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 _____ ИУ7-63Б
(Группа) (Подпись, дата) (И. О. Фамилия)

Руководитель курсовой работы _____ В. Марченко
(Подпись, дата) (И. О. Фамилия)

Консультант _____ Д. А. Шибанова
(Подпись, дата) (И. О. Фамилия)

2023 г.

РЕФЕРАТ

Расчетно-пояснительная записка 53 с., 12 рис., 9 табл., 18 источн., 1 прил.

АВИАБИЛЕТЫ, АВИАПЕРЕЛЕТЫ, БАЗЫ ДАННЫХ, ВЕБ-ПРИЛОЖЕНИЕ, РЕЛЯЦИОННАЯ МОДЕЛЬ ДАННЫХ, SQL

Объектом разработки является база данных и приложение к ней.

Объектом исследования являются кластеризованные и некластеризованные индексы.

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

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

В ходе выполнения конструкторской части были выделены шесть сущностей: пользователь, заказ, рейс, билет, самолет и услуга. А также три роли на уровне базы данных: клиент, модератор и администратор.

При выполнении технологической части были выбраны средства реализации программного обеспечения (язык программирования C# и система управления базами данных Microsoft SQL Server), реализован интерфейс доступа к базе данных и проведено тестирование разработанного функционала (mock-тесты, интеграционные тесты и тесты по сценариям использования).

В ходе проведения исследования было установлено, что при 250 тысячах строк в таблице использование и кластеризованного, и некластеризованного индекса увеличивает скорость выполнения запроса в 21 раз при поиске первой записи и в 97 раз при поиске последней на примере одной из таблиц спроектированной базы данных.

Область применения результатов — дальнейшее развитие и расширение приложения для поиска и покупки билетов на любые виды транспорта.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ **7**

1 Аналитическая часть **8**

1.1 Анализ предметной области 8

1.2 Требования к базе данных и приложению 10

1.3 Модели баз данных 10

1.3.1 Дореляционные базы данных 10

1.3.2 Реляционные базы данных 14

1.3.3 Постреляционные базы данных 15

1.4 Информация, подлежащая хранению в базе данных 17

1.5 ER-диаграмма сущностей базы данных 19

1.6 Пользователи приложения 20

1.7 Диаграмма вариантов использования 21

2 Конструкторская часть **24**

2.1 Описание сущностей базы данных 24

2.2 Описание ограничений целостности базы данных 27

2.3 Описание проектируемой функции на уровне базы данных . . . 29

2.4 Описание ролевой модели на уровне базы данных 30

3 Технологическая часть **32**

3.1 Средства реализации 32

3.2 Реализация сущностей базы данных 35

3.3 Реализация ограничений целостности базы данных 37

3.4 Реализация функции на уровне базы данных 40

3.5 Реализация ролевой модели на уровне базы данных 41

3.6 Реализация интерфейса доступа к базе данных 42

3.7 Тестирование разработанного функционала 43

4 Исследовательская часть **45**

4.1 Технические характеристики устройства 45

4.2 Исследование характеристик разработанного программного обеспечения 45

4.3 Время выполнения запроса 46

ЗАКЛЮЧЕНИЕ **50**

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ **51**

ПРИЛОЖЕНИЕ А Презентация **53**

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей расчетно-пояснительной записке к курсовой работе применяют 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
MPA	Многостраничное приложение (Multi Page Application)
SPA	Одностраничное приложение (Single Page Application)
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:.

1) самый быстрый путь к месту назначения;

2) авиабилеты являются доступными для среднестатистического человека;

3) полет — один из самых безопасных способов путешествовать [2].

Целью курсовой работы является разработка базы данных для хранения и обработки данных авиакомании, 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 [3]. Это метапоисковик билетов, и сама компания перелеты не организует. Так как целью курсовой работы является разработка базы данных для хранения и обработки данных авиакомпании, стоит рассмотреть сервисы, которые предоставляют возможность покупки билетов на самолеты, принадлежащие тем или иным организациям.

Из довольно известных международных компаний, у которых есть собственный флот, можно выделить следующие: Qatar Airways [4], Emirates [5] и United Airlines [6].

Можно сразу выделить тот факт, что в приложении United Airlines есть возможность переключения пользовательского интерфейса на 8 языков, в то время как Qatar Airways и Emirates предлагают на выбор свыше 70 языков. На домашней странице всех приложений пользователю сразу предлагается выбрать пункт вылета и прибытия, тип билета — в одну сторону или «туда и обратно», даты вылета и обратного рейса, количество билетов и их классы. Emirates и United Airlines помимо экономкласса, бизнес-класса и первого класса предлагают билеты премиального экономкласса. Кроме того, во всех приложениях доступны такие функции как онлайн регистрация, управление бронированием и проверка статуса рейса. Еще одно достоинство трех приложений — нет необходимости регистрироваться для поиска билетов. Регистрация нужна только в случае, если нужно купить билет.

С другой стороны, все три приложения обладают большим недостатком. Они слишком «перегружены» различного рода информацией для пользователя. Например, Qatar Airways предлагает купить пакет, в который входят билеты, бронирование отеля и билеты на Формулу 1. Другие приложения предлагают забронировать множество отелей в различных городах мира. Пользователя, который хочет быстро приобрести билет на интересующий его рейс, вся эта информация будет отвлекать. Поэтому одна из задач данной курсовой работы — создание простого, интуитивно понятного приложения для поиска авиабилетов. На рисунках 1.1–1.3 показаны графические интерфейсы пользователя трех рассматриваемых приложений для поиска авиабилетов.

