



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 магазина одежды»*

Студент ИУ7-63Б
(Группа)

(Подпись, дата)

И. А. Гринкевич
(И. О. Фамилия)

Руководитель курсовой работы

(Подпись, дата)

А. Л. Исаев
(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 авиакомпаний 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.

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	8
1.3 Модели баз данных	9
1.3.1 Дореляционные базы данных	9
1.3.2 Реляционные базы данных	9
1.3.3 Постреляционные базы данных	10
1.4 Информация, подлежащая хранению в базе данных	10
1.5 ER-диаграмма сущностей базы данных	12
1.6 Пользователи приложения	12
2 Конструкторская часть	16
2.1 Описание сущностей базы данных	16
2.2 Описание ограничений целостности базы данных	20
2.3 Описание проектируемой функции на уровне базы данных	22
2.4 Описание ролевой модели на уровне базы данных	24
3 Технологическая часть	25
3.1 Средства реализации	25
3.2 Реализация сущностей базы данных	28
3.3 Реализация ограничений целостности базы данных	30
3.4 Реализация функции на уровне базы данных	34
3.5 Реализация ролевой модели на уровне базы данных	35
3.6 Реализация интерфейса доступа к базе данных	36
3.7 Тестирование разработанного функционала	37
4 Исследовательская часть	40
4.1 Технические характеристики устройства	40

4.2	Исследование характеристик разработанного программного обеспечения	40
4.3	Время выполнения запроса	41
ЗАКЛЮЧЕНИЕ		45
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ		46
ПРИЛОЖЕНИЕ А Презентация		48

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

В современном мире информация является ключевым ресурсом для успешного ведения бизнеса. Эффективное управление данными становится все более важным для компаний, особенно в сфере розничной торговли. Магазины одежды не являются исключением, их успешная деятельность напрямую зависит от умения эффективно хранить, управлять разнообразными данными, связанными с продажами, ассортиментом, клиентами и многими другими аспектами бизнеса.

Целью данной курсовой работы является разработка базы данных для хранения и обработки данных магазина одежды.

Для достижения поставленной цели необходимо решить следующие задачи:

- 1) проанализировать предметную область;
- 2) сформулировать требования и ограничения к разрабатываемой базе данных;
- 3) формализовать информацию, хранимую в БД;
- 4) разработать структуру базы данных и определить ролевую модель в контексте БД;
- 5) избрать инструменты для разработки и реализовать спроектированную базу данных.;
- 6) исследовать характеристики разработанного программного обеспечения.

1 Аналитическая часть

1.1 Анализ предметной области

В настоящее время магазины одежды играют важную роль в розничной торговле России. Они предоставляют широкий ассортимент товаров и являются основными местами для покупки одежды и аксессуаров для населения. Среди наиболее популярных онлайн-магазинов одежды в России следует выделить такие платформы, как stockmann.ru и lamoda.ru. Они зарекомендовали себя как надежные и удобные ресурсы для поиска и приобретения модной одежды и обуви.

Однако, несмотря на популярность и широкий выбор товаров, часто пользователи сталкиваются с проблемами в использовании этих онлайн-платформ. Один из основных аспектов, на который обращают внимание, является перегруженный интерфейс и сложность использования. Слишком много информации, разнообразные дополнительные функции и не всегда интуитивно понятный дизайн могут создавать неудобства для покупателей. По этой причине одной из задач курсовой работы заключается в разработке базы данных, которая позволит магазину одежды предоставить простой и понятный функционал для своих клиентов.

Разработка простого и понятного функционала для магазина одежды позволит повысить удовлетворенность клиентов и укрепить позиции магазина на рынке.

1.2 Требования к базе данных и приложению

Приложение должно предоставлять пользователям возможность искать товары по следующим параметрам:

- 1) пол человека;
- 2) категория;
- 3) бренд.

Также должна быть возможность сортировки товаров по возрастанию или убыванию цены.

Пользователи должны иметь возможность зарегистрироваться, авторизоваться, просмотреть информацию о товарах и брендах, добавить и удалить товары из корзины, совершить заказ, просмотреть историю заказов.

Администратор должен иметь возможность просматривать информацию о заказах пользователей, изменять информацию о пользователях, заказах, товарах и брендах, а также загружать и удалять новые товары и бренды.

