



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

ФАКУЛЬТЕТ

ИУ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»

КАФЕДРА

ИУ7 «ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

*Разработка базы данных для хранения и
обработки данных магазина видеоигр*

Студент

ИУ7-62Б

(группа)

(подпись, дата)

Смирнов

(И.О. Фамилия)

Руководитель курсового
проекта

(подпись, дата)

Назаренко Н.В.

(И.О. Фамилия)

Консультант

(подпись, дата)

(И.О. Фамилия)

2025 г.

РЕФЕРАТ

Расчетно-пояснительная записка содержит 56 с., 9 рис., 5 табл., 11 ист.

Цель работы: разработка базы данных и веб-приложения для хранения, обработки и управления данными магазина видеоигр, специализирующегося на продаже цифровых ключей.

Ключевые слова: видеоигры, цифровые ключи, базы данных, Microsoft SQL Server, реляционная модель, ASP.NET, веб-приложение.

В данной работе исследуются принципы проектирования и реализации баз данных для онлайн-магазинов видеоигр.

Объектом исследования является модель представления данных о пользователях, заказах и цифровых ключах в системе магазина видеоигр.

В качестве инструментов разработки выбраны Microsoft SQL Server для управления базой данных и ASP.NET для создания веб-приложения.

Результаты: разработана база данных и веб-приложение, обеспечивающие работу с данными магазина видеоигр. Проведен анализ систем управления базами данных, обоснован выбор Microsoft SQL Server.

СОДЕРЖАНИЕ

РЕФЕРАТ	4
ВВЕДЕНИЕ	8
1 Аналитическая часть	9
1.1 Анализ предметной области	9
1.2 Анализ существующих решений	10
1.3 Классификация СУБД	11
1.3.1 По модели данных	11
1.3.2 По архитектуре организации хранения данных	14
1.3.3 По способу доступа к БД	14
1.4 Определение ролей пользователей системы	15
1.5 Формализация данных	16
1.6 Выбор модели базы данных	17
2 Конструкторская часть	19
2.1 Формализация сущностей системы	19
2.1.1 Таблица Users	19
2.1.2 Таблица Games	20
2.1.3 Таблица Categories	20
2.1.4 Таблица Sellers	21
2.1.5 Таблица Orders	21
2.1.6 Таблица OrderItems	22
2.1.7 Таблица Reviews	22
2.1.8 Таблица ErrorLog	23
2.1.9 Таблица Gifts	23
2.2 Ролевая модель	24
2.2.1 Роль Guest (Гость)	24
2.2.2 Роль User (Покупатель)	24
2.2.3 Роль Seller (Продавец)	25
2.2.4 Роль Admin (Администратор)	25
2.3 Разработка хранимой процедуры	25
2.3.1 Описание процедуры	26

2.4	Разработка триггера для таблицы Users	27
2.4.1	Описание триггера	28
2.5	Разработка функций	28
2.5.1	Функция fn_GetUserTotalSpent	28
2.5.2	Функция fn_GetGameAverageRating	29
3	Технологическая часть	31
3.1	Выбор СУБД	31
3.2	Средства реализации	32
3.3	Создание таблиц	32
3.4	Индексы	37
3.5	Функции	38
3.6	Хранимые процедуры и триггеры	38
3.6.1	Хранимая процедура sp_ManageSqlLogin	39
3.6.2	Триггер trg_Users_AfterInsertDelete	40
3.6.3	Триггер trg_Sellers_AfterInsertDelete	42
3.7	Модель ролей и прав доступа	43
3.8	Тестирование триггера	47
3.8.1	Классы эквивалентности	47
3.8.2	Тестовые случаи	48
3.9	Пример работы программы	49
3.10	Вывод	49
4	Исследовательский раздел	50
4.1	Технические характеристики	50
4.2	Временные характеристики	50
4.3	Вывод	53
	ЗАКЛЮЧЕНИЕ	54
	СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	55
	ПРИЛОЖЕНИЕ А	56

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

БД — база данных.

СУБД — система управления базами данных.

Gamesbakery — разрабатываемый онлайн-магазин цифровых ключей для видеоигр, обеспечивающий покупку, продажу и передачу ключей.

ВВЕДЕНИЕ

Современная индустрия видеоигр стремительно развивается, а вместе с ней растет популярность онлайн-магазинов, предлагающих цифровые копии игр в виде ключей активации. Такие платформы становятся удобным инструментом для геймеров, желающих быстро приобрести игры, и для дистрибьюторов, стремящихся эффективно распространять свой продукт. Однако существующие решения не всегда удовлетворяют потребности пользователей, особенно в части гибкости покупки и передачи нескольких экземпляров одной игры. Это создает необходимость разработки новых систем, способных закрыть подобные пробелы.

Цель курсовой работы — разработка базы данных для хранения и обработки информации магазина видеоигр:

- изучить предметную область магазинов видеоигр;
- сформулировать требования и ограничения к разрабатываемой базе данных и приложению;
- спроектировать сущности базы данных, определить ограничения целостности, а также разработать функции и ролевую модель на уровне базы данных;
- реализовать сущности базы данных магазина видеоигр с учетом ограничений целостности и описать интерфейс доступа к базе данных;
- провести исследование влияния индексов на скорость выполнения запросов разработанной системы.

1 Аналитическая часть

В данном разделе проводится анализ предметной области интернет магазинов видеоигр, формализуются данные, а также анализируются существующие решения.

1.1 Анализ предметной области

Предметная область охватывает функционирование онлайн-магазина цифровых ключей для видеоигр. Основная цель системы — обеспечить удобное взаимодействие между покупателями, продавцами и администраторами, предоставляя средства для покупки, продажи, классификации и передачи цифровых копий игр.

База данных [1] — это организованная совокупность данных, предназначенная для хранения и обработки информации в рамках определенных процессов. В контексте магазина видеоигр база данных должна обеспечивать структурированное хранение данных о пользователях, играх, заказах, отзывах и категориях, а также поддерживать операции в реальном времени, такие как оформление заказов, добавление отзывов, обновление статуса заказов и управление ключами.

Для определения типа базы данных рассмотрим два основных подхода:

- **OLAP** [2] — метод обработки данных, ориентированный на анализ больших объемов информации, например, для построения отчетов и аналитики.
- **OLTP** [3] — метод обработки транзакций в реальном времени, подходящий для операций, таких как оформление заказов и обновление данных.

Учитывая, что магазин видеоигр требует быстрой обработки транзакций (например, оформление заказов, добавление отзывов), для данной работы больше подходит подход OLTP, так как он обеспечивает оперативное выполнение операций.

Для реализации системы необходимо выбрать систему управления базами данных (СУБД). **СУБД** [4] — это программное обеспечение, обеспечивающее создание, хранение, обновление и поиск информации в базе данных.

1.2 Анализ существующих решений

Для оценки текущих подходов к реализации магазинов цифровых ключей были проанализированы несколько популярных решений: *Steampay*[5], *Kupikod*[6] и *Zaka – Zaka*[7]. Сравнение проводилось по трем критериям: модель размещения ключей, система отзывов и возможность покупки нескольких ключей.

В таблице 1.1 представлено сравнение аналогов и предлагаемого решения.

Таблица 1.1 – Сравнение существующих решений

Решение	Модель размещения ключей	Система отзывов	Система подарочных ключей
Steampay	Размещают проверенные продавцы	Комментарии	Покупка одной подарочной копии
Kupikod	Размещают официальные и частные лица	Звездная оценка	Отсутствует
Zaka-Zaka	Размещают проверенные продавцы/издатели	Отсутствует	Покупка нескольких ключей с возможностью подарить
Gamesbakery (предлагаемое решение)	Размещают проверенные продавцы	Комментарии + звездная оценка	Покупка нескольких ключей с возможностью подарить

Анализ показал, что существующие решения имеют ограничения: *Steampay* и *Kupikod* не поддерживают покупку нескольких копий с гибкой передачей (возможностью передачи купленных ключей другим пользователям через интерфейс системы), а *Zaka – Zaka* не предоставляет систему отзывов. *Gamesbakery* решает эти проблемы, предлагая комбинированную систему отзывов (комментарии и звезды) и возможность покупки нескольких ключей с их последующей передачей. На основе анализа можно сделать вывод, что ни одно из существующих решений не предоставляет одновременно комбинированную систему отзывов и возможность гибкой передачи

нескольких ключей, что делает разработку *Gamesbakery* актуальной и востребованной.

1.3 Классификация СУБД

В данном разделе рассматриваются основные подходы к классификации систем управления базами данных (СУБД), что позволяет обосновать выбор подходящей системы для реализации магазина видеоигр. Классификация проводится по трем ключевым критериям: модели данных, архитектуре организации хранения и способу доступа к базе данных.

1.3.1 По модели данных

Модель данных определяет принципы организации, хранения и взаимодействия информации в базе данных. В зависимости от подхода к структурированию данных выделяют три основные категории моделей: дореляционные, реляционные и постреляционные. Рассмотрим каждую из них более подробно.

Дореляционные модели Дореляционные модели данных были распространены на ранних этапах развития баз данных и характеризуются ограниченной гибкостью в управлении связями между данными [8]. К основным типам таких моделей относятся:

- **Инвертированные списки (файлы)** — данные хранятся в виде набора файлов, где для ускорения поиска используются индексы. Однако ограничения целостности в таких системах задаются на уровне приложений, что усложняет обеспечение согласованности данных и увеличивает вероятность ошибок.
- **Иерархическая модель** — информация организована в виде древовидной структуры, где каждый дочерний элемент связан только с одним родительским. Такой подход ограничивает возможности представления сложных связей, так как не позволяет дочернему элементу иметь несколько родителей.

- **Сетевые модели** — данные представлены в виде графа, что позволяет создавать более сложные связи по сравнению с иерархической моделью. Однако выборка данных в таких системах сильно зависит от их физической организации, что усложняет разработку и поддержку.

Дореляционные модели, несмотря на свою историческую значимость, не подходят для современных систем из-за ограниченной гибкости и сложности управления сложными связями.