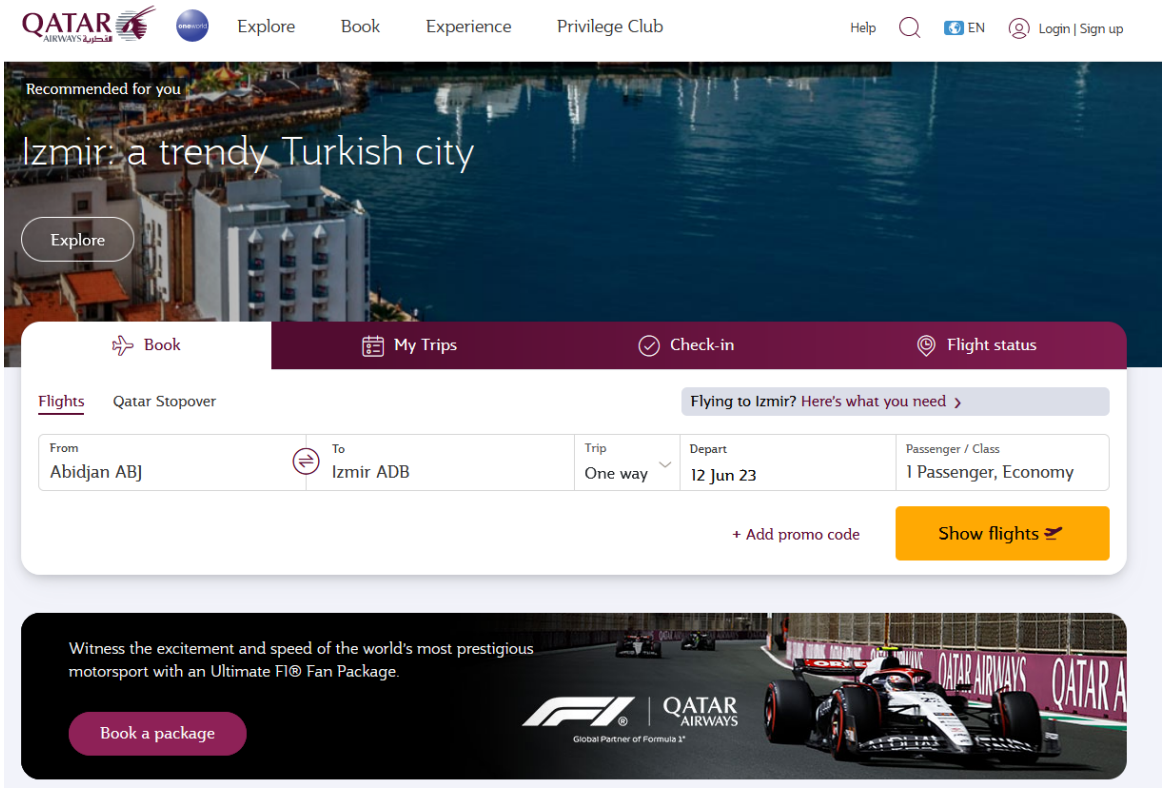

Рисунок 1.1 – Графический интерфейс пользователя приложения Qatar Airways

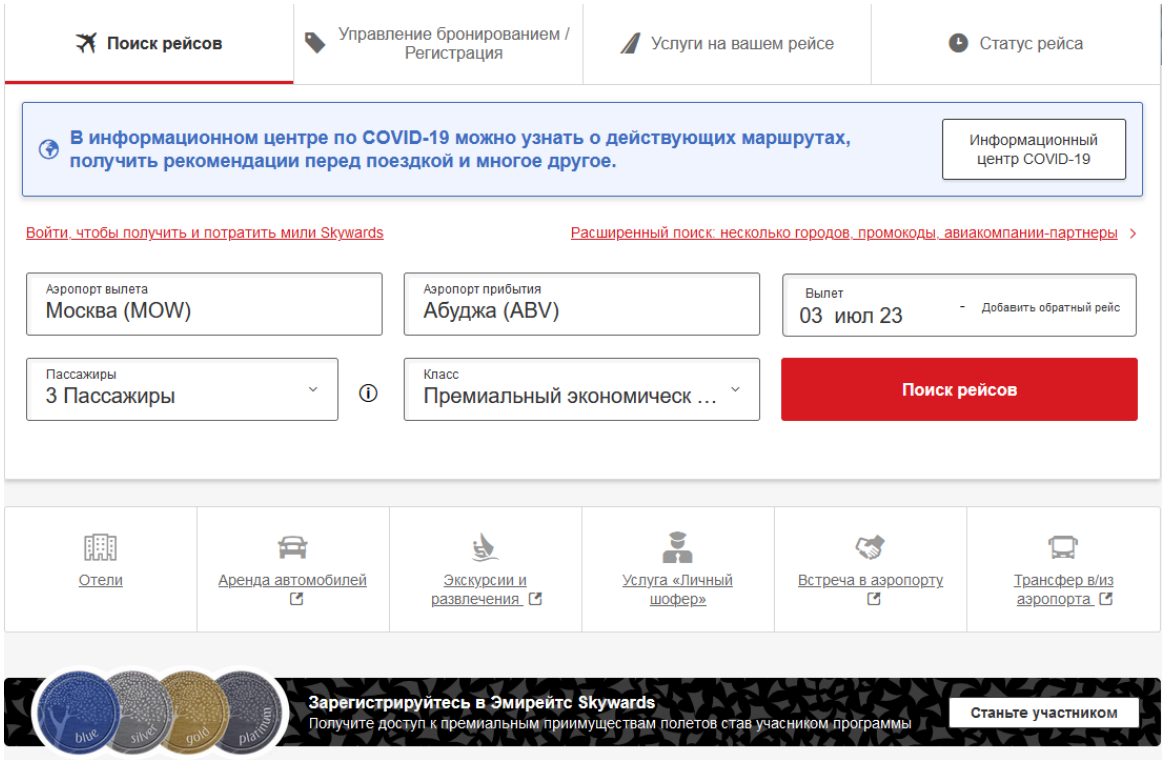

Рисунок 1.2 – Графический интерфейс пользователя приложения Emirates

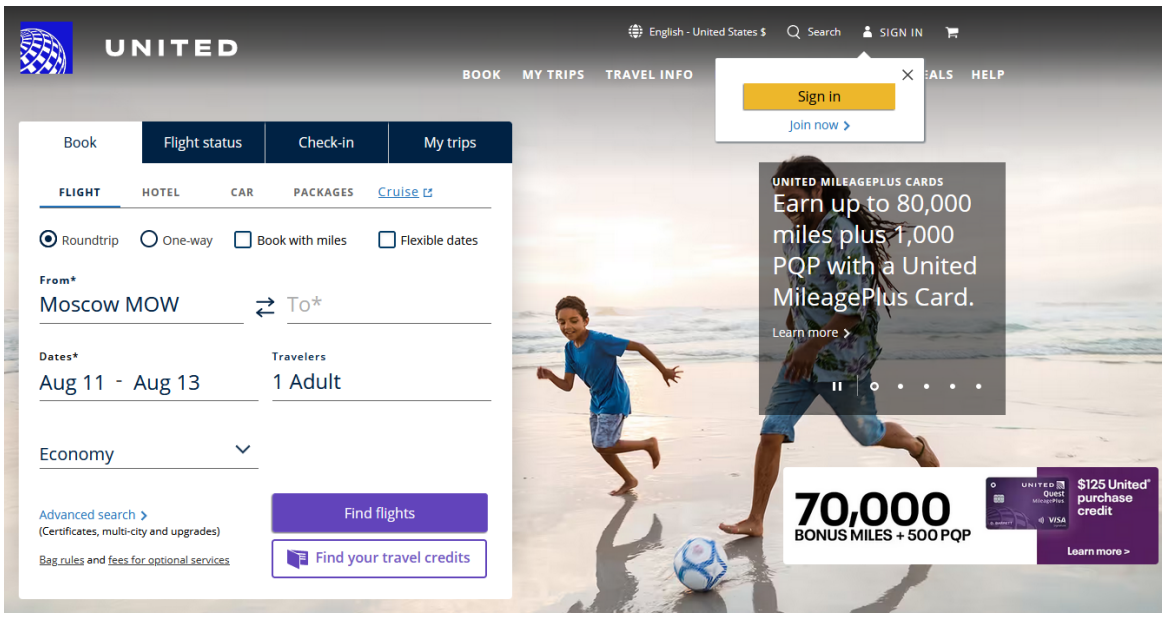

Рисунок 1.3 – Графический интерфейс пользователя приложения United Airlines

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 именений в личные данные и возможность просмотреть информацию о самолетах авиакомании.

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 [7].

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 [7].

Иерархическая модель данных — представление базы данных в виде древовидной (иерархической) структуры, состоящей из объектов (данных) различных уровней [7].

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 [7].

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 [7].

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 [7].

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 [7].

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 [7].

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 [7].

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 [7].

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) [7].

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) [7].

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 [7].

1.3.2 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) [8].

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 [8].

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 арности, 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 [7].

Достоинства реляционной модели данных:

1) гарантия целостности данных при их обновлении;

2) низкий риск потери информации;

3) соответствие требованиям к транзакционным системам (ACID);

4) простое представление и формирование базы данных;

5) универсальность и удобство обработки данных, которая осуществляется с помощью декларативного языка запросов SQL (Structured Query Language) [9].

Недостатки реляционной модели данных:

1) отсутствие специальных средств для отображения различных типов связей и агрегатов;

2) при нормализации отношений данные об одной сущности предметной области распределяются по нескольким таблицам, что усложняет работу с БД [9];

3) отсутствие специальных механизмов навигации [10].

1.3.3 Постреляционные базы данных

Третье поколение баз данных, называемых постреляционные, начало развиваться с 90-годов XX века. Тогда появились объектные, объектно-реляционные и полуструктрированные 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 данные развиваются параллельно на основе сетевых и иерархических баз данных и позволяют работать с частично структурированными данными [8].

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 [8].

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 [8].

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 [8].

Поддерживает следующие типы данных:

1) атомарные;

2) структуры, содержащие набор именованных полей;

3) массивы, хранящие упорядоченный набор элементов;

4) множества и мультиможества, хранящие неупорядоченный набор элементов;

5) пользовательские типы данных;

6) ссылки на объекты (кортежи) пользовательских типов данных [8].

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 [8].

Полуструктрированные 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 вебсайтов);

3) для работы с унаследованными БД при укрупнении, поглощении компаний (описывают обобщенную структуру исходных БД) [8].

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, имя, фамилия и дата регистрации.

2. Заказ. Информация о заказах зарегистрированных пользователей: заказчик билета и статус заказа.

3. Рейс. Информация о рейсах, на которые продаются билеты. Пункт вылета, пункт прибытия, дата и время вылета, дата и время прибытия.

4. Билет. Информация о продаваемых билетах. Рейс, на который продается билет, заказ, в котором находится текущий билет, номер ряда, место, класс, возможность возврата и стоимость.

5. Самолет. Информация о самолетах, которые выполняют рейсы. Название комании-производителя, модель, количество мест экономкласса, бизнес-класса и первого класса.

6. Услуга. Информация о предоставляемых услугах. Наименование, стоимость, возможность получения данной услуги для билета эконом класса, бизнес-класса и первого класса.

Рассмотрим несколько факторов, исходя из которых будет выбрана модель данных:

1) выделенные в текущем разделе шесть сущностей представляют собой структурированные данные, структура которых не подвержена частым изменениям;

2) важно соблюдать целостность и структуризацию хранимых данных;

3) разрабатываемое в ходе выполнения данной курсовой работы программное обеспечение предназначено для покупки авиабилетов, поэтому выбираемая модель данных должна соответствотвать требованиям к транзакционным системам (ACID);

4) в рамках проектируемой базы данных отсутствует необходимость хранения коллекций.

Исходя из перечисленных причин, подходящей моделью данных является реляционная.

1.5 ER-диаграмма сущностей базы данных

На рисунке 1.4 показана диаграмма «сущность-связь» проектируемой базы данных в нотации Чена. Представлены шесть сущностей и их свойства.