1.3 Модели баз данных

1.3.1 Дореляционные базы данных

Дореляционные базы данных - это тип баз данных, который не использует традиционные таблицы и связи, характерные для реляционных баз данных. Вместо этого они организуют данные в более гибких структурах, таких как документы, графы или временные ряды. Это позволяет более эффективно хранить и обрабатывать данные с учетом их специфики. Примерами дореляционных баз данных могут служить NoSQL базы данных, такие как MongoDB, CouchDB, Cassandra.

Преимущества дореляционных баз данных включают более гибкую структуру данных, что делает их подходящими для работы с неструктурированными или полуструктурированными данными. Они также могут быть более масштабируемыми в определенных сценариях, таких как обработка больших объемов данных.

Однако дореляционные базы данных могут потребовать более сложных запросов и могут не обеспечивать те же уровни нормализации данных, что и реляционные базы данных.

1.3.2 Реляционные базы данных

Реляционные базы данных - это тип баз данных, который использует табличную структуру для хранения данных и устанавливает связи между этими таблицами. Каждая таблица представляет собой набор записей с атрибутами,

а связи между таблицами обеспечивают возможность объединения данных из разных таблиц при выполнении запросов.

Преимущества реляционных баз данных включают структурированный подход к хранению данных, что обеспечивает эффективность при работе с нормализованными данными. Они также обладают мощным языком запросов SQL, который позволяет легко извлекать и модифицировать данные.

Однако реляционные базы данных могут столкнуться с проблемами при работе с неструктурированными данными или данными, требующими гибких схем.

1.3.3 Постреляционные базы данных

Постреляционные базы данных - это новый подход к хранению данных, который пытается объединить преимущества как дореляционных, так и реляционных баз данных. Они предлагают гибкую схему данных, что делает их более способными к работе с разнообразными данными. Постреляционные базы данных также предлагают улучшенные механизмы обработки больших объемов данных.

Примеры постреляционных баз данных включают ArangoDB, Amazon DynamoDB и другие. Эти системы стремятся объединить гибкость дореляционных баз данных с мощностью реляционных систем.

1.4 Информация, подлежащая хранению в базе данных

В базе данных нужно будет хранить информацию о пяти сущностях: пользователь, бренд, товар, заказ и позиция заказа.

1. Пользователь. Информация о зарегистрированных пользователях: логин, пароль, имя, пол, роль.
2. Бренд. Информация о брендах товаров: название, год основания, идентификатор логотипа, название компании владельца.
3. Товар. Информация о товаре: категория, цена, пол, идентификатор изоб-

ражения, бренд, наличие.

4. Заказ. Информация о заказе: дата совершения, заказчик, цена, статус.
5. Позиция заказа. Информация о позиции в заказе: заказ, товар, количество.

Идентификаторы логотипа бренда и изображения товара представляют собой ID на сторонних сервисах для хранения изображений.

Так как база данных не имеет большого количества связей и хранимые в ней данные структурированы, для достижения поставленной цели была выбрана реляционная модель баз данных.