Реляционные модели Реляционная модель, предложенная Эдгаром Коддом в 1970 году, стала стандартом для большинства современных баз данных. В этой модели данные хранятся в виде таблиц (отношений), каждая из которых состоит из строк и столбцов. Таблицы связаны между собой через ключи, что обеспечивает гибкость и целостность данных. Реляционная модель включает три основных аспекта:

- **Структурный аспект** — данные представлены в виде набора отношений, где каждое отношение имеет уникальное имя и состоит из кортежей (строк) с фиксированным набором атрибутов (столбцов).
- **Целостностный аспект** — для обеспечения согласованности данных вводятся ограничения целостности, такие как уникальность первичных ключей, ссылочная целостность через внешние ключи и доменные ограничения.
- **Манипуляционный аспект** — операции с данными выполняются с использованием реляционной алгебры или языков запросов, таких как SQL, что позволяет эффективно извлекать и модифицировать информацию.

Реляционная модель обладает рядом преимуществ: она обеспечивает структурированное хранение данных, поддерживает целостность на уровне СУБД и позволяет выполнять сложные запросы. Эта модель позволяет эффективно организовать данные о пользователях, играх, заказах и отзывах, а также поддерживать операции в реальном времени, такие как оформление заказов, добавление отзывов и обновление статуса заказов.

Постреляционные модели Постреляционные модели данных появились как развитие реляционной модели, устраняя некоторые ее ограничения, свя-

занные с обработкой сложных и неструктурированных данных [9]. Ключевые особенности постреляционных моделей:

- **Поддержка сложных типов данных** — в отличие от реляционной модели, где атрибуты таблиц ограничены простыми типами (числа, строки, даты), постреляционные модели позволяют хранить вложенные структуры, массивы, JSON или XML-объекты.
- **Расширенные механизмы связей** — постреляционные модели поддерживают более сложные отношения между данными, такие как наследование или полиморфизм, что упрощает моделирование сложных предметных областей.
- **Гибкость в управлении данными** — такие модели позволяют сочетать реляционные принципы с объектно-ориентированным подходом, что делает их удобными для современных приложений, работающих с большими объемами неструктурированных данных.

Для магазина видеоигр использование постреляционных моделей не является необходимым, так как предметная область хорошо описывается реляционной моделью, а данные (информация о пользователях, играх, заказах) имеют четкую структуру.

Выбор модели данных Для выбора подходящей модели данных проведем сравнение рассмотренных подходов по ключевым критериям: гибкость связей (ГС), поддержка сложных типов данных (ПСТД), производительность для OLTP-систем (OLTP) и сложность разработки (СР). Результаты сравнения представлены в таблице 1.2.

Таблица 1.2 – Сравнение моделей данных

Модель	ГС	ПСТД	OLTP	СР
Дореляционная	Низкая	Нет	Низкая	Высокая
Реляционная	Высокая	Нет	Высокая	Средняя
Постреляционная	Высокая	Да	Средняя	Высокая

На основе сравнения можно сделать вывод, что реляционная модель является предпочтительной для разрабатываемой системы, так как она обеспечивает высокую производительность для OLTP-операций, достаточную гибкость связей и умеренную сложность разработки, что соответствует требованиям магазина видеоигр.

1.3.2 По архитектуре организации хранения данных

Архитектура хранения данных определяет, как физически организованы данные в системе. Выделяют два основных типа:

- **Локальные** — все данные и компоненты СУБД размещены на одном устройстве. Такой подход прост в реализации, но ограничивает масштабируемость и увеличивает риски потери данных при сбоях.
- **Распределенные** — данные распределяются между несколькими устройствами, что повышает отказоустойчивость и позволяет обрабатывать большие объемы информации.

Для работы достаточно локальной архитектуры, так как система разрабатывается для небольшого магазина видеоигр, и на начальном этапе нет необходимости в распределенном хранении. Локальная архитектура упрощает разработку и тестирование, а также снижает требования к серверным ресурсам.

1.3.3 По способу доступа к БД

Способ доступа к базе данных определяет, как приложение взаимодействует с данными, и влияет на производительность и безопасность системы. Выделяют следующие типы:

- **Файл-серверные** — данные хранятся в виде файлов, а клиентское приложение получает доступ к ним напрямую. Это приводит к высокой нагрузке на сеть, так как все данные передаются клиенту, даже если требуется лишь их часть.
- **Клиент-серверные** — обработка запросов выполняется на сервере, а клиенту передаются только результаты. Такой подход минимизирует объем передаваемых данных, повышает безопасность и позволяет централизованно управлять доступом.
- **Встраиваемые** — СУБД встраивается непосредственно в приложение и работает на локальном устройстве. Этот подход подходит для автономных приложений, но неэффективен для веб-систем.
- **Сервисно-ориентированные** — используются для хранения сообще-

ний, метаинформации и промежуточных состояний в распределенных системах, что не соответствует задачам работы.

Для веб-приложения оптимальным выбором является клиент-серверная архитектура. Она обеспечивает высокую производительность за счет обработки запросов на сервере, минимизирует нагрузку на сеть и позволяет реализовать безопасный доступ к данным, что особенно важно для системы, работающей с заказами и отзывами пользователей.

1.4 Определение ролей пользователей системы

Для обеспечения безопасности и разделения прав доступа в системе необходимо определить следующие роли пользователей, которые будут взаимодействовать с базой данных:

1. Гость (Guest): неавторизованный пользователь, который может просматривать каталог игр и категории, но не имеет доступа к оформлению заказов и оставлению отзывов. Эта роль необходима для предоставления базового доступа к информации без регистрации.
2. Покупатель (User): зарегистрированный пользователь, который может оформлять заказы, оставлять отзывы, передавать купленные ключи другим пользователям и управлять своими данными. Роль нужна для реализации основного функционала системы для конечных пользователей.
3. Продавец (Seller): добавляет игры в каталог, редактирует информацию о них, управляет ключами для своих заказов и просматривает статистику продаж. Роль необходима для управления каталогом и обеспечения поставки ключей.
4. Администратор (Admin): управляет данными об играх, категориях, заказах, отзывах и пользователях (включая их блокировку), а также следит за целостностью системы. Роль нужна для администрирования и поддержания работы системы.

На рисунке 1.1 представлена диаграмма прецедентов, иллюстрирующая взаимодействие ролей с системой [10].



Рисунок 1.1 – Диаграмма прецедентов

1.5 Формализация данных

Для эффективного управления данными в системе информация представлена в виде таблиц. Сущности базы данных связаны с функциональными требованиями: таблица **Users** необходима для управления учетными записями и авторизации, **Games** — для хранения каталога игр и их характеристик, **Orders** и **OrderItems** — для обработки заказов и ключей, **Reviews** — для управления отзывами, **Categories** — для классификации игр, **Sellers** — для управления данными продавцов. Основные сущности базы данных:

1. **Пользователи (Users)**: хранит данные о покупателях для авторизации и управления учетными записями (UserID, имя, email, дата регистрации, страна, пароль, статус, баланс).
2. **Игры (Games)**: содержит информацию об играх для отображения в каталоге (GameID, CategoryID, название, цена, дата выпуска, описание, оригинальный издатель, статус продажи).
3. **Категории (Categories)**: классифицирует игры по жанрам для удобной навигации (CategoryID, название жанра, описание).
4. **Продавцы (Sellers)**: данные о продавцах для управления поставщиками (SellerID, имя, дата регистрации, средняя оценка).
5. **Заказы (Orders)**: информация о заказах для отслеживания покупок (OrderID, UserID, дата заказа, цена, статус выполнения, статус просроч-

ки).

6. **Элементы заказа (OrderItems)**: связывает заказы с играми для управления ключами (OrderItemID, OrderID, GameID, SellerID, текст ключа, статус подарка).
7. **Отзывы (Reviews)**: отзывы пользователей об играх для обратной связи (ReviewID, UserID, GameID, текст отзыва, оценка, дата создания).
8. **Подарки (Gifts)**: GiftID, SenderID, RecipientID, OrderItemID, дата подарка.

Связи между сущностями определены следующим образом: пользователь может сделать несколько заказов, каждый заказ включает элементы, связанные с конкретными играми. Игры классифицируются по категориям и принадлежат продавцам. Пользователи могут оставлять отзывы на игры.

На рисунке 1.2 представлена диаграмма сущность-связь в нотации Чена, отражающая структуру базы данных [11].

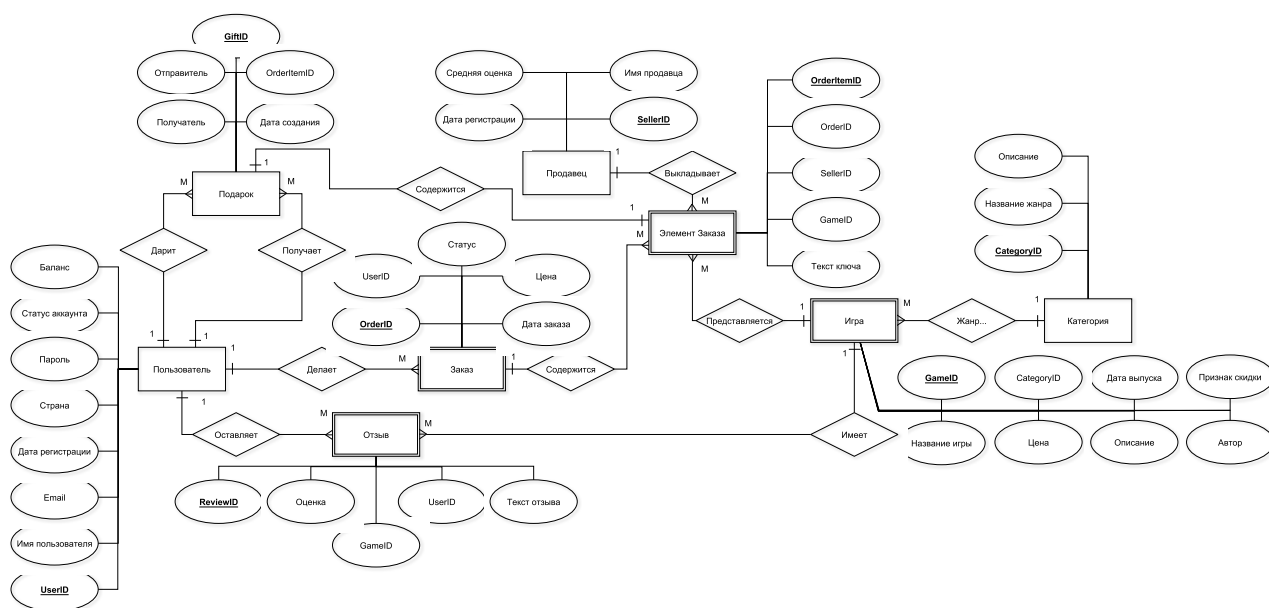


Рисунок 1.2 – Диаграмма сущность-связь в нотации Чена

1.6 Выбор модели базы данных

На основе анализа, проведенного в разделе 1.3.1, выбрана реляционная модель данных, так как она обладает следующими преимуществами, соответствующими требованиям системы:

- Данные хранятся в виде таблиц, что упрощает их структурирование и соответствует четкой структуре данных магазина видеоигр;
- Поддерживает целостность данных через ограничения, что важно для обеспечения согласованности информации о заказах и ключах;
- Позволяет выполнять сложные запросы с использованием SQL, что необходимо для реализации аналитики и фильтрации данных;
- Исключает дублирование данных за счет связей через внешние ключи, что повышает эффективность хранения.

