;

    [Key]
    public Int64 Id { get; set; }
    public string Name { get; set; }
    public decimal Price { get; set; }
    public bool EconomyClass { get; set; }
    public bool FirstClass { get; set; }
    public bool BusinessClass { get; set; }
}
```

## 3.3 Реализация ограничений целостности базы данных

В листингах 3.7–3.15 показана реализация ограничений целостности базы данных (ограничения задаются при создании таблиц).

### Листинг 3.7 – Ограничения для таблицы «Пользователи»

```
IF OBJECT_ID('[Users]', 'U') IS NULL
CREATE TABLE [Users] (
    [Id] [bigint] IDENTITY(1, 1),
    [Role] [varchar](32) NOT NULL,
    [Email] [varchar](32) NOT NULL UNIQUE,
    [Password] [varchar](64) NOT NULL,
    [FirstName] [nvarchar](32),
    [LastName] [nvarchar](32),
    [RegDate] [date] NOT NULL,
    PRIMARY KEY (Id),
);
```

Листинг 3.8 – Ограничения для таблицы «Рейсы»

```
IF OBJECT_ID('[Flights]', 'U') IS NULL
CREATE TABLE [Flights] (
    [Id] [bigint] IDENTITY(1, 1),
    [DeparturePoint] [nvarchar](32) NOT NULL,
    [ArrivalPoint] [nvarchar](32) NOT NULL,
    [DepartureDate] [date] NOT NULL,
    [ArrivalDate] [date] NOT NULL,
    [DepartureTime] [time] NOT NULL,
    [ArrivalTime] [time] NOT NULL,
    PRIMARY KEY (Id),
);
```

Листинг 3.9 – Ограничения для таблицы «Заказы»

```
IF OBJECT_ID('[Orders]', 'U') IS NULL
CREATE TABLE [Orders] (
    [Id] [bigint] IDENTITY(1, 1),
    [UserId] [bigint],
    [Status] [nvarchar](16),
    PRIMARY KEY (Id),
    CONSTRAINT FK_USERS_ID
    FOREIGN KEY (UserId) REFERENCES [Users] (Id)
    ON DELETE CASCADE
);
```

Листинг 3.10 – Ограничения для таблицы «Самолеты»

```
IF OBJECT_ID('[Planes]', 'U') IS NULL
CREATE TABLE [Planes] (
    [Id] [bigint] IDENTITY(1, 1),
    [Manufacturer] [nvarchar](32) NOT NULL,
    [Model] [nvarchar](16) NOT NULL,
    [EconomyClassNum] [integer] NOT NULL,
    [FirstClassNum] [integer] NOT NULL,
    [BusinessClassNum] [integer] NOT NULL,
    PRIMARY KEY (Id),
);
```

Листинг 3.11 – Ограничения для таблицы «Билеты»

```
IF OBJECT_ID('[Tickets]', 'U') IS NULL
CREATE TABLE [Tickets] (
    [Id] [bigint] IDENTITY(1, 1),
    [FlightId] [bigint] NOT NULL,
    [Available] [bit] NOT NULL,
    [Row] [int] NOT NULL,
    [Place] [char] NOT NULL,
    [Class] [nvarchar](16) NOT NULL,
    [Refund] [bit] NOT NULL,
    [Price] [money]
    PRIMARY KEY (Id),
    CONSTRAINT FK_FLIGHTS_2_ID
    FOREIGN KEY (FlightId) REFERENCES [Flights] (Id)
    ON DELETE CASCADE
);
```

Листинг 3.12 – Ограничения для таблицы «Услуги»

```
IF OBJECT_ID('[Services]', 'U') IS NULL
CREATE TABLE [Services] (
    [Id] [bigint] IDENTITY(1, 1),
    [Name] [nvarchar](64) NOT NULL,
    [Price] [money] NOT NULL,
    [EconomyClass] [bit] NOT NULL,
    [FirstClass] [bit] NOT NULL,
    [BusinessClass] [bit] NOT NULL,
    PRIMARY KEY (Id),
);
```



Листинг 3.13 – Ограничения для вспомогательной таблицы «Самолеты-рейсы»