1.5 ER-диаграмма сущностей базы данных

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

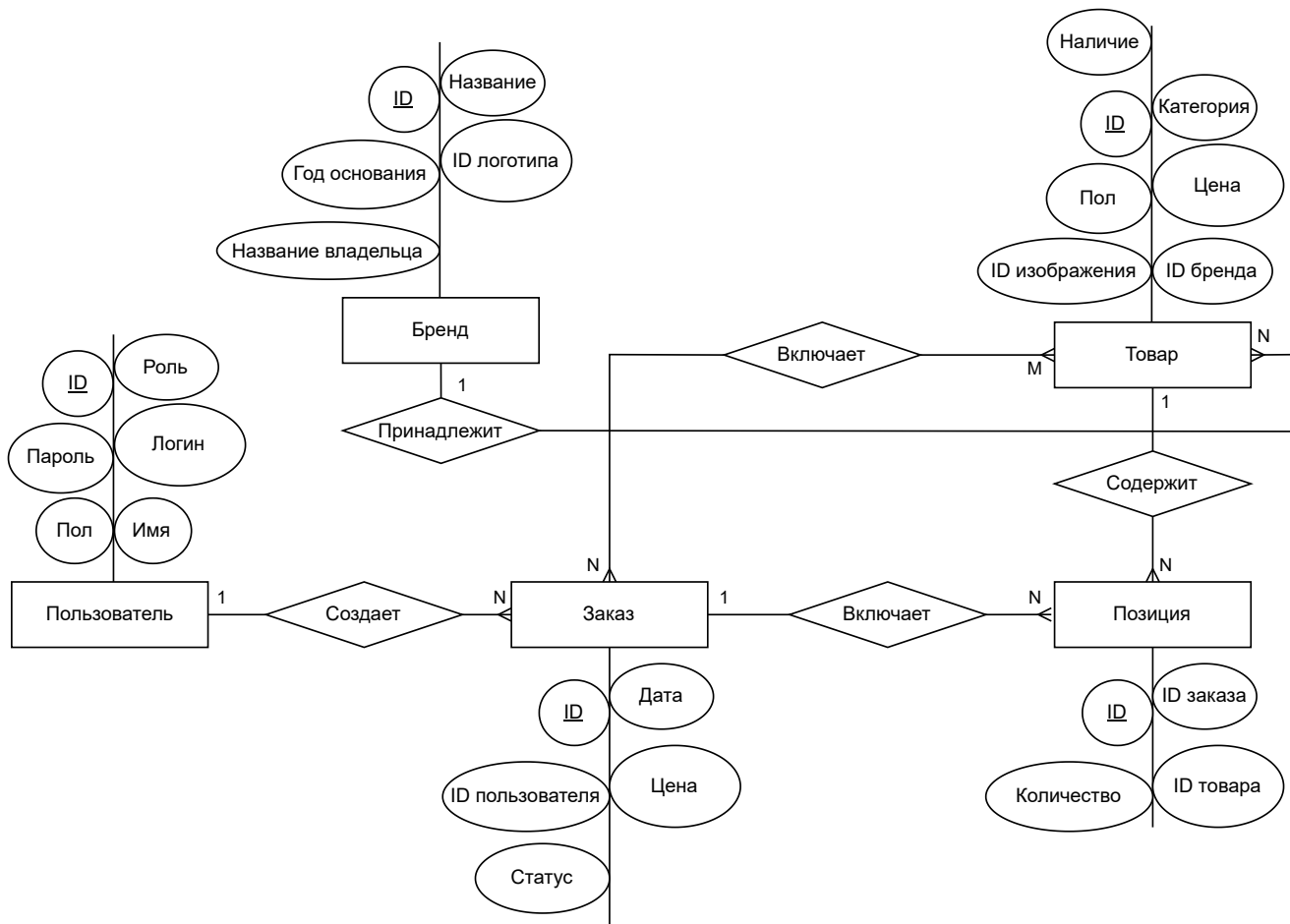


Рисунок 1.1 – ER-диаграмма

1.6 Пользователи приложения

Пользователи могут иметь одну из трех ролей: посетитель, клиент и администратор.

Посетитель — неавторизованный пользователь. Он может смотреть товары и бренды, авторизоваться и зарегистрироваться. На рисунке 1.2 представлена use-case диаграмма для роли посетитель.

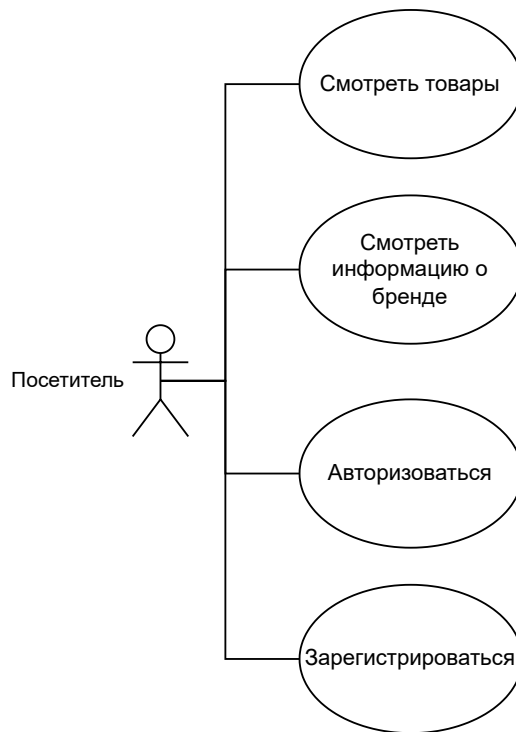


Рисунок 1.2 – Use-case диаграмма для роли посетитель

Клиент — зарегистрированный и авторизованный пользователь. Он имеет те же возможности что и посетитель (кроме регистрации), но в добавок он может добавить или удалить товары из корзины, совершить заказ, просмотреть историю заказов. На рисунке 1.3 представлена use-case диаграмма для роли клиент.