Реляционная модель обеспечивает высокую производительность для *OLTP*-операций, таких как оформление заказов и добавление отзывов, что делает ее оптимальным выбором для данной системы.

ВЫВОД

В данном разделе проведен анализ предметной области интернет магазинов видеоигр. Анализ предметной области онлайн-магазинов видеоигр и существующих решений (*Steampay, Kupikod, Zaka — Zaka*) выявил недостатки: отсутствие гибкой передачи ключей и комбинированной системы отзывов. Это определило задачу разработки "Gamesbakery обеспечивающей покупку, продажу и передачу цифровых ключей с поддержкой отзывов (комментарии и звезды). Проведена классификация СУБД, определены роли пользователей, формализованы данные. Построены диаграммы прецедентов и сущность-связь в нотации Чена, отражающие структуру системы. На основе анализа выбрана реляционная модель данных, которая оптимальна для реализации поставленных задач.

2 Конструкторская часть

Конструкторская часть посвящена проектированию базы данных для системы Gamesbakery. На данном этапе определяются сущности системы в виде таблиц, их атрибуты, ограничения целостности, а также разрабатывается ролевая модель, хранимая процедура и триггер, обеспечивающие выполнение необходимых операций в рамках работы.

2.1 Формализация сущностей системы

На основе анализа предметной области, проведенного в предыдущем разделе, выделены ключевые сущности системы. Каждая сущность представлена в виде таблицы базы данных, содержащей поля с указанием их типов, описания и ограничений целостности. Ограничения описаны словесно в соответствии с SQL-стандартами.

2.1.1 Таблица Users

Таблица предназначена для хранения информации о пользователях системы (покупателях). Она включает следующие поля:

- **UserID**: тип **UUID**, уникальный идентификатор пользователя, является первичным ключом. Ограничение: **PRIMARY KEY**, значение по умолчанию — генерация нового **UUID**.
- **Name**: строковый тип, имя пользователя. Ограничение: **NOT NULL**.
- **Email**: строковый тип, адрес электронной почты пользователя. Ограничения: **NOT NULL**, **UNIQUE**.
- **RegistrationDate**: тип **дата**, дата регистрации пользователя. Ограничение: **NOT NULL**.
- **Country**: строковый тип, страна проживания пользователя. Ограничение: отсутствует.
- **Password**: строковый тип, хэш пароля пользователя. Ограничение: **NOT NULL**.

- **IsBlocked**: логический тип, статус пользователя: 0 — «не заблокирован», 1 — «заблокирован». Ограничение: NOT NULL, значение по умолчанию — 0 (не заблокирован).
- **Balance**: вещественное число, баланс пользователя для оплаты заказов. Ограничение: NOT NULL, значение по умолчанию — 0.00, CHECK (Balance >= 0).

2.1.2 Таблица Games

Таблица хранит информацию об играх, доступных для покупки. Поля таблицы:

- **GameID**: тип UUID, уникальный идентификатор игры, первичный ключ. Ограничение: PRIMARY KEY, значение по умолчанию — генерация нового UUID.
- **CategoryID**: тип UUID, идентификатор категории (жанра) игры. Ограничение: FOREIGN KEY (ссылается на CategoryID в таблице Categories), NOT NULL.
- **Title**: строковый тип, название игры. Ограничение: NOT NULL.
- **Price**: вещественное число, цена игры. Ограничения: NOT NULL, CHECK (Price >= 0).
- **ReleaseDate**: тип дата, дата выпуска игры. Ограничение: NOT NULL.
- **Description**: строковый тип, описание игры. Ограничение: отсутствует.
- **OriginalPublisher**: строковый тип, оригинальный издатель игры. Ограничение: NOT NULL.
- **IsForSale**: логический тип, статус продажи: 0 — «не продается», 1 — «продается». Ограничение: NOT NULL, значение по умолчанию — 1 (продается).

2.1.3 Таблица Categories

Таблица классифицирует игры по жанрам. Поля:

- **CategoryID**: тип UUID, уникальный идентификатор категории, первичный ключ. Ограничение: PRIMARY KEY, значение по умолчанию — гене-

рация нового UUID.

- **Name**: строковый тип, название жанра. Ограничение: NOT NULL.
- **Description**: строковый тип, описание жанра. Ограничение: отсутствует.

2.1.4 Таблица Sellers

Таблица содержит данные о продавцах, добавляющих игры в систему. Поля:

- **SellerID**: тип UUID, уникальный идентификатор продавца, первичный ключ. Ограничение: PRIMARY KEY, значение по умолчанию — генерация нового UUID.
- **Name**: строковый тип, название продавца. Ограничение: NOT NULL.
- **RegistrationDate**: тип дата, дата регистрации продавца. Ограничение: NOT NULL.
- **AverageRating**: вещественное число, средняя оценка продавца на основе отзывов. Ограничения: CHECK (AverageRating >= 0 AND AverageRating <= 5), значение по умолчанию — 0.00.

2.1.5 Таблица Orders

Таблица хранит информацию о заказах пользователей. Поля:

- **OrderID**: тип UUID, идентификатор заказа, первичный ключ. Ограничение: PRIMARY KEY, значение по умолчанию — генерация нового UUID.
- **UserID**: тип UUID, идентификатор пользователя, сделавшего заказ. Ограничение: FOREIGN KEY (ссылается на UserID в таблице Users), NOT NULL.
- **OrderDate**: тип дата и время, дата и время создания заказа. Ограничение: NOT NULL.
- **TotalPrice**: вещественное число, общая стоимость заказа. Ограничения: NOT NULL, CHECK (TotalPrice >= 0).
- **IsCompleted**: логический тип, статус заказа: 0 — «в обработке», 1 — «выполнен». Ограничение: NOT NULL, значение по умолчанию — 0 (в обработке).

- **IsOverdue**: логический тип, статус просрочки: 0 — «не просрочен», 1 — «просрочен». Ограничение: NOT NULL, значение по умолчанию — 0 (не просрочен).

2.1.6 Таблица OrderItems

Таблица связывает заказы с играми, указывая, какие игры были куплены в рамках заказа. Поля:

- **OrderItemID**: тип UUID, уникальный идентификатор элемента заказа, первичный ключ. Ограничение: PRIMARY KEY, значение по умолчанию — генерация нового UUID.
- **OrderID**: тип UUID, идентификатор заказа. Ограничение: FOREIGN KEY (ссылается на OrderID в таблице Orders), NOT NULL.
- **GameID**: тип UUID, идентификатор игры. Ограничение: FOREIGN KEY (ссылается на GameID в таблице Games), NOT NULL.
- **SellerID**: тип UUID, идентификатор продавца, который продает игру. Ограничение: FOREIGN KEY (ссылается на SellerID в таблице Sellers), NOT NULL.
- **KeyText**: строковый тип, текст цифрового ключа для активации игры. Ограничения: UNIQUE, может быть NULL (ключ может быть установлен позже).
- **IsGifted**: логический тип, статус подарка: 0 — «не подарен», 1 — «подарен». Ограничение: NOT NULL, значение по умолчанию — 0 (не подарен).

2.1.7 Таблица Reviews

Таблица хранит отзывы пользователей об играх. Поля:

- **ReviewID**: тип UUID, уникальный идентификатор отзыва, первичный ключ. Ограничение: PRIMARY KEY, значение по умолчанию — генерация нового UUID.
- **UserID**: тип UUID, идентификатор пользователя, оставившего отзыв. Ограничение: FOREIGN KEY (ссылается на UserID в таблице Users), NOT NULL.

- **GameID**: тип `UUID`, идентификатор игры, на которую оставлен отзыв. Ограничение: `FOREIGN KEY` (ссылается на `GameID` в таблице `Games`), `NOT NULL`.
- **Comment**: строковый тип, текст отзыва. Ограничение: `NOT NULL`.
- **StarRating**: целочисленный тип, оценка в звездах (от 1 до 5). Ограничения: `NOT NULL`, `CHECK (StarRating >= 1 AND StarRating <= 5)`.
- **CreationDate**: тип дата и время, дата создания отзыва. Ограничение: `NOT NULL`.

2.1.8 Таблица `ErrorLog`

Таблица предназначена для хранения логов ошибок, возникающих при выполнении триггеров и процедур. Поля:

- **ErrorID**: тип `UUID`, уникальный идентификатор записи лога, первичный ключ. Ограничение: `PRIMARY KEY`, значение по умолчанию — генерация нового `UUID`.
- **ErrorMessage**: строковый тип, сообщение об ошибке. Ограничение: `NOT NULL`.
- **ErrorDate**: тип дата и время, дата и время возникновения ошибки. Ограничение: `NOT NULL`, значение по умолчанию — текущая дата и время.

2.1.9 Таблица `Gifts`

Таблица управляет подарками между пользователями. Поля:

- **GiftID**: тип `UUID`, уникальный идентификатор подарка, первичный ключ. Ограничение: `PRIMARY KEY`, значение по умолчанию — генерация нового `UUID`.
- **SenderID**: тип `UUID`, идентификатор отправителя. Ограничение: `FOREIGN KEY` (ссылается на `UserID` в `Users`), `NOT NULL`.
- **RecipientID**: тип `UUID`, идентификатор получателя. Ограничение: `FOREIGN KEY` (ссылается на `UserID` в `Users`), `NOT NULL`.
- **OrderItemID**: тип `UUID`, идентификатор элемента заказа. Ограниче-

ние: FOREIGN KEY (ссылается на OrderItemID в OrderItems), NOT NULL.