```
IF OBJECT_ID('[PlanesFlights]', 'U') IS NULL
CREATE TABLE [PlanesFlights] (
    [PlaneId] [bigint],
    [FlightId] [bigint],
    PRIMARY KEY (PlaneId, FlightId),
    CONSTRAINT FK_PLANES_ID
    FOREIGN KEY (PlaneId) REFERENCES [Planes] (Id)
    ON DELETE CASCADE,
    CONSTRAINT FK_FLIGHTS_ID
    FOREIGN KEY (FlightId) REFERENCES [Flights] (Id)
    ON DELETE CASCADE
);
```

Листинг 3.14 – Ограничения для вспомогательной таблицы «Заказы-билеты»

```
IF OBJECT_ID('[OrdersTickets]', 'U') IS NULL
CREATE TABLE [OrdersTickets] (
    [OrderId] [bigint],
    [TicketId] [bigint],
    PRIMARY KEY (OrderId, TicketId),
    CONSTRAINT FK_ORDERS_ID
    FOREIGN KEY (OrderId) REFERENCES [Orders] (Id)
    ON DELETE CASCADE,
    CONSTRAINT FK_TICKETS_ID
    FOREIGN KEY (TicketId) REFERENCES [Tickets] (Id)
    ON DELETE CASCADE
);
```

Листинг 3.15 – Ограничения для вспомогательной таблицы «Билеты-услуги»

```
IF OBJECT_ID('[TicketsServices]', 'U') IS NULL
CREATE TABLE [TicketsServices] (
    [TicketId] [bigint],
    [ServiceId] [bigint],
    PRIMARY KEY (TicketId, ServiceId),
    CONSTRAINT FK_TICKETS_2_ID
    FOREIGN KEY (TicketId) REFERENCES [Tickets] (Id)
    ON DELETE CASCADE,
    CONSTRAINT FK_SERVICES_ID
    FOREIGN KEY (ServiceId) REFERENCES [Services] (Id)
    ON DELETE CASCADE
);
```

### 3.4 Реализация функции на уровне базы данных

В листинге 3.16 показана реализация функции на уровне базы данных, которая по идентификатору заказа считает стоимость всех билетов и услуг. Здесь используется функция COALESCE из Transact-SQL, которая вычисляет аргументы по порядку и возвращает текущее значение первого выражения, изначально не вычисленного как NULL [8].

Листинг 3.16 – Функция, возвращающая стоимость всех билетов и услуг конкретного заказа

```
CREATE FUNCTION GetOrderPrice (@OrderId bigint)
    RETURNS money
    BEGIN
        DECLARE @sumTickets money, @sumServices money, @result
            money
        SET @sumTickets = 0;
        SET @sumServices = 0;
        SET @result = 0;
        SELECT @sumTickets = COALESCE(SUM(t.Price), 0) FROM
            Orders o
            JOIN OrdersTickets ot ON o.Id = ot.OrderId JOIN
            Tickets t ON ot.TicketId = t.Id WHERE @OrderId = o.Id;
        SELECT @sumServices = COALESCE(SUM(s.Price), 0) FROM Orders o
            JOIN OrdersTickets ot ON o.Id = ot.OrderId JOIN
            TicketsServices ts ON ot.TicketId = ts.TicketId
            JOIN Services s ON s.Id = ts.ServiceId WHERE @OrderId =
                o.Id;
        SELECT @result = @sumTickets + @sumServices
        RETURN @result
    END;
```

## 3.5 Реализация ролевой модели на уровне базы данных

В листинге 3.17 показана реализация ролевой модели на уровне базы данных, которая состоит из трех ролей: клиента (db\_customer), модератора (db\_moderator) и администратора (db\_admin).

Листинг 3.17 – Ролевая модель на уровне базы данных

```
CREATE LOGIN customer WITH PASSWORD = 'customer';
CREATE USER customer FOR LOGIN customer;
CREATE ROLE db_customer;

GRANT EXECUTE ON GetOrderPrice TO db_customer;
GRANT SELECT ON Flights TO db_customer;
GRANT SELECT, UPDATE, INSERT ON Orders TO db_customer;
GRANT SELECT, UPDATE, INSERT, DELETE ON OrdersTickets TO
    db_customer;
GRANT SELECT ON Planes TO db_customer;
GRANT SELECT ON PlanesFlights TO db_customer;
GRANT SELECT ON Services TO db_customer;
GRANT SELECT, UPDATE ON Tickets TO db_customer;
GRANT SELECT, UPDATE, INSERT, DELETE ON TicketsServices TO
    db_customer;
GRANT SELECT, UPDATE, INSERT ON Users TO db_customer;