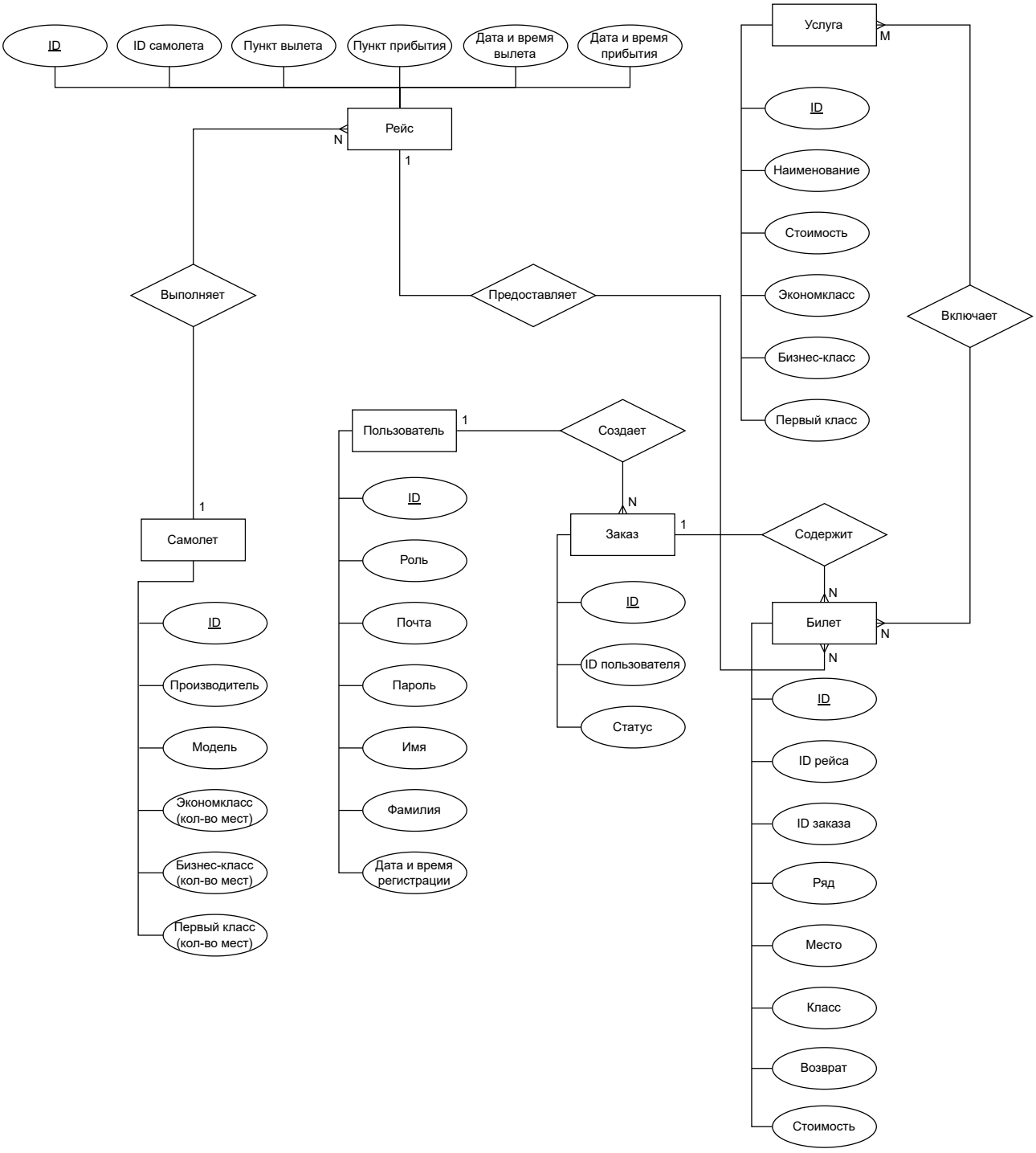

Рисунок 1.4 – ER-диаграмма

1.6 Пользователи приложения

Пользователи приложения могуть 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 создать и просмотреть, добавить и удалить билеты, добавить и удалить услуги, очистить или оплатить заказ. Также у клиента 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 или удалять их оттуда.

Администратор может удалять учетные записи пользователей через приложение, а также имеет полный доступ к базе данных и, соответственно, может создавать новые таблицы или удалять старые. Администратор может напрямую работать с базой данных без приложения.

1.7 Диаграмма вариантов использования

На рисунках 1.5–1.8 показаны use case диаграммы для разных пользователей.