- **GiftDate**: тип дата и время, дата отправки подарка. Ограничение: NOT NULL.

2.2 Ролевая модель

Для обеспечения безопасности и управления доступом к данным в системе Gamesbakery разработана ролевая модель. Она определяет права доступа для различных категорий пользователей, что позволяет ограничить доступ к данным в зависимости от их роли, минимизировать риски несанкционированного изменения информации и упростить администрирование системы.

2.2.1 Роль Guest (Гость)

Гость — это неавторизованный пользователь, который может только просматривать общедоступную информацию. Права:

- SELECT — над таблицей **Games** (просмотр каталога).
- SELECT — над таблицей **Categories** (просмотр жанров).

Гость не имеет доступа к данным о заказах, отзывах и пользователях, что обеспечивает защиту конфиденциальной информации.

2.2.2 Роль User (Покупатель)

Покупатель — зарегистрированный пользователь, который может оформлять заказы и оставлять отзывы. Права:

- SELECT — над таблицами **Games**, **Categories**, **Sellers** (просмотр данных).
- SELECT, INSERT — над таблицей **Orders** (просмотр и создание заказов).
- SELECT — над таблицей **OrderItems** (просмотр элементов заказов).
- SELECT, INSERT — над таблицей **Reviews** (просмотр и добавление отзывов).

- SELECT, UPDATE — над таблицей **Users** (просмотр и редактирование профиля).

Покупатель также имеет доступ через представления:

- SELECT — над **UserOrders, UserOrderItems, UserReviews, UserProfile, UserSentGifts, UserReceivedGifts, SellerOrderItems**.

2.2.3 Роль Seller (Продавец)

Продавец добавляет игры в каталог и управляет ключами для своих заказов. Права:

- SELECT, INSERT — над таблицей **Games** (просмотр и добавление игр).
- SELECT — над таблицами **Categories, Orders, Users** (просмотр данных).
- SELECT, INSERT — над таблицей **OrderItems** (просмотр и добавление элементов).

Продавец также имеет доступ через представления:

- SELECT, UPDATE — над **SellerOrderItems, SellerProfile**.

2.2.4 Роль Admin (Администратор)

Администратор обладает полным доступом ко всем данным. Права:

- Все права (SELECT, INSERT, UPDATE, DELETE) — над таблицами **Users, Games, Categories, Sellers, Orders, OrderItems, Reviews, ErrorLog, Gifts**.

Администратор может управлять всеми аспектами системы, включая блокировку пользователей, удаление отзывов и изменение данных об играх.

2.3 Разработка хранимой процедуры

Для управления SQL-логинами и пользователями базы данных разработана процедура **sp_ManageSqlLogin**. Она используется в триггере для автоматизации создания и удаления SQL-логинов.

Схема алгоритма представлена на рисунке 2.1.

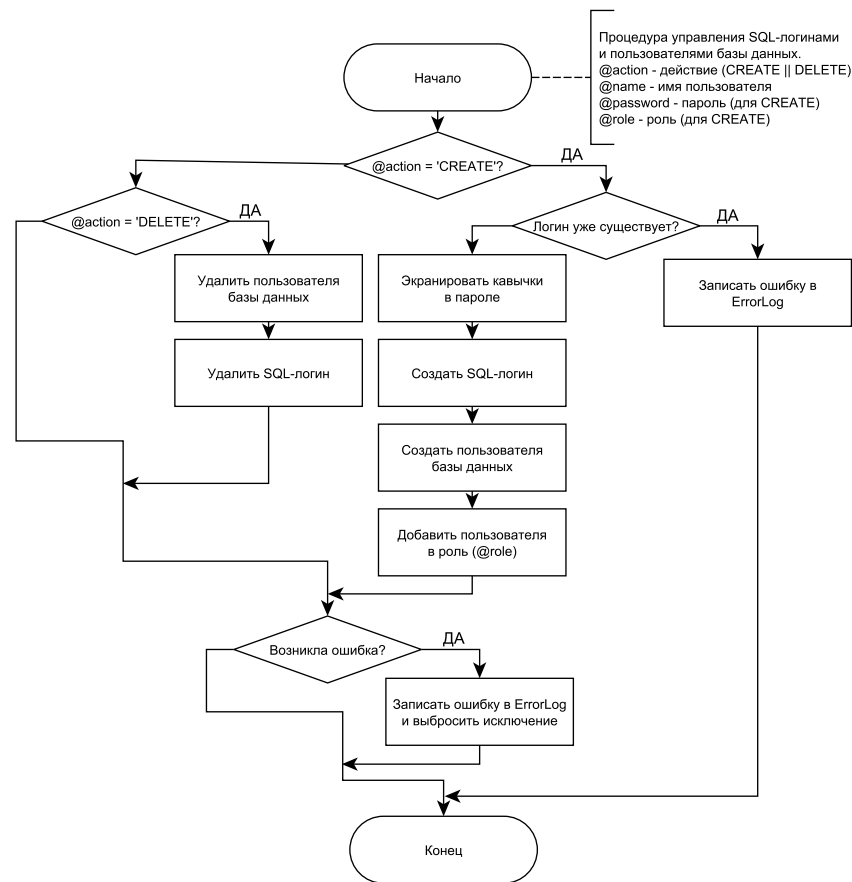


Рисунок 2.1 – Схема алгоритма работы процедуры sp_ManageSqlLogin

2.3.1 Описание процедуры

Процедура принимает следующие параметры:

@Action (NVARCHAR(10)) — действие: **CREATE** или **DELETE**.

@Name (NVARCHAR(100)) — имя пользователя.

@Password (NVARCHAR(100)) — пароль (для **CREATE**).

@Role (NVARCHAR(50)) — роль (для **CREATE**).

Действия процедуры:

1. Если @Action = 'CREATE': проверяет, существует ли логин. Если логин существует, записывает ошибку в таблицу **ErrorLog** и завершает выполнение. Иначе экранирует кавычки в пароле, создает SQL-логин, пользователя базы данных и добавляет пользователя в указанную роль.

2. Если @Action = 'DELETE': удаляет пользователя базы данных. Удаляет SQL-логин.
3. Обрабатывает ошибки, логируя их в таблицу ErrorLog.

2.4 Разработка триггера для таблицы Users

Для управления SQL-логинами при добавлении и удалении пользователей разработан триггер `trg_Users_AfterInsertDelete`. Триггер использует процедуру `sp_ManageSqlLogin` для автоматизации создания и удаления SQL-логинов.

Схема алгоритма представлена на рисунке 2.2.

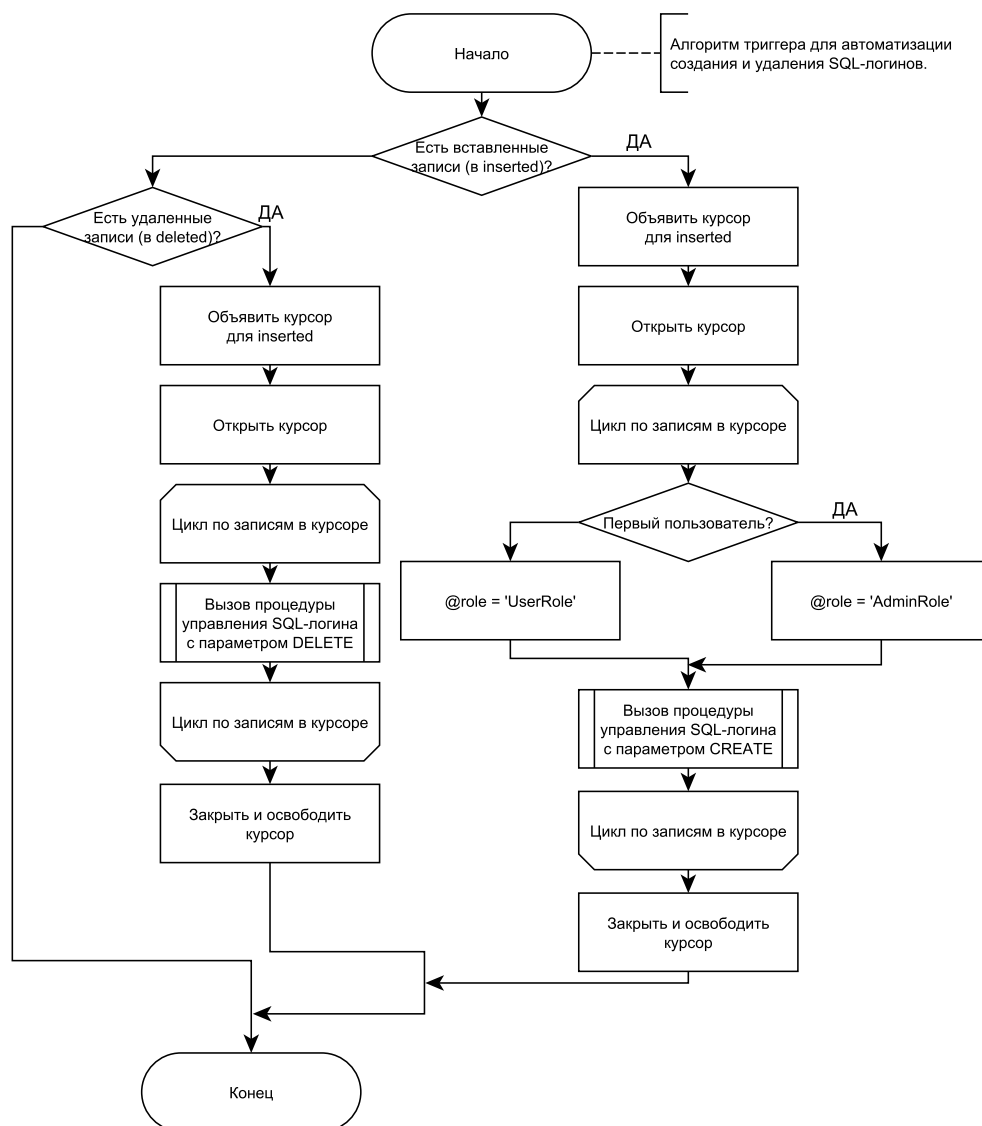


Рисунок 2.2 – Схема алгоритма триггера `trg_Users_AfterInsertDelete`

2.4.1 Описание триггера

Триггер срабатывает на события `INSERT` и `DELETE` в таблице `Users`. Действия триггера:

1. При добавлении записи (`INSERT`): проверяет, является ли пользователь первым в системе (если да, назначает роль `AdminRole`, иначе `UserRole`). Для каждой вставленной записи вызывает `sp_ManageSqlLogin` с действием `CREATE`.
2. При удалении записи (`DELETE`): Для каждой удаленной записи вызывает `sp_ManageSqlLogin` с действием `DELETE`.
3. Использует курсоры для обработки нескольких записей.