ALTER ROLE db_customer ADD MEMBER customer;

CREATE LOGIN moderator WITH PASSWORD = 'moderator';
CREATE USER moderator FOR LOGIN moderator;
CREATE ROLE db_moderator;

GRANT EXECUTE ON GetOrderPrice TO db_moderator;
GRANT SELECT, UPDATE, INSERT, DELETE ON Flights TO db_moderator;
GRANT SELECT, UPDATE, INSERT ON Orders TO db_moderator;
GRANT SELECT, UPDATE, INSERT, DELETE ON OrdersTickets TO
    db_moderator;
GRANT SELECT, UPDATE, INSERT, DELETE ON Planes TO db_moderator;
GRANT SELECT, UPDATE, INSERT, DELETE ON PlanesFlights TO
    db_moderator;
GRANT SELECT, UPDATE, INSERT, DELETE ON Services TO db_moderator;
GRANT SELECT, UPDATE, INSERT, DELETE ON Tickets TO db_moderator;
GRANT SELECT, UPDATE, INSERT, DELETE ON TicketsServices TO
```

```

    db_moderator;
GRANT SELECT, UPDATE, INSERT ON Users TO db_moderator;

ALTER ROLE db_moderator ADD MEMBER moderator;

CREATE LOGIN admin WITH PASSWORD = 'admin';
CREATE USER admin FOR LOGIN admin;
CREATE ROLE db_admin;

GRANT EXECUTE ON GetOrderPrice TO db_admin;
GRANT SELECT, UPDATE, INSERT, DELETE ON Flights TO db_admin;
GRANT SELECT, UPDATE, INSERT, DELETE ON Orders TO db_admin;
GRANT SELECT, UPDATE, INSERT, DELETE ON OrdersTickets TO
    db_admin;
GRANT SELECT, UPDATE, INSERT, DELETE ON Planes TO db_admin;
GRANT SELECT, UPDATE, INSERT, DELETE ON PlanesFlights TO
    db_admin;
GRANT SELECT, UPDATE, INSERT, DELETE ON Services TO db_admin;
GRANT SELECT, UPDATE, INSERT, DELETE ON Tickets TO db_admin;
GRANT SELECT, UPDATE, INSERT, DELETE ON TicketsServices TO
    db_admin;
GRANT SELECT, UPDATE, INSERT, DELETE ON Users TO db_admin;

ALTER ROLE db_admin ADD MEMBER admin;

```

## 3.6 Реализация интерфейса доступа к базе данных

Для обеспечения взаимодействия приложения с базой данных были написаны интерфейсы, которые предоставляют моделям (сущностям) определенные методы для работы с данными. У каждой модели есть приватное поле, которое представляет собой экземпляр конкретного интерфейса.

В листингах 3.18–3.19 показаны соответственно интерфейс репозитория для сущности «Заказ» и реализация одного из его методов.

Листинг 3.18 – Интерфейс репозитория для сущности «Заказ»

```
public interface IOrderRepository<T>
    where T : class
{
    public IEnumerable<T> GetOrderByUserId(Int64 userId);
    public IEnumerable<OrderedTicket> GetTicketsByOrderId(Int64
        orderId);
    public T GetActiveOrderByUserId(Int64 userId);
    public void DeleteTicketFromOrder(Int64 ticketId);
    public decimal GetOrderPrice(Int64 orderId);
    public void InsertOrder(Int64 userId);
    public void UpdateOrder(T order);
    public void DeleteOrder(Int64 orderId);
}
```

Листинг 3.19 – Реализация одного из методов интерфейса репозитория для сущности «Заказ»

```
public IEnumerable<Order> GetOrderByUserId(Int64 userId)
{
    List<Order> orders = new List<Order>();

    using (SqlConnection connection = new
        SqlConnection(this._connectionString))
    {
        connection.Open();