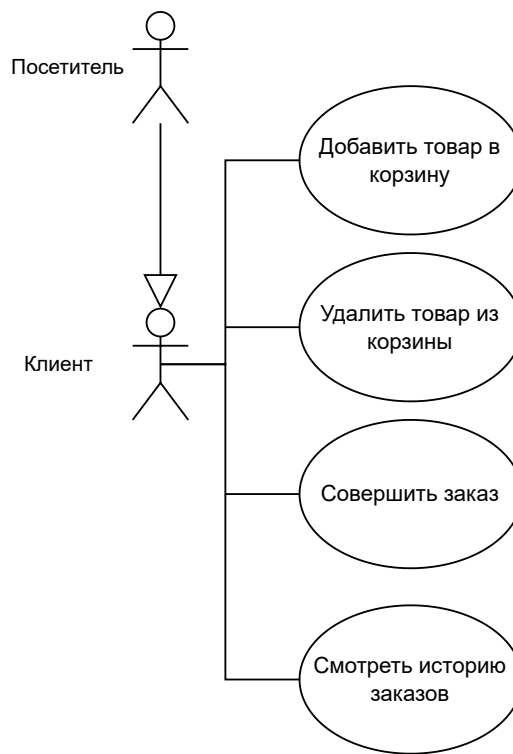


Рисунок 1.3 – Use-case диаграмма для роли клиент

Администратор имеет те же возможности что и клиент, но вдобавок может просматривать информацию о заказах пользователей, изменять информацию о пользователях, заказах, товарах и брендах, а также загружать и удалять новые товары и бренды. На рисунке 1.4 представлена use-case диаграмма для роли администратор.

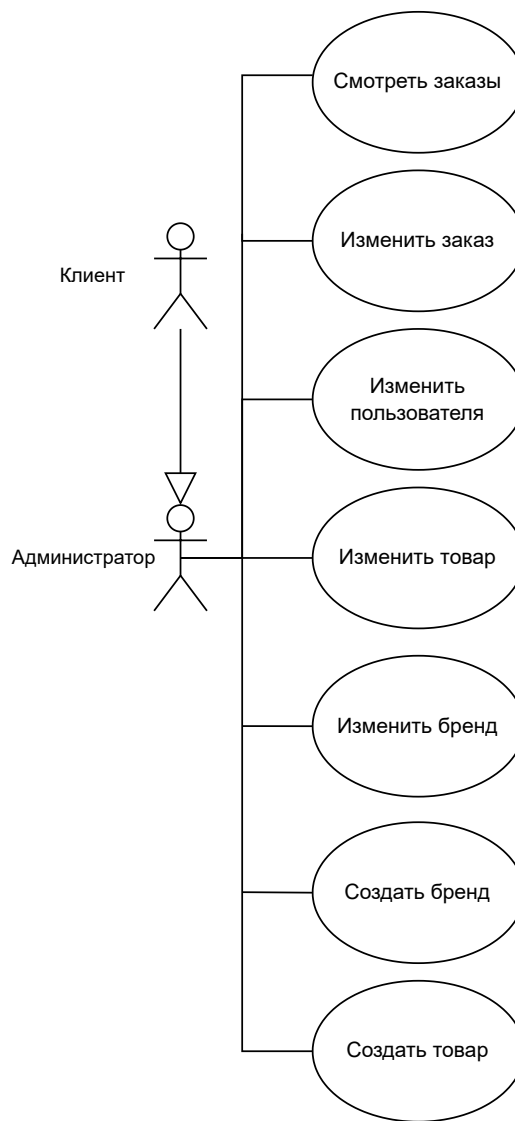


Рисунок 1.4 – Use-case диаграмма для роли администратор

Вывод из аналитической части

По ходу выполнения аналитической части данной курсовой работы была проанализирована рассматриваемая предметная область, а так же сформулированы требования и ограничения к разрабатываемой БД. Были рассмотрены модели баз данных и выбрана подходящая под поставленные задачи. Также была формализована информация, хранящиеся в БД и представлены диаграммы сущностей разрабатываемой базы данных и вариантов использования пользователями.

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.