2.5 Разработка функций

Для расширения функциональности магазина игр разработаны функции, предоставляющие пользователям полезную информацию.

2.5.1 Функция `fn_GetUserTotalSpent`

Функция вычисляет общую сумму, потраченную пользователем на заказы.

Параметры: `@UserID` (`UNIQUEIDENTIFIER`) — идентификатор пользователя.

Тип возвращаемого значения: `DECIMAL(12,2)` — общая сумма, потраченная пользователем.

Действия: Функция суммирует значения поля `TotalPrice` из таблицы `Orders` для заказов, выполненных пользователем с указанным `UserID`, и возвращает результат. Если заказов нет, возвращается 0.00.

Схема алгоритма представлена на рисунке 2.3.

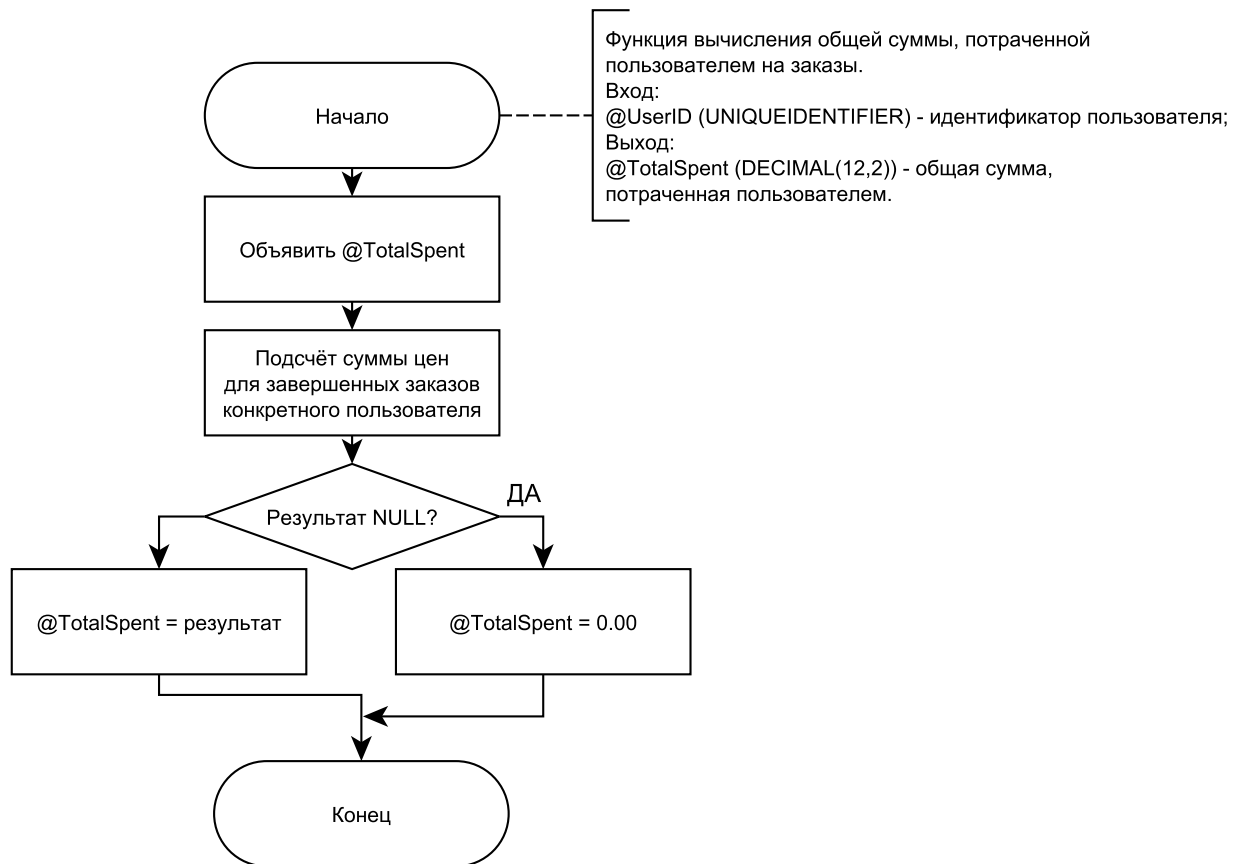


Рисунок 2.3 – Схема алгоритма функции fn_GetUserTotalSpent

2.5.2 Функция fn_GetGameAverageRating

Функция определяет среднюю оценку игры на основе отзывов пользователей.

Параметры: @GameID (UNIQUEIDENTIFIER) — идентификатор игры.

Тип возвращаемого значения: DECIMAL(3,2) — средняя оценка игры (от 0.00 до 5.00).

Действия: Функция вычисляет среднее значение поля **StarRating** из таблицы **Reviews** для указанной игры и возвращает результат. Если отзывов нет, возвращается 0.00.

Схема алгоритма представлена на рисунке 2.4.

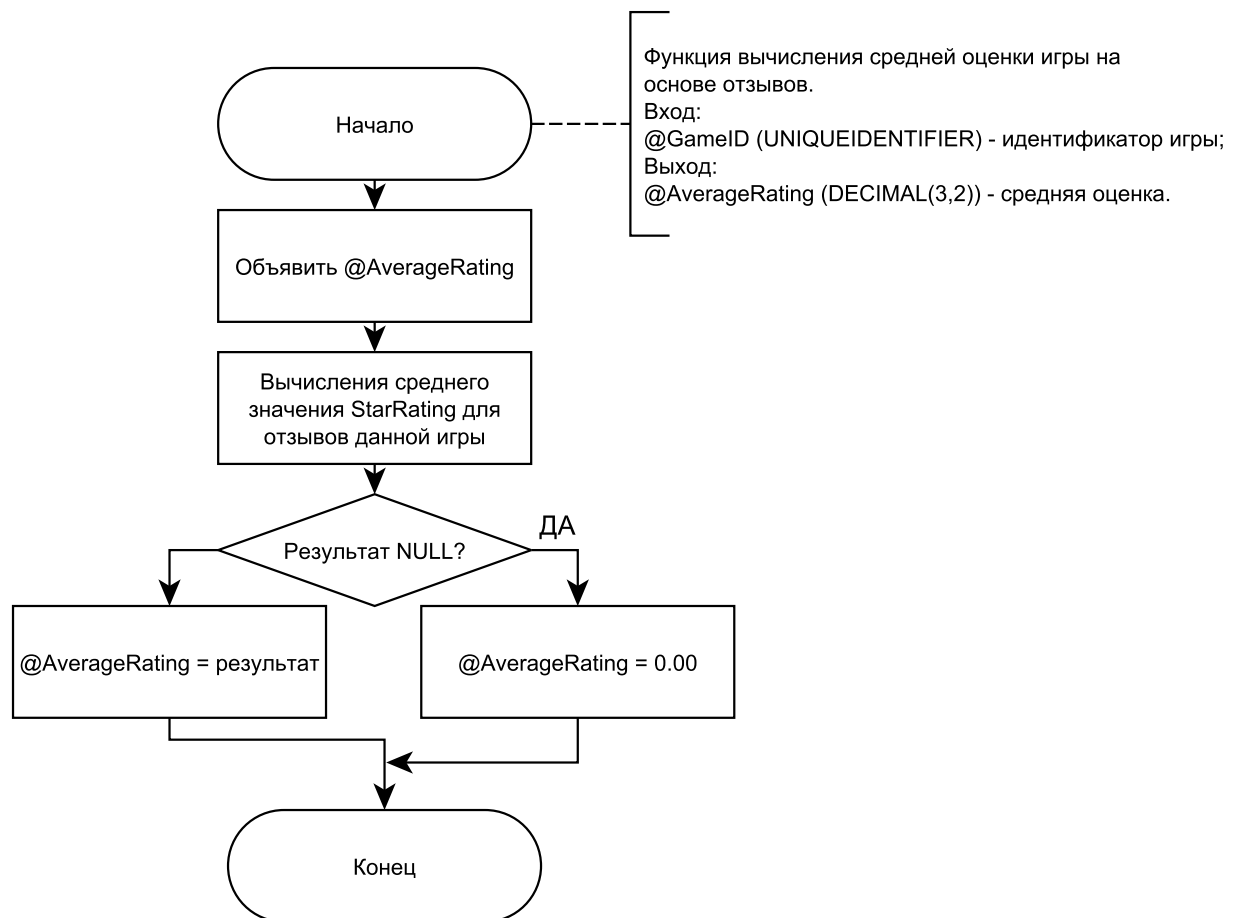


Рисунок 2.4 – Схема алгоритма функции fn_GetGameAverageRating

ВЫВОД

В данном разделе были определены ключевые сущности магазина видеоигр в виде таблиц с указанием их атрибутов, типов данных и ограничений целостности. Разработана ролевая модель, обеспечивающая управление доступом для ролей **Guest**, **User**, **Seller** и **Admin**. Создана хранимая процедура **sp_ManageSqlLogin** для автоматизации управления SQL-логинами и триггер **trg_Users_AfterInsertDelete** для синхронизации создания и удаления логинов с изменениями в таблице **Users**. Также разработаны функции **fn_GetUserTotalSpent** и **fn_GetGameAverageRating**, которые предоставляют аналитические данные о расходах пользователей и рейтингах игр.

3 Технологическая часть

В данном разделе представлена реализация базы данных. Описана структура базы данных, включая SQL-скрипты для создания таблиц с наложением ограничений целостности. Реализация выполнена с использованием реляционной модели данных и СУБД Microsoft SQL Server.