        SqlCommand command = new SqlCommand($"SELECT * FROM
            [Orders] where [Orders].UserId = @UserId",
            connection);
        command.Parameters.AddWithValue("UserId", userId);

        var dataReader = command.ExecuteReader();
```

```

        orders = QueryHandler.GetList<Order>(dataReader);
    }

    return orders;
}
}

```

### 3.7 Тестирование разработанного функционала

Для тестирования программного обеспечения на разных этапах разработки использовались различные методы тестирования: mock-тесты для проверки корректности реализации бизнес-логики приложения, интеграционные тесты для проверки корректности работы приложения с базой данных и тесты по сценариям использования для проверки корректности работы реализованного веб-приложения.

В листинге 3.20 показана реализация одного из mock-тестов для сущности «Пользователь». Данный тест проверяет корректность работы метода регистрации нового пользователя.

Листинг 3.20 – Mock-тест для сущности «Пользователь»

```

[Fact]
public void RegisterNewUser()
{
    string email = "testEmail@test.com";
    string password = "testPassword123";
    var mock = new Mock<IUserRepository<User>>();

    mock.Setup(repo => repo.GetUserByEmail(email))
        .Returns(GetNonExistentUserByEmail(email));
    mock.Setup(repo => repo.InsertUser(new User()))
        .Callback(Insert);

    var user = new User(mock.Object);

    user.Register(email, password);

    Assert.Equal(email, user.Email);
    Assert.NotEqual(password, user.Password);
}

```

В листинге 3.21 показана реализация одного из интеграционных тестов для сущности «Пользователь». Данный тест проверяет корректность работы метода регистрации нового пользователя.

Листинг 3.21 – Интеграционный тест для сущности «Пользователь»

```
[Fact]
public void RegisterNewUser()
{
    string email = "testEmail@test.com";
    string password = "testPassword123";

    var user = new User();

    user.Register(email, password);

    Assert.Equal(email, user.Email);
    Assert.NotEqual(password, user.Password);
}
```

## Вывод из технологической части

В ходе выполнения технологической части курсовой работы были сформулированы требования к программному обеспечению; выбраны средства реализации; написан исходный код сущностей, ограничений целостности, проектируемой функции, ролевой модели базы данных; реализован интерфейс доступа к базе данных и протестирован разработанный функционал.

## **4 Исследовательская часть**

### **4.1 Технические характеристики устройства**

Технические характеристики устройства, на котором было проведено измерение времени выполнения запросов:

- 1) операционная система Windows 10 Pro x64;
- 2) оперативная память 16 ГБ;
- 3) процессор Intel® Core™ i7-4790K @ 4.00 ГГц.

Измерение времени выполнения запросов проводилось на стороне сервера с помощью класса Stopwatch, который предоставляет набор методов и свойств, которые можно использовать для точного измерения затраченного времени [9].

### **4.2 Исследование характеристик разработанного программного обеспечения**

Индекс — это структура, которая определяет соответствие значения ключа записи (атрибута или группы атрибутов) и местоположения этой записи. Каждый индекс связан с определенной таблицей, но является внешним по отношению к таблице. Индекс обычно хранится в отдельном файле или отдельной области памяти. Пустые значения атрибутов (NULL) не индексируются. Индексирование используется для ускорения доступа к записям по значению ключа и не влияет на размещение данных этой таблицы. Ускорение поиска данных через индекс обеспечивается за счет упорядочивания значений индексируемого атрибута (что позволяет просматривать в среднем половину индекса при линейном поиске) и за счет того, что индекс занимает меньше страниц памяти, чем сама таблица, поэтому система тратит меньше времени на чтение индекса, чем на чтение таблицы [10].

Таблица или представление может иметь индексы кластеризованные и некластеризованные. Кластеризованные индексы сортируют и хранят строки данных в таблицах или представлениях на основе их ключевых значений. Этими значениями являются столбцы, включенные в определение индекса.



Существует только один кластеризованный индекс для каждой таблицы, так как строки данных могут храниться в единственном порядке. Строки данных в таблице хранятся в порядке сортировки только в том случае, если таблица содержит кластеризованный индекс. Некластеризованные индексы имеют структуру, отдельную от строк данных. В некластеризованном индексе содержатся значения ключа некластеризованного индекса, и каждая запись значения ключа содержит указатель на строку данных, содержащую значение ключа. В Microsoft SQL Server индексы создаются автоматически при определении ограничений PRIMARY KEY или UNIQUE на основе столбцов таблицы [11].

Во всех таблицах базы данных, которую использует реализованное в ходе выполнения курсовой работы приложение, есть ограничение PRIMARY KEY, соответственно, Microsoft SQL Server автоматически создает кластеризованные индексы. В таблице «Пользователи» есть уникальный атрибут «почта», для которого СУБД создает некластеризованный индекс. Имеет смысл вручную создать некластеризованный составной индекс по атрибутам «пункт вылета», «пункт прибытия» и «дата вылета» для таблицы «Рейсы» для увеличения скорости поиска рейсов.

Кроме того, можно вручную создать кластеризованный индекс по атрибуту «ID пользователя» в таблице «Заказы» и измерить время выполнения запроса поиска заказа по идентификатору пользователя без индекса и с ним.

### 4.3 Время выполнения запросов

В листинге 4.1 приведена команда создания кластеризованного индекса для таблицы «Заказы».

Листинг 4.1 – Создание кластеризованного индекса