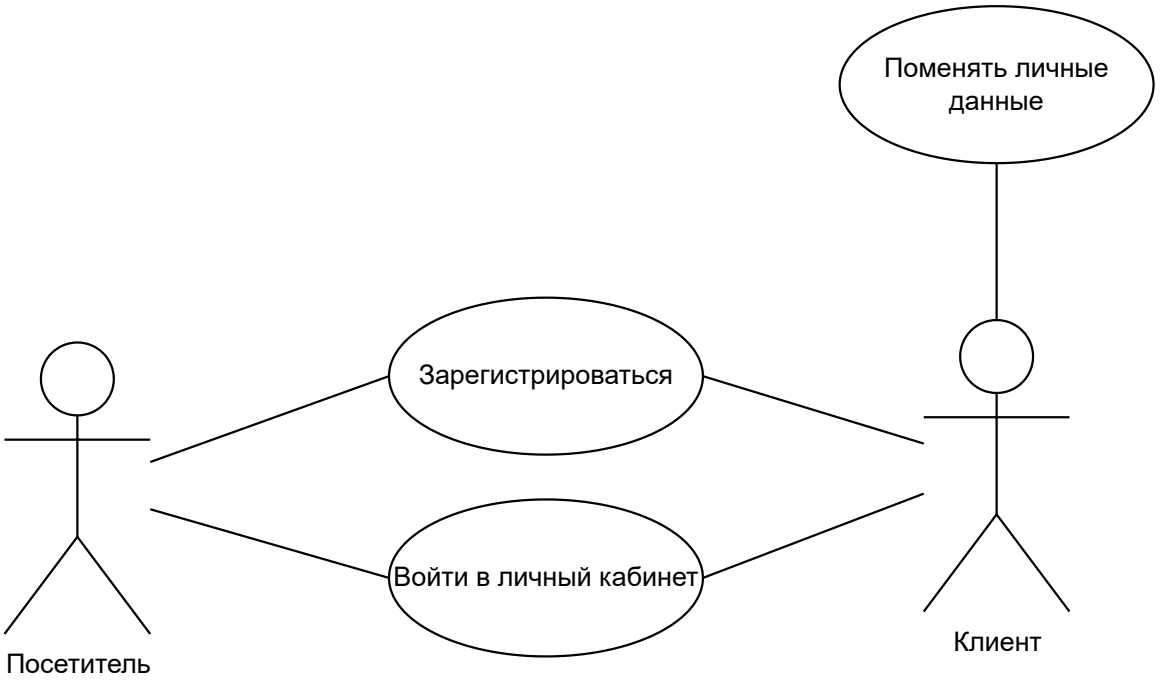

Рисунок 1.5 – Взаимодействие пользователя с личными данными

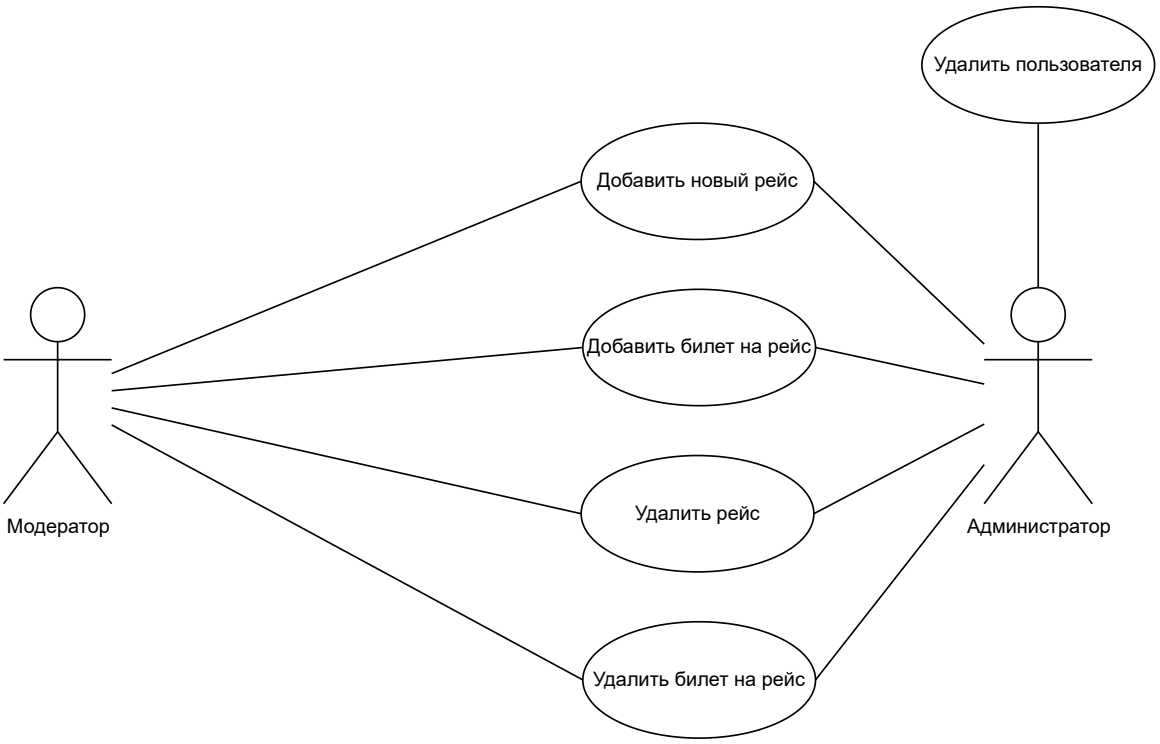

Рисунок 1.6 – Администрирование

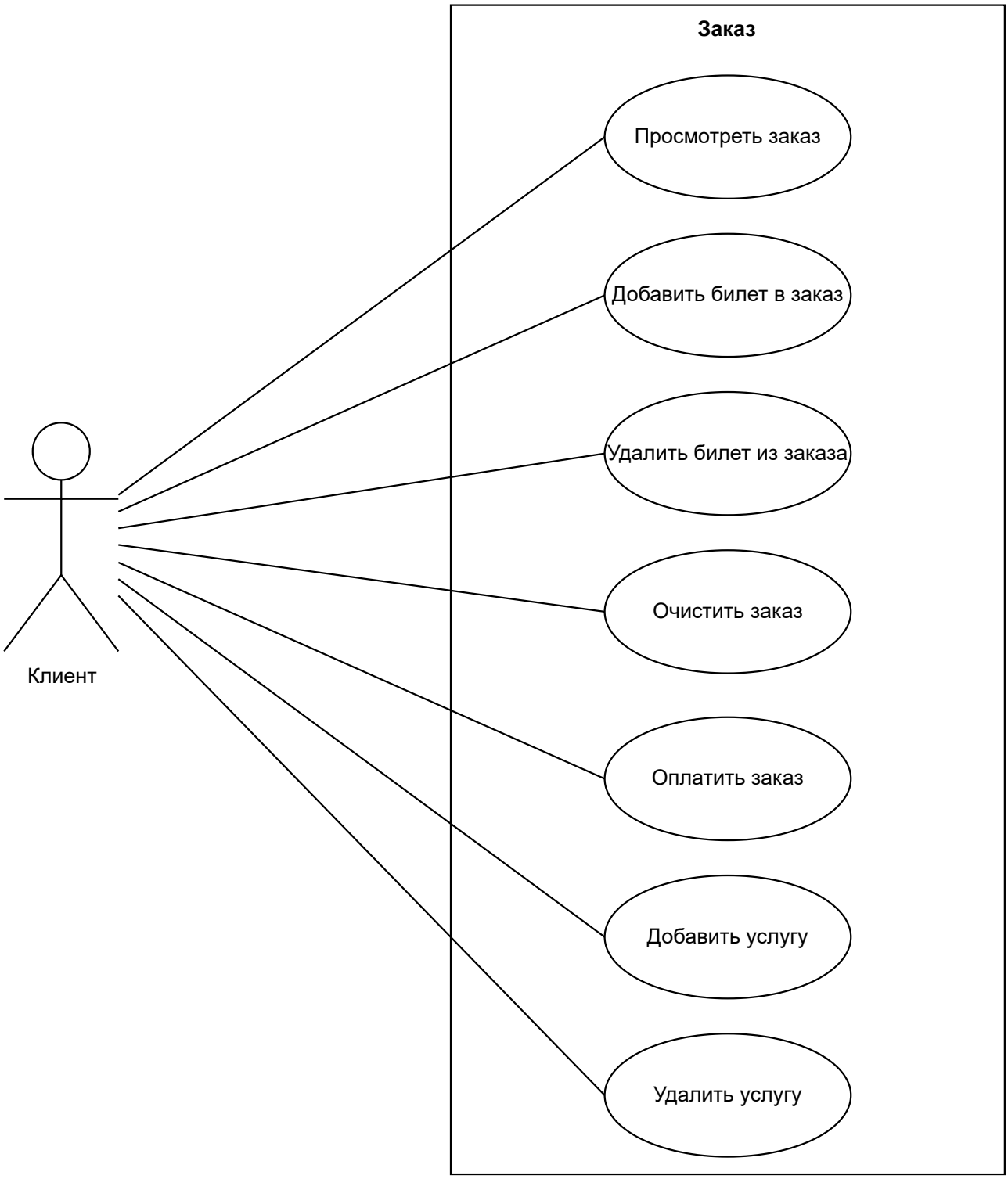

Рисунок 1.7 – Взаимодействие пользователя с заказом

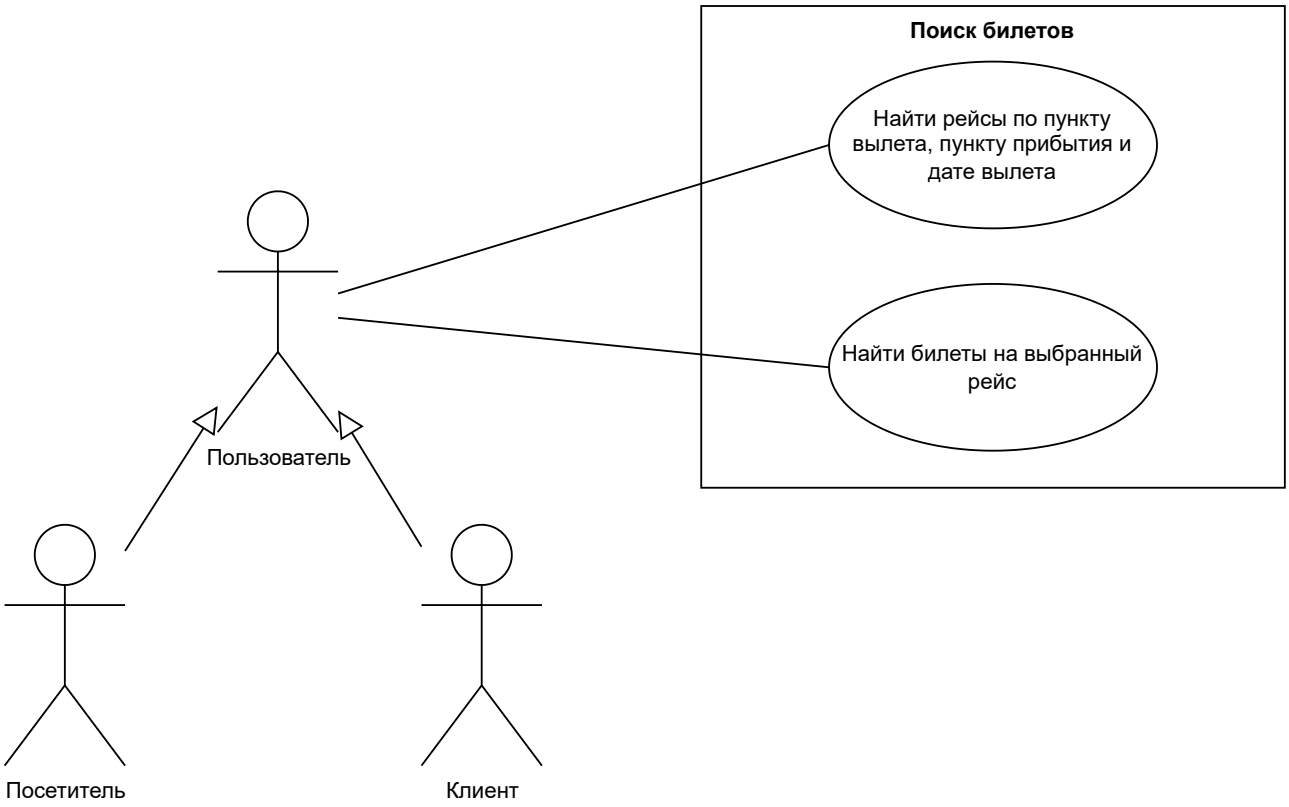

Рисунок 1.8 – Поиск рейсов и билетов

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.