3.1 Выбор СУБД

Выбор СУБД проводился среди SQLite, MySQL, MSSQL и PostgreSQL. Критериями для сравнения были выбраны возможность создания ролевой модели (РМ), интеграция с технологиями Microsoft (И) и также стоимость лицензии (СЛ). В таблице 3.1 представлены результаты сравнения СУБД по выбранным критериям.

Таблица 3.1 – Сравнение СУБД

СУБД	РМ	И	СЛ
SQLite	-	-	+
MySQL	+	-	+
MSSQL	+	+	-
PostgreSQL	+	-	+

В рамках данной работы в качестве СУБД была выбрана Microsoft SQL Server (*MSSQL*). Основная причина выбора *MSSQL* — интеграция с Microsoft-технологиями, поскольку веб-приложение в рамках работы разработано с использованием *ASP.NET CORE* [12] с поддержкой *EF Core*, что позволяет использовать объектно-реляционное отображение (*ORM*) для работы с базой данных. Учитывая, что в рамках курсовой работы не требуется коммерческая лицензия, используется бесплатная учебная версия *MSSQL*, тогда как *PostgreSQL* остается альтернативой благодаря своей бесплатности и популярности в России.

3.2 Средства реализации

Для реализации программного обеспечения системы Gamesbakery был выбран язык программирования *C#*. Данный выбор обусловлен его строгой типизацией, поддержкой всех необходимых типов данных, определенных на этапе проектирования, и интеграцией с Microsoft-технологиями (*ASP.NET*), что упрощает разработку веб-приложений.

Для реализации серверной части использовался фреймворк *ASP.NET*. Этот фреймворк обеспечивает высокую производительность, масштабируемость и кроссплатформенность, что делает его подходящим для создания веб-приложения онлайн-магазина. *ASP.NET* поддерживает архитектуру MVC (Model-View-Controller), что позволило структурировать код для обработки запросов, управления бизнес-логикой и взаимодействия с базой данных.

Для доступа к данным использовалась библиотека *Entity Framework Core*. Она предоставляет *ORM*, позволяя выполнять запросы к базе данных MSSQL как на уровне SQL, так и на уровне объектной модели *C#*. Это упростило реализацию операций с данными, таких как создание заказов, управление отзывами и обновление пользовательских профилей.

Для реализации пользовательского интерфейса использовалась технология *Razor Pages* в составе *ASP.NET* в сочетании с библиотекой *Bootstrap*.

3.3 Создание таблиц

Ниже приведены SQL-скрипты для создания таблиц базы данных с наложением всех ограничений целостности. Каждая таблица создается с последующим добавлением ограничений через команды **ALTER TABLE**, что соответствует предоставленной структуре.

Листинг 3.1 – Создание таблицы Users

```
1 CREATE TABLE Users (  
2     UserID UNIQUEIDENTIFIER ,  
3     Name NVARCHAR(50) ,  
4     Email NVARCHAR(100) ,  
5     RegistrationDate DATE ,  
6     Country NVARCHAR(300) ,
```

```

7      Password NVARCHAR(100) ,
8      IsBlocked BIT ,
9      Balance DECIMAL(10,2)
10 );
11 GO
12
13 ALTER TABLE Users
14     ALTER COLUMN UserID UNIQUEIDENTIFIER NOT NULL;
15     ADD CONSTRAINT PK_Users PRIMARY KEY (UserID);
16     ALTER COLUMN Name NVARCHAR(50) NOT NULL;
17     ADD CONSTRAINT UQ_Users_Name UNIQUE (Name);
18     ALTER COLUMN Email NVARCHAR(100) NOT NULL;
19     ALTER COLUMN RegistrationDate DATE NOT NULL;
20     ALTER COLUMN Password NVARCHAR(100) NOT NULL;
21     ALTER COLUMN IsBlocked BIT NOT NULL;
22     ADD CONSTRAINT DF_Users_IsBlocked DEFAULT 0 FOR IsBlocked;
23     ALTER COLUMN Balance DECIMAL(10,2) NOT NULL;
24     ADD CONSTRAINT DF_Users_Balance DEFAULT 0.00 FOR Balance;
25     ADD CONSTRAINT CHK_Users_Balance CHECK (Balance >= 0);
26 GO

```

Листинг 3.2 – Создание таблицы Sellers

```

1 CREATE TABLE Sellers (
2     SellerID UNIQUEIDENTIFIER,
3     Name NVARCHAR(100),
4     RegistrationDate DATE,
5     AverageRating DECIMAL(3,2),
6     Password NVARCHAR(100)
7 );
8 GO
9
10 ALTER TABLE Sellers
11     ALTER COLUMN SellerID UNIQUEIDENTIFIER NOT NULL;
12     ADD CONSTRAINT PK_Sellers PRIMARY KEY (SellerID);
13     ALTER COLUMN Name NVARCHAR(100) NOT NULL;
14     ADD CONSTRAINT UQ_Sellers_Name UNIQUE (Name);
15     ALTER COLUMN RegistrationDate DATE NOT NULL;
16     ALTER COLUMN Password NVARCHAR(100) NOT NULL;
17     ADD CONSTRAINT DF_Sellers_AverageRating DEFAULT 0.00 FOR
        AverageRating;
18     ADD CONSTRAINT CHK_Sellers_AverageRating CHECK (AverageRating

```

```
19    >= 0 AND AverageRating <= 5);  
GO
```

Листинг 3.3 – Создание таблицы Categories

```
1 CREATE TABLE Categories (  
2     CategoryID UNIQUEIDENTIFIER,  
3     Name NVARCHAR(50),  
4     Description NVARCHAR(255)  
5 );  
6 GO  
7  
8 ALTER TABLE Categories  
9     ALTER COLUMN CategoryID UNIQUEIDENTIFIER NOT NULL;  
10    ADD CONSTRAINT PK_Categories PRIMARY KEY (CategoryID);  
11    ALTER COLUMN Name NVARCHAR(50) NOT NULL;  
12 GO
```

Листинг 3.4 – Создание таблицы Games

```
1 CREATE TABLE Games (  
2     GameID UNIQUEIDENTIFIER,  
3     CategoryID UNIQUEIDENTIFIER,  
4     Title NVARCHAR(100),  
5     Price DECIMAL(10,2),  
6     ReleaseDate DATE,  
7     Description NVARCHAR(MAX),  
8     OriginalPublisher NVARCHAR(100),  
9     IsForSale BIT  
10 );  
11 GO  
12  
13 ALTER TABLE Games  
14     ALTER COLUMN GameID UNIQUEIDENTIFIER NOT NULL;  
15     ADD CONSTRAINT PK_Games PRIMARY KEY (GameID);  
16     ALTER COLUMN CategoryID UNIQUEIDENTIFIER NOT NULL;  
17     ADD CONSTRAINT FK_Games_Categories FOREIGN KEY (CategoryID)  
18     REFERENCES Categories(CategoryID) ON DELETE CASCADE;  
19     ALTER COLUMN Title NVARCHAR(100) NOT NULL;  
20     ALTER COLUMN Price DECIMAL(10,2) NOT NULL;  
21     ADD CONSTRAINT CHK_Games_Price CHECK (Price >= 0);  
22     ALTER COLUMN ReleaseDate DATE NOT NULL;  
23     ALTER COLUMN OriginalPublisher NVARCHAR(100) NOT NULL;
```



```

24 ALTER COLUMN IsForSale BIT NOT NULL;
25 ADD CONSTRAINT DF_Games_IsForSale DEFAULT 1 FOR IsForSale;
26 GO

```

Листинг 3.5 – Создание таблицы Orders

```

1 CREATE TABLE Orders (
2     OrderID UNIQUEIDENTIFIER,
3     UserID UNIQUEIDENTIFIER,
4     OrderDate DATETIME,
5     TotalPrice DECIMAL(10,2),
6     IsCompleted BIT,
7     IsOverdue BIT
8 );
9 GO
10
11 ALTER TABLE Orders
12     ALTER COLUMN OrderID UNIQUEIDENTIFIER NOT NULL;
13     ADD CONSTRAINT PK_Orders PRIMARY KEY (OrderID);
14     ALTER COLUMN UserID UNIQUEIDENTIFIER NOT NULL;
15     ADD CONSTRAINT FK_Orders_Users FOREIGN KEY (UserID)
16     REFERENCES Users(UserID) ON DELETE CASCADE;
17     ALTER COLUMN OrderDate DATETIME NOT NULL;
18     ALTER COLUMN TotalPrice DECIMAL(10,2) NOT NULL;
19     ADD CONSTRAINT CHK_Orders_TotalPrice CHECK (TotalPrice >= 0);
20     ALTER COLUMN IsCompleted BIT NOT NULL;
21     ADD CONSTRAINT DF_Orders_IsCompleted DEFAULT 0 FOR
        IsCompleted;
22     ALTER COLUMN IsOverdue BIT NOT NULL;
23     ADD CONSTRAINT DF_Orders_IsOverdue DEFAULT 0 FOR IsOverdue;
24 GO

```

Листинг 3.6 – Создание таблицы OrderItems

```

1 CREATE TABLE OrderItems (
2     OrderItemID UNIQUEIDENTIFIER,
3     OrderID UNIQUEIDENTIFIER,
4     GameID UNIQUEIDENTIFIER,
5     SellerID UNIQUEIDENTIFIER,
6     KeyText NVARCHAR(50)
7 );
8 GO
9

```

```

10 ALTER TABLE OrderItems
11     ALTER COLUMN OrderItemID UNIQUEIDENTIFIER NOT NULL;
12     ADD CONSTRAINT PK_OrderItems PRIMARY KEY (OrderItemID);
13     ALTER COLUMN OrderID UNIQUEIDENTIFIER NOT NULL;
14     ADD CONSTRAINT FK_OrderItems_Orders FOREIGN KEY (OrderID)
15     REFERENCES Orders(OrderID) ON DELETE CASCADE;
16     ALTER COLUMN GameID UNIQUEIDENTIFIER NOT NULL;
17     ADD CONSTRAINT FK_OrderItems_Games FOREIGN KEY (GameID)
18     REFERENCES Games(GameID) ON DELETE CASCADE;
19     ALTER COLUMN SellerID UNIQUEIDENTIFIER NOT NULL;
20     ADD CONSTRAINT FK_OrderItems_Sellers FOREIGN KEY (SellerID)
21     REFERENCES Sellers(SellerID) ON DELETE CASCADE;
22 GO
23
24 CREATE UNIQUE NONCLUSTERED INDEX UQ_OrderItems_KeyText
25     ON dbo.OrderItems (KeyText)
26     WHERE KeyText IS NOT NULL;
27 GO