```
CREATE CLUSTERED INDEX user_id_index on Orders (UserId);
```

На рисунках 4.1–4.2 показана зависимость времени выполнения запроса (в миллисекундах) от количества строк в таблице.

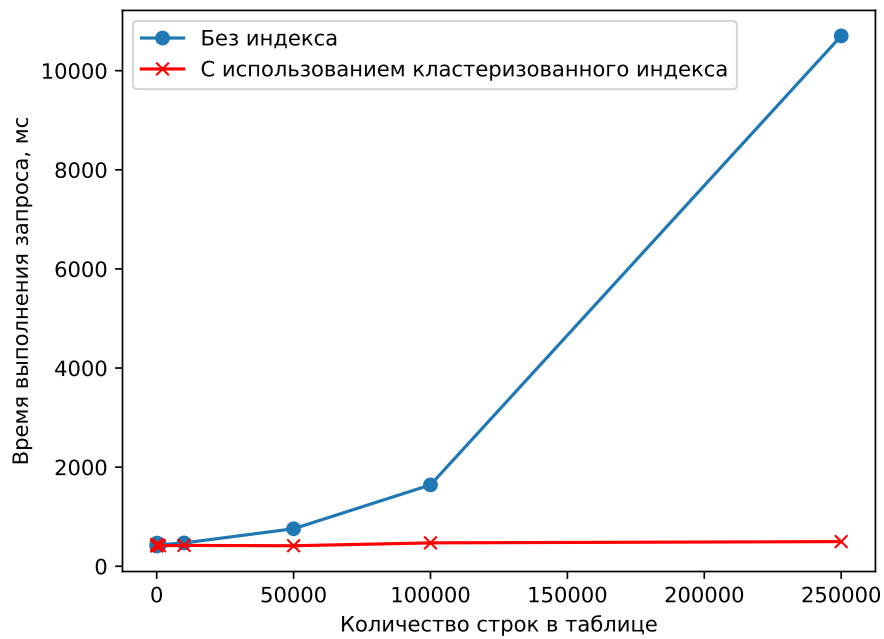


Рисунок 4.1 – Зависимость времени выполнения запроса от количества строк в таблице при поиске первой записи

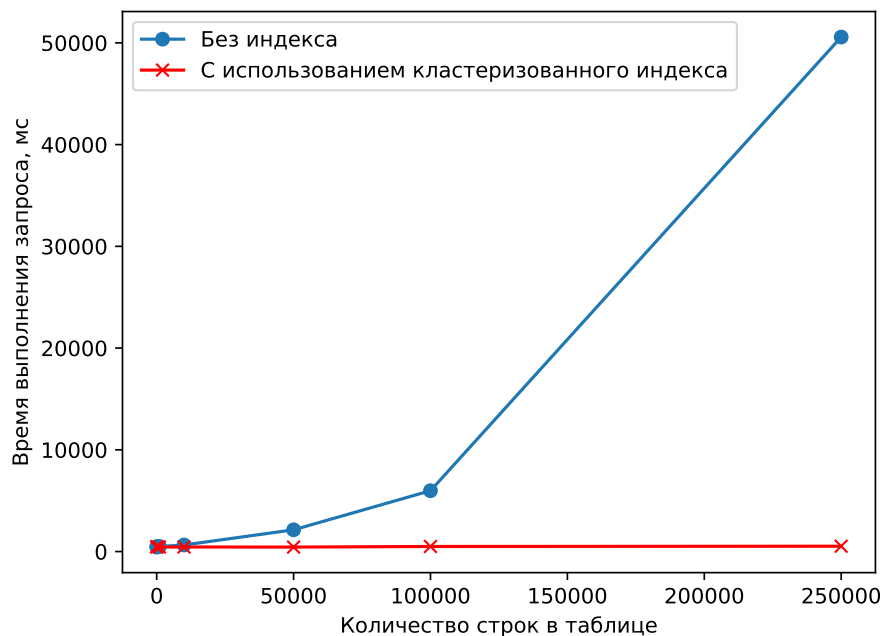


Рисунок 4.2 – Зависимость времени выполнения запроса от количества строк в таблице при поиске последней записи

## Вывод из исследовательской части

В ходе выполнения исследовательской части курсовой работы был проведен краткий обзор кластеризованных и некластеризованных индексов; определены таблицы и атрибуты, для которых имеет смысл создавать индексы,

и проведен эксперимент по измерению времени выполнения запроса без использования кластеризованного индекса и с ним. Согласно полученным при проведении эксперимента данным, при количестве строк в таблице до 10 тысяч существенной разницы во времени выполнения запроса не наблюдается. При большем количестве записей время выполнения запроса без использования индекса растет в разы быстрее, в то время как время выполнения запроса с использованием индекса остается практически неизменным. При 250 тысячах строк в таблице использование индекса увеличивает скорость выполнения запроса в 21 раз при поиске первой записи и в 97 раз при поиске последней.

## ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была разработана база данных для хранения и обработки данных авиакомпании, а также веб-приложение, которое ее использует.

Были выполнены следующие задачи:

- 1) проведен обзор существующих веб-приложений для покупки авиабилетов и сформулированы требования и ограничения к разрабатываемой базе данных и приложению;
- 2) спроектирована архитектура базы данных, ограничения целостности и ролевая модель на уровне базы данных;
- 3) выбраны средства реализации и реализованы спроектированная база данных и необходимый интерфейс для взаимодействия с ней;
- 4) исследованы характеристики разработанного программного обеспечения.