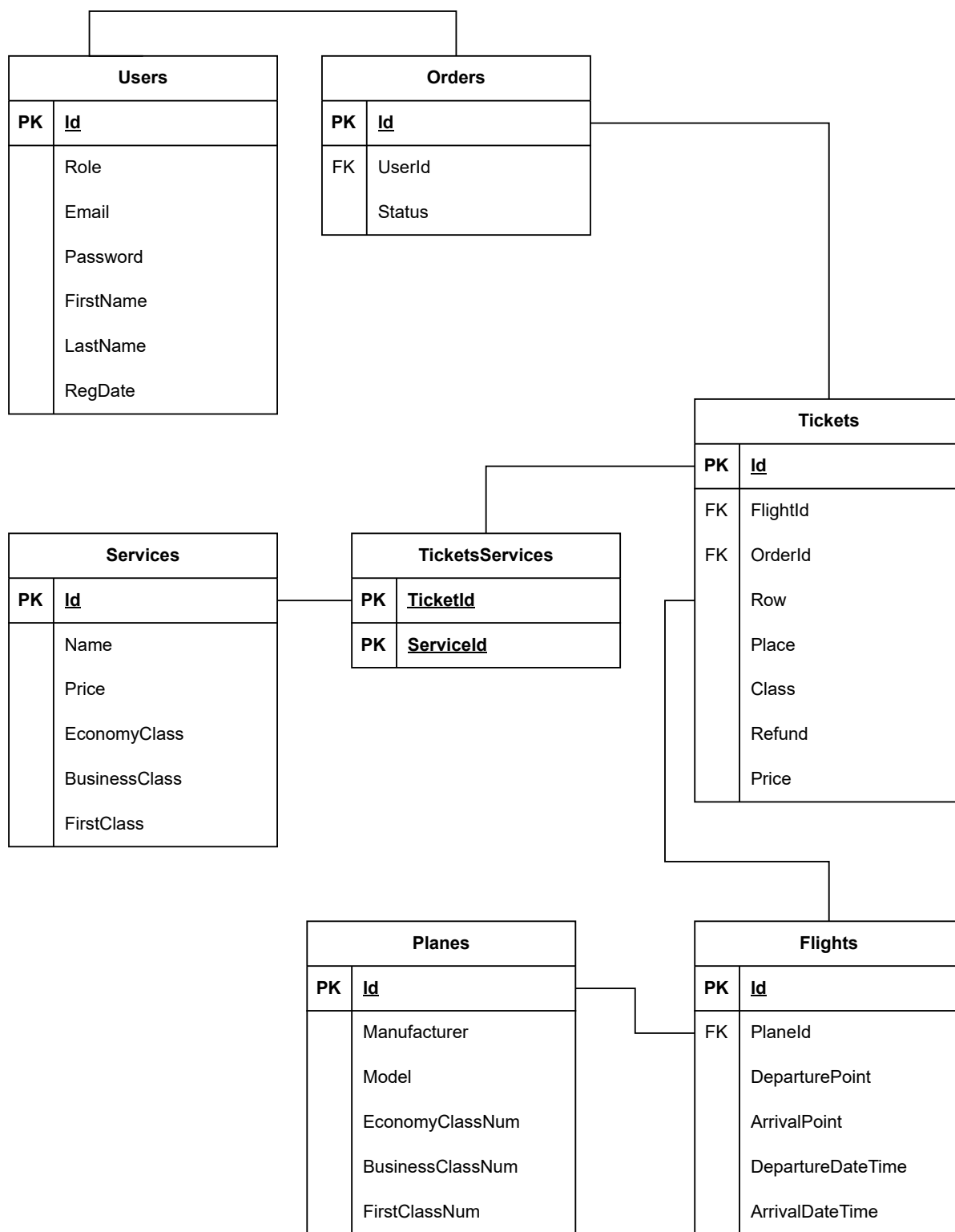


Рисунок 2.1 – Диаграмма базы данных

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 и время 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 времени 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 должна 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.