```

Листинг 3.7 – Создание таблицы Reviews

```

1 CREATE TABLE Reviews (
2     ReviewID UNIQUEIDENTIFIER,
3     UserID UNIQUEIDENTIFIER,
4     GameID UNIQUEIDENTIFIER,
5     Comment NVARCHAR(MAX),
6     StarRating INT,
7     CreationDate DATETIME
8 );
9 GO
10
11 ALTER TABLE Reviews
12     ALTER COLUMN ReviewID UNIQUEIDENTIFIER NOT NULL;
13     ADD CONSTRAINT PK_Reviews PRIMARY KEY (ReviewID);
14     ALTER COLUMN UserID UNIQUEIDENTIFIER NOT NULL;
15     ADD CONSTRAINT FK_Reviews_Users FOREIGN KEY (UserID)
16     REFERENCES Users(UserID) ON DELETE CASCADE;
17     ALTER COLUMN GameID UNIQUEIDENTIFIER NOT NULL;
18     ADD CONSTRAINT FK_Reviews_Games FOREIGN KEY (GameID)
19     REFERENCES Games(GameID) ON DELETE CASCADE;
20     ALTER COLUMN Comment NVARCHAR(MAX) NOT NULL;
21     ALTER COLUMN StarRating INT NOT NULL;

```

```

22      ADD CONSTRAINT CHK_Reviews_StarRating CHECK (StarRating >= 1
          AND StarRating <= 5);
23      ALTER COLUMN CreationDate DATETIME NOT NULL;
24 GO

```

Листинг 3.8 – Создание таблицы Gifts

```

1 CREATE TABLE Gifts (
2     GiftID UNIQUEIDENTIFIER,
3     SenderID UNIQUEIDENTIFIER,
4     RecipientID UNIQUEIDENTIFIER,
5     OrderItemID UNIQUEIDENTIFIER,
6     GiftDate DATETIME
7 );
8 GO
9
10 ALTER TABLE Gifts
11     ALTER COLUMN SenderID UNIQUEIDENTIFIER NOT NULL;
12     ALTER COLUMN RecipientID UNIQUEIDENTIFIER NOT NULL;
13     ALTER COLUMN OrderItemID UNIQUEIDENTIFIER NOT NULL;
14     ALTER COLUMN GiftDate DATETIME NOT NULL;
15     ADD CONSTRAINT PK_Gifts PRIMARY KEY (GiftID);
16     ADD CONSTRAINT FK_Gifts_Sender FOREIGN KEY (SenderID)
          REFERENCES Users(UserID);
17     ADD CONSTRAINT FK_Gifts_Recipient FOREIGN KEY (RecipientID)
          REFERENCES Users(UserID);
18     ADD CONSTRAINT FK_Gifts_OrderItem FOREIGN KEY (OrderItemID)
          REFERENCES OrderItems(OrderItemID);
19 GO

```

3.4 Индексы

В таблице **OrderItems** создан уникальный некластеризованный индекс на столбец **KeyText** для обеспечения уникальности ключей активации, если они заданы.

3.5 Функции

Функция `fn_GetUserTotalSpent` вычисляет общую сумму, потраченную пользователем на заказы, а функция `fn_GetGameAverageRating` определяет среднюю оценку игры на основе отзывов пользователей.

```
1 CREATE FUNCTION fn_GetUserTotalSpent (@UserID UNIQUEIDENTIFIER)
2 RETURNS DECIMAL(12,2)
3 AS
4 BEGIN
5     DECLARE @TotalSpent DECIMAL(12,2);
6     SELECT @TotalSpent = SUM(o.TotalPrice)
7     FROM Orders o
8     WHERE o.UserID = @UserID AND o.IsCompleted = 1;
9     RETURN ISNULL(@TotalSpent, 0.00);
10 END;
11 GO
12
13 CREATE FUNCTION fn_GetGameAverageRating (@GameID UNIQUEIDENTIFIER)
14 RETURNS DECIMAL(3,2)
15 AS
16 BEGIN
17     DECLARE @AverageRating DECIMAL(3,2);
18     SELECT @AverageRating = AVG(CAST(r.StarRating AS
19     DECIMAL(3,2)))
19     FROM Reviews r
20     WHERE r.GameID = @GameID;
21     RETURN ISNULL(@AverageRating, 0.00);
22 END;
23 GO
```

3.6 Хранимые процедуры и триггеры

В системе применяются хранимые процедуры и триггеры для автоматизации задач, обеспечения целостности данных и ускорения обработки транзакций.

3.6.1 Хранимая процедура sp_ManageSqlLogin

Процедура sp_ManageSqlLogin управляет созданием и удалением SQL-логинов для пользователей и продавцов, обеспечивая безопасность данных и упрощая администрирование.

```
1 CREATE PROCEDURE sp_ManageSqlLogin
2     @Action NVARCHAR(10), — 'CREATE' or 'DELETE'
3     @Name NVARCHAR(100),
4     @Password NVARCHAR(100) = NULL,
5     @Role NVARCHAR(50) = NULL
6 AS
7 BEGIN
8     SET NOCOUNT ON;
9     BEGIN TRY
10         IF @Action = 'CREATE'
11             BEGIN
12                 IF EXISTS (SELECT * FROM sys.server_principals WHERE
13                     name = @Name)
14                     BEGIN
15                         INSERT INTO ErrorLog (ErrorMessage)
16                         VALUES ('SQL login already exists for Name: ' +
17                             @Name);
18                         RETURN;
19                     END
20                 SET @Password = REPLACE(@Password, '', ' ');
21                 EXEC('CREATE LOGIN [' + @Name + '] WITH PASSWORD =
22                     '' + @Password + '', CHECK_POLICY = OFF;');
23                 EXEC('CREATE USER [' + @Name + '] FOR LOGIN [' +
24                     @Name + '];');
25                 EXEC('ALTER ROLE ' + @Role + ' ADD MEMBER [' + @Name
26                     + '];');
27             END
28         ELSE IF @Action = 'DELETE'
29             BEGIN
30                 EXEC('IF EXISTS (SELECT * FROM
31                     sys.database_principals WHERE name = '' + @Name +
32                     '') DROP USER [' + @Name + '];');
33                 EXEC('IF EXISTS (SELECT * FROM sys.server_principals
34                     WHERE name = '' + @Name + '') DROP LOGIN [' +
35                     @Name + '];');
```

```

27         END
28     END TRY
29     BEGIN CATCH
30         INSERT INTO ErrorLog (ErrorMessage)
31         VALUES ( 'Error in sp_ManageSqlLogin for ' + @Name + ': '
32                 + ERROR_MESSAGE());
33     THROW;
34 END CATCH
35 END;
36 GO

```

3.6.2 Триггер trg_Users_AfterInsertDelete

Триггер `trg_Users_AfterInsertDelete` срабатывает после вставки или удаления записей в таблице `Users` и выполняет создание или удаление SQL-ЛОГИНОВ.

Листинг 3.9 – Триггер `trg_Users_AfterInsertDelete`

```

1 CREATE OR ALTER TRIGGER trg_Users_AfterInsertDelete
2 ON Users
3 AFTER INSERT, DELETE
4 AS
5 BEGIN
6     SET NOCOUNT ON;
7     IF EXISTS (SELECT * FROM inserted)
8     BEGIN
9         DECLARE @Name NVARCHAR(50);
10        DECLARE @Password NVARCHAR(100);
11        DECLARE @UserID UNIQUEIDENTIFIER;
12        DECLARE @Role NVARCHAR(50);
13        DECLARE user_cursor CURSOR FOR
14        SELECT Name, Password, UserID
15        FROM inserted;
16        OPEN user_cursor;
17        FETCH NEXT FROM user_cursor INTO @Name, @Password,
18            @UserID;
19        WHILE @@FETCH_STATUS = 0
20        BEGIN
21            SET @Role = 'UserRole';

```

```

21         BEGIN TRY
22             EXEC sp_ManageSqlLogin
23                 @Action = 'CREATE',
24                 @Name = @Name,
25                 @Password = 'UserPass123',
26                 @Role = @Role;
27         END TRY
28         BEGIN CATCH
29             INSERT INTO ErrorLog (ErrorMessage)
30             VALUES ('Error creating SQL user for ' + @Name +
31                 ': ' + ERROR_MESSAGE());
32         END CATCH
33         FETCH NEXT FROM user_cursor INTO @Name, @Password,
34             @UserID;
35     END
36     CLOSE user_cursor;
37     DEALLOCATE user_cursor;
38 END
39
40 IF EXISTS (SELECT * FROM deleted)
41 BEGIN
42     DECLARE @NameToDelete NVARCHAR(50);
43     DECLARE delete_cursor CURSOR FOR
44     SELECT Name
45     FROM deleted;
46     OPEN delete_cursor;
47     FETCH NEXT FROM delete_cursor INTO @NameToDelete;
48     WHILE @@FETCH_STATUS = 0
49     BEGIN
50         BEGIN TRY
51             EXEC sp_ManageSqlLogin
52                 @Action = 'DELETE',
53                 @Name = @NameToDelete;
54         END TRY
55         BEGIN CATCH
56             INSERT INTO ErrorLog (ErrorMessage)
57             VALUES ('Error dropping SQL user ' +
58                 @NameToDelete + ': ' + ERROR_MESSAGE());
59         END CATCH
60         FETCH NEXT FROM delete_cursor INTO @NameToDelete;
61     END
62 END

```

```

59         CLOSE delete_cursor;
60         DEALLOCATE delete_cursor;
61     END
62 END;
63 GO

```

3.6.3 Триггер trg_Sellers_AfterInsertDelete

Триггер `trg_Sellers_AfterInsertDelete` срабатывает после вставки или удаления записей в таблице `Sellers` и выполняет создание или удаление SQL-логинов для продавцов.