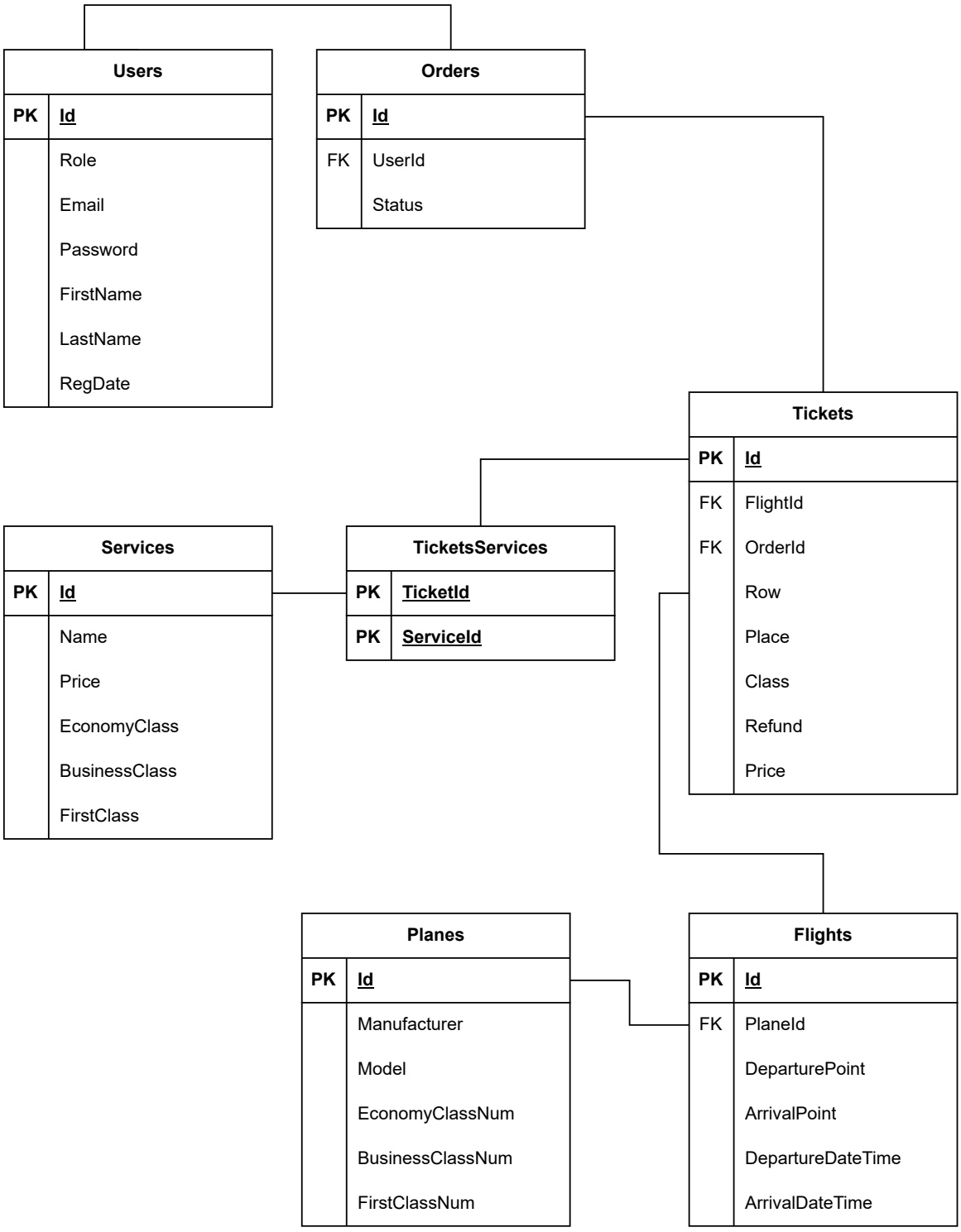

Рисунок 2.1 – Диаграмма базы данных

В таблицах 2.1–2.7 описаны атрибуты таблиц базы данных.

Таблица 2.1 – Атрибуты таблицы «Пользователи»

Имя атрибута	**Тип атрибута**	**Описание**
Id	целое	Идентификатор — первичный ключ
Role	строка	Роль пользователя
Email	строка	Адрес электронной почты
Password	строка	Зашифрованный пароль
FirstName	строка	Имя пользователя
LastName	строка	Фамилия пользователя
RegDate	дата и время	Дата регистрации

Таблица 2.2 – Атрибуты таблицы «Заказы»

Имя атрибута	**Тип атрибута**	**Описание**
Id	целое	Идентификатор — первичный ключ
UserId	целое	Идентификатор пользователя — внешний ключ
Status	строка	Статус заказа

Таблица 2.3 – Атрибуты таблицы-связки «Билеты-Услуги»

Имя атрибута	**Тип атрибута**	**Описание**
TicketId	целое	Идентификатор билета — первичный и внешний ключ
ServiceId	целое	Идентификатор услуги — первичный и внешний ключ

Таблица 2.4 – Атрибуты таблицы «Билеты»

Имя атрибута	**Тип атрибута**	**Описание**
Id	целое	Идентификатор — первичный ключ
FlightId	целое	Идентификатор рейса — внешний ключ
OrderId	целое	Идентификатор заказа
Row	целое	Ряд в самолете
Place	символ	Место в ряду
Class	строка	Класс билета
Refund	логический тип	Возможен ли возврат билета
Price	денежный тип	Стоимость

Таблица 2.5 – Атрибуты таблицы «Рейсы»

Имя атрибута	**Тип атрибута**	**Описание**
Id	целое	Идентификатор — первичный ключ
PlaneId	целое	Идентификатор самолета — внешний ключ
DeparturePoint	строка	Пункт вылета
ArrivalPoint	строка	Пункт прибытия
DepartureDateTime	дата и время	Дата и время вылета
ArrivalDateTime	дата и время	Дата и время прибытия

Таблица 2.6 – Атрибуты таблицы «Самолеты»

Имя атрибута	**Тип атрибута**	**Описание**
Id	целое	Идентификатор — первичный ключ
Manufacturer	строка	Название компании-производителя
Model	строка	Название модели самолета
EconomyClassNum	целое	Количество мест экономкласса
BusinessClassNum	целое	Количество мест бизнес-класса
FirstClassNum	целое	Количество мест первого класса

Таблица 2.7 – Атрибуты таблицы «Услуги»

Имя атрибута	**Тип атрибута**	**Описание**
Id	целое	Идентификатор — первичный ключ
Name	строка	Название услуги
Price	денежный тип	Стоимость услуги
EconomyClass	логический тип	Доступна ли услуга для билета экономкласса
BusinessClass	логический тип	Доступна ли услуга для билета бизнес-класса
FirstClass	логический тип	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.

Ограничения для таблицы «Пользователи». Идентификатор — уникальное положительное число, первичный ключ. Роль пользователя не может отсутствовать. Адрес электронной почты не может отсутствовать и должен быть уникальным. Пароль не может отсутствовать. Дата регистрации пользователя не должна отсутствовать.

Ограничения для таблицы «Заказы». Идентификатор — уникальное положительное число, первичный ключ. Идентификатор пользователя — положительное число, внешний ключ, не должен отсутствовать. Статус не должен отсутствовать.

Ограничения для таблицы «Рейсы». Идентификатор — уникальное положительное число, первичный ключ. Идентификатор самолета — положительное число, внешний ключ, не должен отсутствовать. Пункты вылета и прибытия не должны отсутствовать. Дата и время вылета и прибытия не должны отсутствовать, притом дата и время вылета должны быть меньше даты и время прибытия.

Ограничения для таблицы «Билеты». Идентификатор — уникальное положительное число, первичный ключ. Идентификатор рейса — положительное число, внешний ключ, не должен отсутствовать. Ряд в самолете — положительное число, не должен отсутствовать. Место в ряду не должно отсутствовать. Класс не должен отсутствовать. Возможность возврата не должен отсутствовать. Стоимость — положительное число, не должна отсутствовать.

Ограничения для таблицы «Самолеты». Идентификатор — уникальное положительное число, первичный ключ. Производитель и модель не должны отсутствовать. Количество мест разных классов — неотрицательные числа, не должны отсутствовать.

Ограничения для таблицы «Услуги». Идентификатор — уникальное положительное число, первичный ключ. Наименование услуги не должно отсутствовать. Стоимость — положительное число, не должна отсутствовать. Доступность для мест разных классов не должна отсутствовать.

Ограничения для таблицы-связки «Билеты-Услуги». Идентификатор билета — положительное число, первичный и внешний ключ. Идентификатор услуги — положительное число, первичный и внешний ключ.

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.