В ходе проведения эксперимента при выполнении исследовательской части курсовой работы было установлено, что при 250 тысячах строк в таблице использование кластеризованного индекса увеличивает скорость выполнения запроса в 21 раз при поиске первой записи и в 97 раз при поиске последней.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Сьоре Э.* Проектирование и реализация систем управления базами данных // М.: ДМК Пресс. — 2020. — С. 476.
2. *Airport A.* Why Plane Is The Best Transportation Vehicle. — 2021. — (Дата обращения: 20.03.2023). <https://ataturkairport.com/why-plane-is-the-best-transportation-vehicle/>.
3. *Аврунев О. Е., Стасышин В. М.* Модели баз данных // Новосибирский государственный технический университет. — 2018. — С. 124.
4. *Виноградов В. И., Виноградова М. В.* Постреляционные модели баз данных и языки запросов // Московский государственный технический университет имени Н. Э. Баумана. — 2016. — С. 100.
5. *Бекасов Д. Е.* Конспект лекций по проектированию программного обеспечения // Московский государственный технический университет им. Н. Э. Баумана. — 2023.
6. *Taylor P.* Ranking of the most popular database management systems worldwide, as of February 2023. — 2023. — (Дата обращения: 10.05.2023). <https://www.statista.com/statistics/809750/worldwide-popularity-ranking-database-management-systems/>.
7. *Ranking D.-E.* Knowledge Base of Relational and NoSQL Database Management Systems. — 2023. — (Дата обращения: 10.05.2023). <https://db-engines.com/en/ranking>.
8. *Microsoft.* COALESCE (Transact-SQL). — 2023. — (Дата обращения: 13.05.2023). <https://learn.microsoft.com/ru-ru/sql/t-sql/language-elements/coalesce-transact-sql?view=sql-server-ver16>.
9. *Microsoft.* Документация по .NET. — 2023. — (Дата обращения: 15.05.2023). <https://learn.microsoft.com/ru-ru/dotnet/api/system.diagnostics.stopwatch?view=net-7.0>.
10. *Карпова И. П.* Базы данных. Учебное пособие // М. — 2009. — С. 131.

11. *Microsoft*. Описание кластеризованных и некластеризованных индексов. — 2023. — (Дата обращения: 15.05.2023). <https://learn.microsoft.com/ru-ru/sql/relational-databases/indexes/clustered-and-nonclustered-indexes-described?view=sql-server-ver16>.

# ПРИЛОЖЕНИЕ А

## Презентация