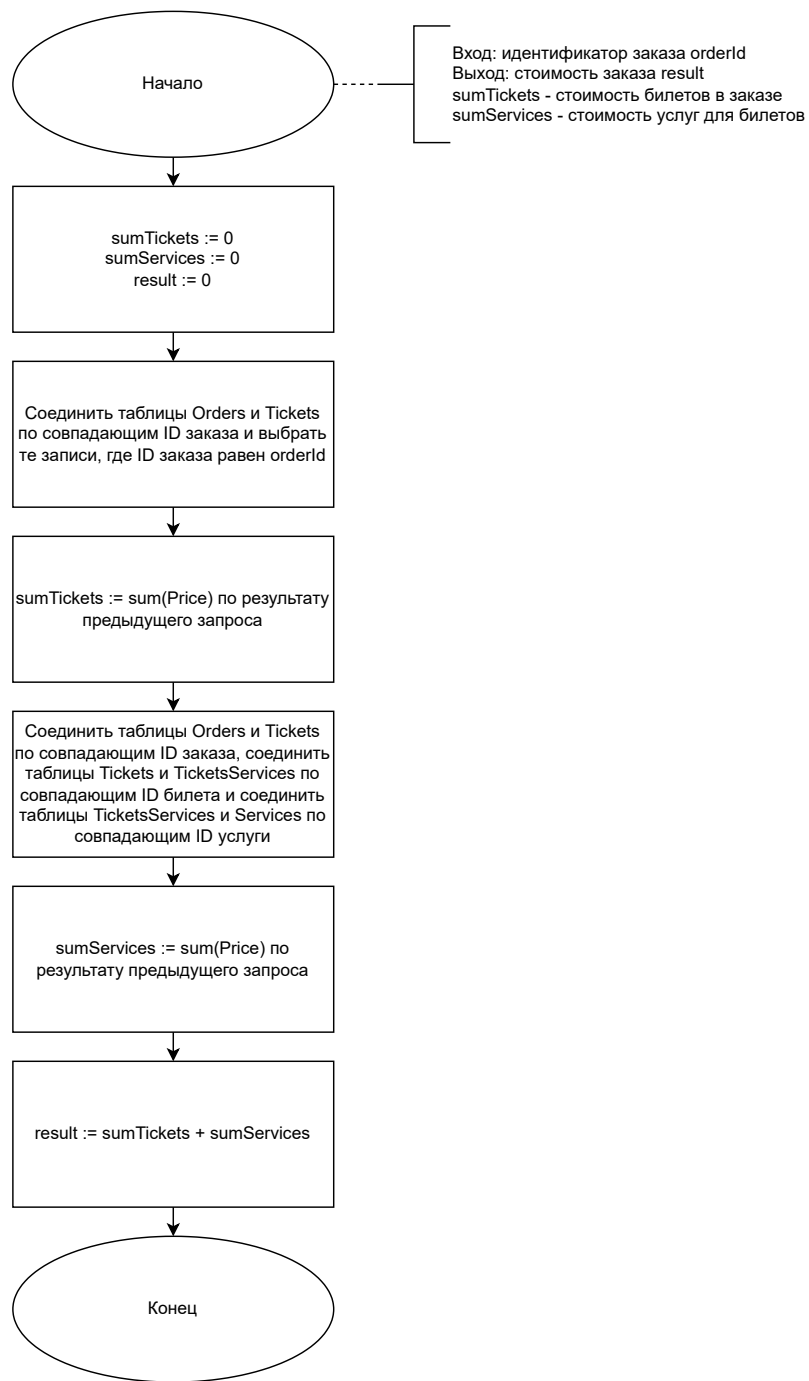


Рисунок 2.2 – Функция, возвращающая стоимость заказа

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. 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.

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 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 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) МРА, 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 МРА может занимать независимое место в результатах поиска, потенциально увеличивая трафик на сайт;
- 2) поскольку МРА 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 МРА часто быстрее, поскольку браузеру нужно загрузить только содержимое текущей страницы, а не все содержимое всего приложения;
- 4) МРА — это классический способ создания веб-страниц, поэтому он совместим с большинством старых браузеров и устаревших систем [11].

Основные недостатки многостраничных приложений:

- 1) поскольку МРА требуют полной перезагрузки страницы, когда пользователь взаимодействует с приложением, они обеспечивают низкую производительность, что негативно влияет на взаимодействие с пользователем;
- 2) МРА 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) МРА 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 веб-сайтов 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 веб-приложения 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_moderator) 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;
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: мок-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 мок-тестов для сущности «Пользователь». Данный тест проверяет корректность работы метода регистрации нового пользователя.