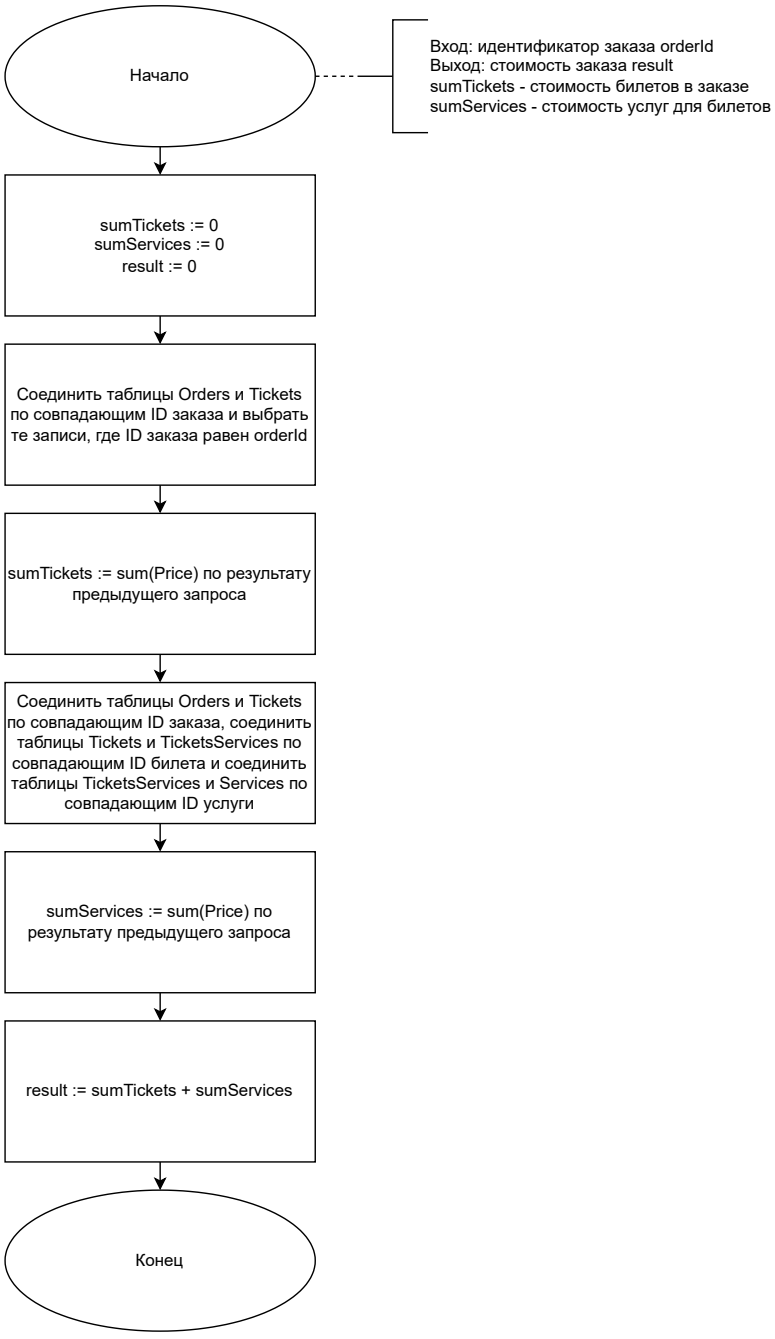

Рисунок 2.2 – Функция, возвращающая стоимость заказа

В функции используются следующие переменные:

1) orderId — идентификатор заказа (входной параметр);

2) result — стоимость заказа (выходной параметр);

3) sumTickets — стоимость билетов, входящих в заказ;

4) sumServices — стоимость услуг для билетов, входящих в заказ.

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. Взаимодействией 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 зазать билет, пользователь будет переадресован на страницу регистрации. Для трех других типов пользователей нужно создавать роли.

Доступны три роли на уровне базы данных.

1. Клиент взаимодействует с базой данных через приложение и получает информацию о рейсах, билетах, самолетах, услугах и своих заказах. Также клиент может изменить информацию о себе, которая хранится в таблице «Пользователи».

2. Модератор обладает всеми правами клиента. Он тоже взаимодействует с базой данных через приложение и может добавлять информацию о билетах и рейсах или удалять ее.

3. Администратор обладает всеми правами модератора. Он взаимодействует с базой данных и через приложение, и напрямую. Администратор может добавлять, удалять или изменять любую информацию, которая хранится в базе данных.

Вывод из конструкторской части

В ходе выполнения конструкторской части куросовой 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 используются для разных целей и имеют свои преимущества и недостатки.

Одностраничные приложения обладают следующими преимуществами:

1) SPA загружают необходимые компоненты только один раз, а последующие обновления страницы выполняются динамически, не требуя полной перезагрузки страницы, что приводит к ускорению загрузки страниц и более плавному взаимодействию с пользователем;

2) SPA легче поддерживать, поскольку они используют модульную конструкцию, где каждый компонент отвечает за свою функциональность, что упрощает изменение или замену компонентов, не затрагивая другие части приложения;

3) запросы к серверу сведены к минимуму, поскольку приложение использует программный интерфейс (API) для извлечения данных с сервера, а не требует, чтобы сервер создавал целую страницу для каждого запроса;

4) когда пользователь находится в автономном режиме, SPA может отображать кэшированную версию веб-приложения, чтобы пользователь мог продолжать использовать приложение даже при отсутствии подключения к Интернету;

5) клиентская и серверная части приложения разделены, что позволяет разработчикам изменять одно, не затрагивая другое [11].

Несмотря на свои преимущества, одностраничные приложения имеют несколько существенных недостатков:

1) поскольку SPA загружают весь свой контент одновременно, первоначальная загрузка может занять больше времени, что может быть проблемой для пользователей с медленным подключением к Интернету или старых устройств;

2) SPA трудно оптимизировать для поисковых систем, потому что они обычно имеют только один URL-адрес и ограниченный контент при начальной загрузке страницы, что влияет на их рейтинг в поисковых системах;

3) SPA могут быть несовместимы со старыми браузерами или устройствами, что может ограничить потенциальную базу пользователей;

4) поскольку большая часть логики приложения выполняется на стороне клиента, оно может быть более уязвимо для определенных типов атак, таких как межсайтовый скриптинг или подделка межсайтовых запросов [11].

Преимущества многостраничных приложений:

1) MPA, как правило, хорошо ранжируются в результатах поисковых систем, поскольку каждая страница имеет уникальный URL-адрес и может быть проиндексирована отдельно, что означает, что каждая страница в MPA может занимать независимое место в результатах поиска, потенциально увеличивая трафик на сайт;

2) поскольку MPA отправляют отдельные запросы для каждой страницы, проще реализовать меры безопасности, такие как аутентификация и авторизация;

3) в то время как SPA обеспечивают более высокую производительность после загрузки начальной страницы, начальное время загрузки MPA часто быстрее, поскольку браузеру нужно загрузить только содержимое текущей страницы, а не все содержимое всего приложения;

4) MPA — это классический способ создания веб-страниц, поэтому он совместим с большинством старых браузеров и устаревших систем [11].

Основные недостатки многостраничных приложений:

1) поскольку MPA требуют полной перезагрузки страницы, когда пользователь взаимодействует с приложением, они обеспечивают низкую производительность, что негативно влияет на взаимодействие с пользователем;

2) MPA требуют более сложной логики на стороне сервера по сравнению с SPA, поскольку каждая страница должна проектироваться и разрабатываться отдельно, что может привести к необходимости поддерживать больше кода и повысить риск возникновения ошибок;

3) MPA требуют, чтобы сервер обрабатывал больше запросов, поскольку для загрузки каждой страницы требуется отдельный запрос, что может привести к увеличению нагрузки на сервер и увеличению времени отклика, особенно для приложений с большим количеством пользователей [11].

Многостраничные приложения обычно выбираются для создания вебсайтов с большим количеством контента, такие как новостные сайты или сайты электронной коммерции, где каждая страница представляет собой отдельный фрагмент контента или продукта [11][12]. Так как цель курсовой работы — реализация приложения для поиска авиабилетов, был выбран подход с созданием многостраничных приложений.

Для реализации программного обеспечения был выбран язык программирования C# ввиду следующих причин:

1) на языке программирования C# можно создавать многостраничные вебприложения с помощью кроссплатформенного фреймворка ASP.NET Core;

2) фреймворк ASP.NET Identity представляет встроенную систему аутентификации и авторизации;

3) в стандартной библиотеке C# реализованы необходимые структуры данных для работы с БД (такие как IEnumerable<T>, List<T> и SortedSet<T>);

4) ASP.NET MVC Framework реализует шаблон Model-View-Controller, который предназначен для отделения бизнес-логики приложения от визуализации данных, что позволяет проводить тестирование одних компонентов приложения независимо от других.

Помимо языка программирования нужно выбрать систему управления базами данных. Наиболее популярные реляционные системы управления базами данных: Oracle, MySQL, Microsoft SQL Server, PostgreSQL и SQLite [13][14].

Для реализации программного продукта была выбрана СУБД Microsoft SQL Server ввиду следующих причин:

1) она подходит для многопользовательских приложений;

2) может работать без использования контейнеризации;

3) язык программирования C# предоставляет необходимый набор инструментов для работы с этой СУБД.

Таким образом, с помощью языка C# и СУБД Microsoft SQL Server можно реализовать программное обеспечение, которое соответствует перечисленным в аналитическом разделе 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 Int64 PlaneId { 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 Int64 OrderId { 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 BusinessClassNum { get; set; }
    public uint FirstClassNum { 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 BusinessClass { get; set; }
    public bool FirstClass { get; set; }
}
```

3.3 Реализация ограничений целостности базы данных

В листингах 3.7–3.13 показана реализация ограничений целостности базы данных (ограничения задаются при создании таблиц).

Листинг 3.7 – Ограничения для таблицы «Пользователи»

```
CREATE TABLE [Users] (
    [Id] [bigint] IDENTITY(1, 1) CHECK (Id > 0),
    [Role] [varchar](32) NOT NULL,
    [Email] [varchar](32) NOT NULL UNIQUE,
    [Password] [varchar](64) NOT NULL,
    [FirstName] [nvarchar](32),
    [LastName] [nvarchar](32),
    [RegDate] [datetime] NOT NULL,
```

```
    PRIMARY KEY (Id),
);
```

Листинг 3.8 – Ограничения для таблицы «Заказы»

```
CREATE TABLE [Orders] (
    [Id] [bigint] IDENTITY(1, 1) CHECK (Id > 0),
    [UserId] [bigint] NOT NULL CHECK (UserId > 0),
    [Status] [nvarchar](16) NOT NULL,
    PRIMARY KEY (Id),
    CONSTRAINT FK_USERS_ID
    FOREIGN KEY (UserId) REFERENCES [Users] (Id)
    ON DELETE CASCADE,
);
```

Листинг 3.9 – Ограничения для таблицы «Рейсы»

```
CREATE TABLE [Flights] (
    [Id] [bigint] IDENTITY(1, 1) CHECK (Id > 0),
    [PlaneId] [bigint] NOT NULL CHECK (PlaneId > 0),
    [DeparturePoint] [nvarchar](32) NOT NULL,
    [ArrivalPoint] [nvarchar](32) NOT NULL,
    [DepartureDateTime] [datetime] NOT NULL,
    [ArrivalDateTime] [datetime] NOT NULL,
    PRIMARY KEY (Id),
    CHECK (ArrivalDateTime > DepartureDateTime),
    CONSTRAINT FK_PLANES_ID
    FOREIGN KEY (PlaneId) REFERENCES [Planes] (Id)
    ON DELETE CASCADE
);
```

Листинг 3.10 – Ограничения для таблицы «Самолеты»

```
CREATE TABLE [Planes] (
    [Id] [bigint] IDENTITY(1, 1) CHECK (Id > 0),
    [Manufacturer] [nvarchar](32) NOT NULL,
    [Model] [nvarchar](16) NOT NULL,
    [EconomyClassNum] [integer] NOT NULL CHECK (EconomyClassNum
      >= 0),
    [BusinessClassNum] [integer] NOT NULL CHECK
      (BusinessClassNum >= 0),
    [FirstClassNum] [integer] NOT NULL CHECK (FirstClassNum >=
      0),
    PRIMARY KEY (Id),
);
```

Листинг 3.11 – Ограничения для таблицы «Услуги»

```
CREATE TABLE [Services] (
    [Id] [bigint] IDENTITY(1, 1) CHECK (Id > 0),
    [Name] [nvarchar](64) NOT NULL,
    [Price] [money] NOT NULL CHECK (Price > 0),
    [EconomyClass] [bit] NOT NULL,
    [BusinessClass] [bit] NOT NULL,
    [FirstClass] [bit] NOT NULL,
    PRIMARY KEY (Id),
);
```

Листинг 3.12 – Ограничения для таблицы «Билеты»

```
CREATE TABLE [Tickets] (
    [Id] [bigint] IDENTITY(1, 1) CHECK (Id > 0),
    [FlightId] [bigint] NOT NULL CHECK (FlightId > 0),
    [OrderId] [bigint],
    [Row] [int] NOT NULL CHECK (Row > 0),
    [Place] [char] NOT NULL,
    [Class] [nvarchar](16) NOT NULL,
    [Refund] [bit] NOT NULL,
    [Price] [money] CHECK (Price > 0),
    PRIMARY KEY (Id),
    CONSTRAINT FK_FLIGHTS_ID
    FOREIGN KEY (FlightId) REFERENCES [Flights] (Id)
    ON DELETE CASCADE,
);
```

Листинг 3.13 – Ограничения для вспомогательной таблицы-связки «Билеты-Услуги»

```
CREATE TABLE [TicketsServices] (
    [TicketId] [bigint] CHECK (TicketId > 0),
    [ServiceId] [bigint] CHECK (ServiceId > 0),
    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 3.14 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 [15].

Листинг 3.14 – 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 Tickets t ON o.Id = t.OrderId WHERE o.Id = @OrderId;
    SELECT @sumServices = COALESCE(SUM(s.Price), 0) FROM Orders o
    JOIN Tickets t ON t.OrderId = @OrderId JOIN
    TicketsServices ts ON t.Id = ts.TicketId
    JOIN Services s ON s.Id = ts.ServiceId WHERE o.Id = @OrderId;
    SELECT @result = @sumTickets + @sumServices
RETURN @result
END
```

3.5 Реализация ролевой модели на уровне базы данных

В листинге 3.15 показана реализация ролевой модели на уровне базы данных, которая состоит из трех ролей: клиента (db_customer), модератора (db_modreator) и администратора (db_admin).

Листинг 3.15 – Ролевая модель на уровне базы данных

```
CREATE ROLE db_customer;
CREATE ROLE db_moderator;
CREATE ROLE db_admin;
GRANT EXECUTE ON GetOrderPrice TO db_customer, db_moderator,
   db_admin;
GRANT SELECT ON Flights TO db_customer, db_moderator, db_admin;
GRANT SELECT, UPDATE, INSERT ON Orders TO db_customer,
   db_moderator, db_admin;
GRANT SELECT ON Planes TO db_customer, db_moderator, db_admin;
GRANT SELECT ON Services TO db_customer, db_moderator, db_admin;
GRANT SELECT, UPDATE ON Tickets TO db_customer, db_moderator,
   db_admin;
GRANT SELECT, UPDATE, INSERT, DELETE ON TicketsServices TO
   db_customer, db_moderator, db_admin;
GRANT SELECT, UPDATE, INSERT ON Users TO db_customer,
   db_moderator, db_admin;
GRANT UPDATE, INSERT, DELETE ON Flights TO db_moderator,
   db_admin;
GRANT UPDATE, INSERT, DELETE ON Planes TO db_moderator, db_admin;
GRANT INSERT, DELETE ON Tickets TO db_moderator, db_admin;
GRANT DELETE ON Orders TO db_admin;
GRANT UPDATE, INSERT, DELETE ON Services TO db_admin;
GRANT DELETE ON Users TO db_admin;
DENY CREATE TABLE TO db_customer, db_moderator;
GRANT CREATE TABLE TO db_admin;
GRANT ALTER ANY SCHEMA TO db_admin;
DENY DELETE ON OBJECT::Users TO db_customer, db_moderator;
DENY DELETE ON OBJECT::Orders TO db_customer, db_moderator;
DENY DELETE ON OBJECT::Tickets TO db_customer, db_moderator;
DENY DELETE ON OBJECT::Flights TO db_customer, db_moderator;
DENY DELETE ON OBJECT::Planes TO db_customer, db_moderator;
DENY DELETE ON OBJECT::Services TO db_customer, db_moderator;
DENY DELETE ON OBJECT::TicketsServices TO db_customer,
   db_moderator;
```

```
GRANT DELETE ON OBJECT::Users TO db_admin;
GRANT DELETE ON OBJECT::Orders TO db_admin;
GRANT DELETE ON OBJECT::Tickets TO db_admin;
GRANT DELETE ON OBJECT::Flights TO db_admin;
GRANT DELETE ON OBJECT::Planes TO db_admin;
GRANT DELETE ON OBJECT::Services TO db_admin;
GRANT DELETE ON OBJECT::TicketsServices TO db_admin;
CREATE LOGIN customer WITH PASSWORD = 'customer';
CREATE USER customer FOR LOGIN customer;
CREATE LOGIN moderator WITH PASSWORD = 'moderator';
CREATE USER moderator FOR LOGIN moderator;
CREATE LOGIN admin WITH PASSWORD = 'admin';
CREATE USER admin FOR LOGIN admin;
ALTER ROLE db_customer ADD MEMBER customer;
ALTER ROLE db_moderator ADD MEMBER moderator;
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 3.16–3.17 показаны соответственно интерфейс репозитория для сущности «Заказ» и реализация одного из его методов.

Листинг 3.16 – 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 3.17 – 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 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 3.18 показана реализация одного из mock-тестов для сущности «Пользователь». Данный тест проверяет корректность работы метода регистрации нового пользователя.