Листинг 3.18 – Мок-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 запроса 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 кла- стеризованным индексом, мс	Время выполнения запроса с некла- стеризованным 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 кла- стеризованным индексом, мс	Время выполнения запроса с некла- стеризованным 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.

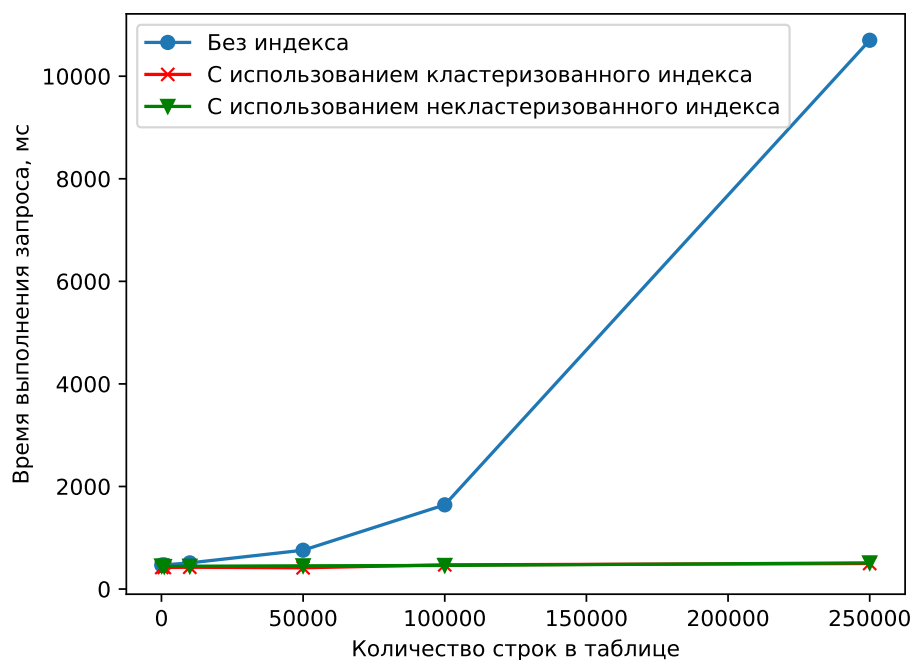


Рисунок 4.1 – Зависимость времени выполнения запроса от количества строк в таблице при поиске первой записи

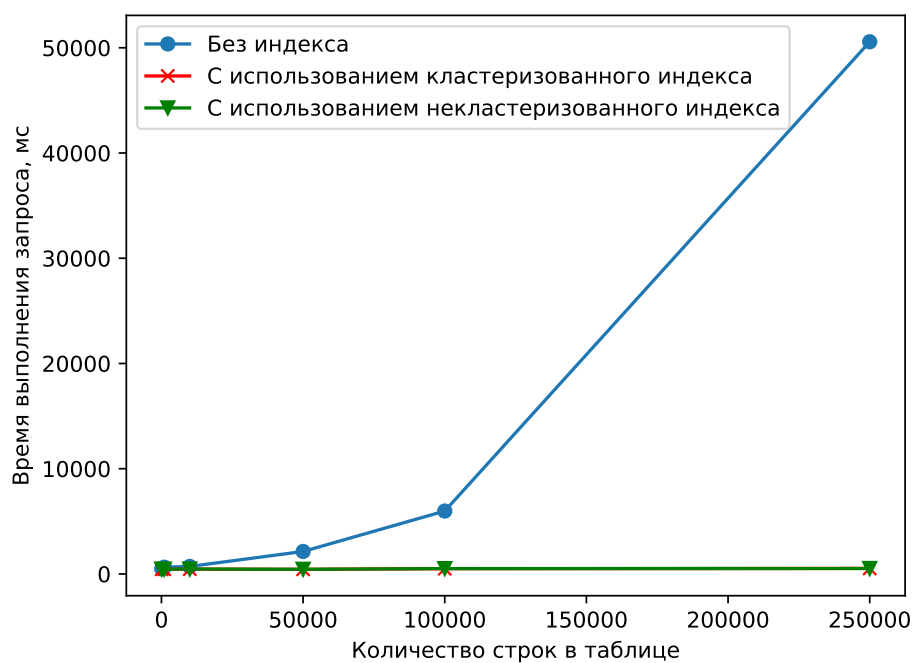


Рисунок 4.2 – Зависимость времени выполнения запроса от количества строк в таблице при поиске последней записи

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 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 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