Листинг 3.18 – 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 3.19 показана реализация одного из интеграционных тестов для сущности «Пользователь».

Листинг 3.19 – 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 выбраны средства реализации программного обеспечения; 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 [16].

4.2 Исследование характеристик разработанного программного обеспечения

Индекс — это объект базы данных, обеспечивающий дополнительные способы быстрого поиска и извлечения данных. Индекс может создаваться на одном (простой индекс) или нескольких (составной индекс) атрибутах. Если в таблице нет индекса, то поиск нужных строк выполняется простым сканированием по всей таблице. При наличии индекса время поиска нужных строк можно существенно уменьшить [17].

Индексирование не влияет на размещение данных какой-либо таблицы в базе данных. 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 [9].

В Microsoft SQL Server индексы хранятся в виде сбалансированных деревьев. Представление индекса в виде сбалансированного дерева означает, что стоимость поиска любой строки остается относительно постоянной, независимо от того, где находится эта строка [17].

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 [18].

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 и некластеризованный индекс по атрибуту «ID пользователя» в таблице «Заказы» и измерить время выполнения запроса поиска заказа по идентификатору пользователя без индекса, с кластеризованным и некластеризованным индексом.

4.3 Время выполнения запроса

В листинге 4.1 приведены команды создания кластеризованного и некластеризованного индекса для таблицы «Заказы».

Листинг 4.1 – Создание кластеризованного и некластеризованного индекса

```
CREATE CLUSTERED INDEX user_id_index ON Orders (UserId);
CREATE NONCLUSTERED INDEX user_id_index ON Orders (UserId);
```

В таблицах 4.1–4.2 приведено время выполнения в миллисекундах за-

проса без индекса, с кластеризованным и некластеризованным индексом при поиске первой и последней записи в таблице соответственно.

Таблица 4.1 – Время выполнения запроса при поиске первой записи в таблице

Кол-во строк	**Время выполнения запроса без индекса, мс**	**Время выполнения запроса с кластеризованным индексом, мс**	**Время выполнения запроса с некластеризованным индексом, мс**
100	468	440	450
1000	471	421	439
10000	510	425	445
50000	757	414	453
100000	1642	472	460
250000	10701	498	510

Таблица 4.2 – Время выполнения запроса при поиске последней записи в таблице

Кол-во строк	**Время выполнения запроса без индекса, мс**	**Время выполнения запроса с кластеризованным индексом, мс**	**Время выполнения запроса с некластеризованным индексом, мс**
100	495	442	461
1000	644	448	452
10000	715	457	470
50000	2136	440	445
100000	5978	498	510
250000	50571	524	520

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

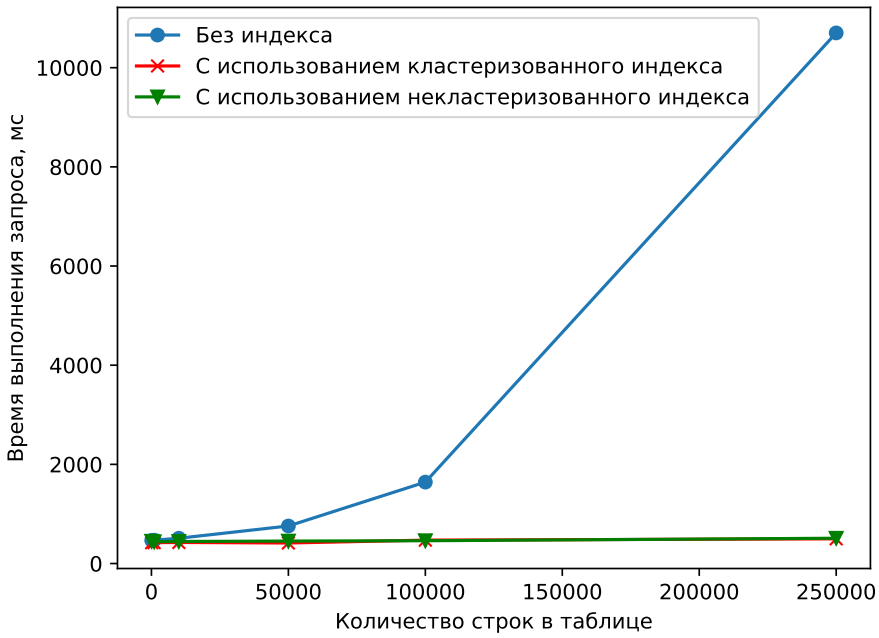

Рисунок 4.1 – Зависимость времени выполнения запроса от количества строк в таблице при поиске первой записи

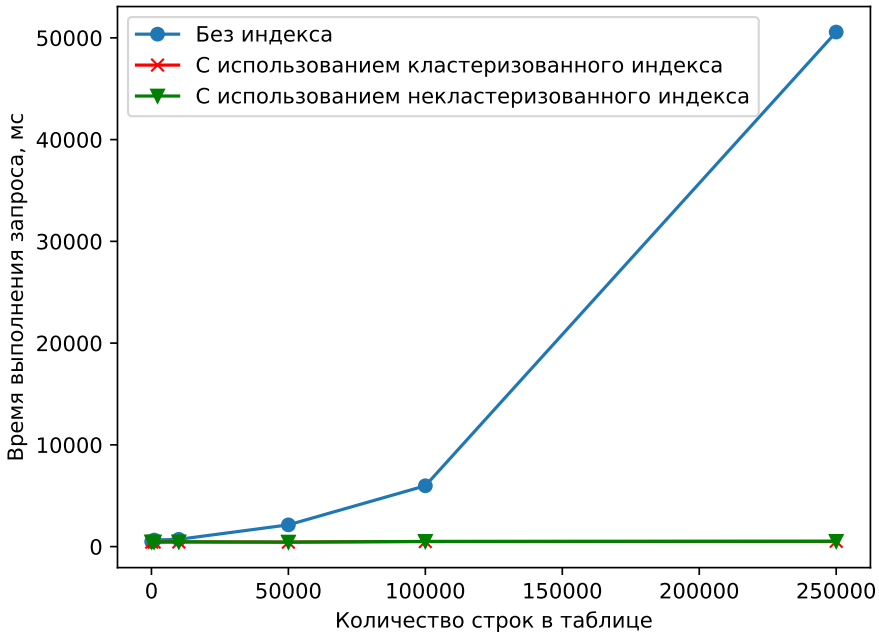

Рисунок 4.2 – Зависимость времени выполнения запроса от количества строк в таблице при поиске последней записи

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 индекса, с кластеризованным и некластеризованным индексом. 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, а время выполнения запроса с использованием двух типов индексов остается практически неизменным. Разница во времени выполнения запроса с кластеризованным и некластеризованным индексом практически отсутствует. 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 авиакомании, 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 и кластеризованного, и некластеризованного 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. *Авиасейлс.* Поиск дешёвых авиабилетов. — 2023. — (Дата обращения: 10.05.2023). `https://www.aviasales.ru/`.
4. *Airways Q.* Qatar Airways. — 2023. — (Дата обращения: 10.05.2023). `https://www.qatarairways.com/`.
5. *Emirates.* Emirates. — 2023. — (Дата обращения: 10.05.2023). `https://www.emirates.com/`.
6. *Airlines U.* United Airlines. — 2023. — (Дата обращения: 10.05.2023). `https://www.united.com/`.
7. *Аврунев О. Е.*, *Стасышин В. М.* Модели баз данных // Новосибирский государственный технический университет. — 2018. — С. 124.
8. *Виноградов В. И.*, *Виноградова М. В.* Постреляционные модели баз данных и языки запросов // Московский государственный технический университет имени Н. Э. Баумана. — 2016. — С. 100.
9. *Карпова И. П.* Базы данных. Учебное пособие // М. — 2009. — С. 131.
10. *Карпова И. П.* Введение в базы данных. Учебное пособие // Московский Государственный институт электроники и математики. — 2005. — С. 75.
11. *Davidson T.* Single Page Application (SPA) vs Multi Page Application (MPA): Which Is The Best? — 2023. — (Дата обращения: 15.05.2023). `https://cleancommit.io/blog/spa-vs-mpa-which-is-the-king/`.
12. *Tran T.* SPA vs. MPA: Pros, Cons and How To Make Final Choice. — 2022. — (Дата обращения: 15.05.2023). `https://www.simicart.com/blog/spa-vs-mpa/`.

13. *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/.

14. *Ranking D.-E.* Knowledge Base of Relational and NoSQL Database Management Systems. — 2023. — (Дата обращения: 10.05.2023). https://db-engines.com/en/ranking.

15. *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.

16. *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.

17. *Гаврилова Ю. М.* Конспект лекций по базам данных // Московский государственный технический университет им. Н. Э. Баумана. — 2022.

18. *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