Листинг 3.10 – Триггер `trg_Sellers_AfterInsertDelete`

```

1 CREATE OR ALTER TRIGGER trg_Sellers_AfterInsertDelete
2 ON Sellers
3 AFTER INSERT, DELETE
4 AS
5 BEGIN
6     SET NOCOUNT ON;
7     IF EXISTS (SELECT * FROM inserted)
8     BEGIN
9         DECLARE @Name NVARCHAR(100);
10        DECLARE @Password NVARCHAR(100);
11        DECLARE seller_cursor CURSOR FOR
12        SELECT Name, Password
13        FROM inserted;
14        OPEN seller_cursor;
15        FETCH NEXT FROM seller_cursor INTO @Name, @Password;
16        WHILE @@FETCH_STATUS = 0
17        BEGIN
18            BEGIN TRY
19                EXEC sp_ManageSqlLogin
20                    @Action = 'CREATE',
21                    @Name = @Name,
22                    @Password = 'SellerPass123',
23                    @Role = 'SellerRole';
24            END TRY
25            BEGIN CATCH
26                INSERT INTO ErrorLog (ErrorMessage)

```



```

27             VALUES ('Error creating SQL user for ' + @Name +
28                 ': ' + ERROR_MESSAGE());
29         END CATCH
30         FETCH NEXT FROM seller_cursor INTO @Name, @Password;
31     END
32     CLOSE seller_cursor;
33     DEALLOCATE seller_cursor;
34 END
35 IF EXISTS (SELECT * FROM deleted)
36 BEGIN
37     DECLARE @NameToDelete NVARCHAR(100);
38     DECLARE delete_cursor CURSOR FOR
39     SELECT Name
40     FROM deleted;
41     OPEN delete_cursor;
42     FETCH NEXT FROM delete_cursor INTO @NameToDelete;
43     WHILE @@FETCH_STATUS = 0
44     BEGIN
45         BEGIN TRY
46             EXEC sp_ManageSqlLogin
47                 @Action = 'DELETE',
48                 @Name = @NameToDelete;
49         END TRY
50         BEGIN CATCH
51             INSERT INTO ErrorLog (ErrorMessage)
52             VALUES ('Error dropping SQL user ' +
53                 @NameToDelete + ': ' + ERROR_MESSAGE());
54         END CATCH
55         FETCH NEXT FROM delete_cursor INTO @NameToDelete;
56     END
57     CLOSE delete_cursor;
58     DEALLOCATE delete_cursor;
59 END
60 GO

```

3.7 Модель ролей и прав доступа

В системе определены роли: GuestRole, UserRole, SellerRole, AdminRole.

Для реализации ограничений доступа используются представления , которые позволяют пользователям видеть и редактировать только свои данные.

Листинг 3.11 – Создание ролей

```
1 USE Gamesbakery;  
2 GO  
3 CREATE ROLE GuestRole;  
4 CREATE ROLE UserRole;  
5 CREATE ROLE SellerRole;  
6 CREATE ROLE AdminRole;  
7 GO
```

Листинг 3.12 – Настройка прав для GuestRole и сертификата для процедур

```
1 GRANT SELECT ON Games TO GuestRole;  
2 GRANT SELECT ON Categories TO GuestRole;  
3 GRANT SELECT ON Reviews TO GuestRole;  
4 GO  
5 CREATE CERTIFICATE RegistrationCert  
6     ENCRYPTION BY PASSWORD = 'INSERTCert'  
7     WITH SUBJECT = 'Certificate for Registration Procedures',  
8     EXPIRY_DATE = '20261231';  
9 GO  
10 CREATE USER RegistrationCertUser FROM CERTIFICATE  
    RegistrationCert;  
11 GO  
12 GRANT INSERT ON Users TO RegistrationCertUser;  
13 GRANT INSERT ON Sellers TO RegistrationCertUser;  
14 GO  
15 ADD SIGNATURE TO sp_RegisterUser  
16     BY CERTIFICATE RegistrationCert  
17     WITH PASSWORD = 'INSERTCert';  
18 GO  
19 ADD SIGNATURE TO sp_RegisterSeller  
20     BY CERTIFICATE RegistrationCert  
21     WITH PASSWORD = 'INSERTCert';  
22 GO  
23 ADD SIGNATURE TO sp_ManageSqlLogin  
24     BY CERTIFICATE RegistrationCert  
25     WITH PASSWORD = 'INSERTCert';  
26 GO
```

```

27 GRANT EXECUTE ON sp_ManageSqlLogin TO GuestRole;
28 GRANT EXECUTE ON sp_RegisterUser TO GuestRole;
29 GRANT EXECUTE ON sp_RegisterSeller TO GuestRole;
30 GO
31 REVOKE INSERT ON Users TO GuestRole;
32 REVOKE INSERT ON Sellers TO GuestRole;
33 GO

```

Листинг 3.13 – Создание представлений для ограничения доступа

```

1 CREATE OR ALTER VIEW UserOrders AS
2 SELECT o.*
3 FROM Orders o
4 JOIN Users u ON o.UserID = u.UserID
5 WHERE u.Name = CURRENT_USER;
6 GO
7 CREATE OR ALTER VIEW UserOrderItems AS
8 SELECT oi.*
9 FROM OrderItems oi
10 JOIN Orders o ON oi.OrderID = o.OrderID
11 JOIN Users u ON o.UserID = u.UserID
12 WHERE u.Name = CURRENT_USER;
13 GO
14 CREATE OR ALTER VIEW UserReviews AS
15 SELECT r.*
16 FROM Reviews r
17 JOIN Users u ON r.UserID = u.UserID
18 WHERE u.Name = CURRENT_USER;
19 GO
20 CREATE OR ALTER VIEW UserProfile AS
21 SELECT u.*
22 FROM Users u
23 WHERE u.Name = CURRENT_USER;
24 GO
25 CREATE OR ALTER VIEW SellerOrderItems AS
26 SELECT oi.*
27 FROM OrderItems oi
28 JOIN Sellers s ON oi.SellerID = s.SellerID
29 WHERE s.Name = CURRENT_USER;
30 GO
31 CREATE OR ALTER VIEW SellerProfile AS
32 SELECT s.*

```

```

33 FROM Sellers s
34 WHERE s.Name = CURRENT_USER;
35 GO

```

Листинг 3.14 – Назначение прав для UserRole

```

1 GRANT SELECT ON Games TO UserRole;
2 GRANT SELECT ON Categories TO UserRole;
3 GRANT SELECT ON Sellers TO UserRole;
4 GRANT SELECT, INSERT ON Orders TO UserRole;
5 GRANT SELECT, UPDATE ON OrderItems TO UserRole;
6 GRANT SELECT, INSERT ON Reviews TO UserRole;
7 GRANT SELECT, UPDATE ON Users TO UserRole;
8 GRANT INSERT ON Gifts TO UserRole;
9 GO
10 GRANT SELECT, INSERT ON UserOrders TO UserRole;
11 GRANT SELECT ON UserOrderItems TO UserRole;
12 GRANT SELECT, INSERT, UPDATE ON UserReviews TO UserRole;
13 GRANT SELECT, UPDATE ON UserProfile TO UserRole;
14 GRANT SELECT ON UserSentGifts TO UserRole;
15 GRANT SELECT ON UserReceivedGifts TO UserRole;
16 GRANT SELECT ON dbo.SellerOrderItems TO UserRole;
17 GO

```

Листинг 3.15 – Назначение прав для SellerRole

```

1 GRANT SELECT ON Users TO SellerRole;
2 GRANT SELECT, INSERT ON Games TO SellerRole;
3 GRANT SELECT ON Categories TO SellerRole;
4 GRANT SELECT ON Orders TO SellerRole;
5 GRANT SELECT, INSERT, UPDATE ON OrderItems TO SellerRole;
6 GRANT SELECT, UPDATE ON Sellers TO SellerRole;
7 GRANT SELECT ON UserProfile TO SellerRole;
8 GO
9 GRANT SELECT, UPDATE ON SellerOrderItems TO SellerRole;
10 GRANT SELECT, UPDATE ON SellerProfile TO SellerRole;
11 GO

```

Листинг 3.16 – Назначение прав для AdminRole

```

1 GRANT SELECT, INSERT, UPDATE, DELETE ON Users TO AdminRole;
2 GRANT SELECT, INSERT, UPDATE, DELETE ON Games TO AdminRole;
3 GRANT SELECT, INSERT, UPDATE, DELETE ON Categories TO AdminRole;
4 GRANT SELECT, INSERT, UPDATE, DELETE ON Sellers TO AdminRole;

```

```

5 GRANT SELECT, INSERT, UPDATE, DELETE ON Orders TO AdminRole;
6 GRANT SELECT, INSERT, UPDATE, DELETE ON OrderItems TO AdminRole;
7 GRANT SELECT, INSERT, UPDATE, DELETE ON Reviews TO AdminRole;
8 GO
9 REVOKE SELECT, INSERT, UPDATE, DELETE ON UserProfile TO AdminRole;
10 REVOKE SELECT, INSERT, UPDATE, DELETE ON UserReviews TO AdminRole;
11 REVOKE SELECT, INSERT, UPDATE, DELETE ON UserOrderItems TO
    AdminRole;
12 REVOKE SELECT, INSERT, UPDATE, DELETE ON UserOrders TO AdminRole;
13 REVOKE SELECT, INSERT, UPDATE, DELETE ON SellerProfile TO
    AdminRole;
14 REVOKE SELECT, INSERT, UPDATE, DELETE ON SellerOrderItems TO
    AdminRole;
15 GO

```

3.8 Тестирование триггера

Реализация триггера `trg_Users_AfterInsertDelete` приведена в листинге 3.9. Триггер срабатывает после операций вставки или удаления записей в таблице `Users` и вызывает хранимую процедуру `sp_ManageSqlLogin` для создания или удаления SQL-логинов. При вставке пользователя в систему ему присваивается роль `UserRole`. При удалении пользователя соответствующий SQL-логин удаляется. Тестирование триггера для таблицы `Sellers` (`trg_Sellers_AfterInsertDelete`) проводится аналогично, с использованием роли `SellerRole`.

3.8.1 Классы эквивалентности

Для тестирования триггера `trg_Users_AfterInsertDelete` выделены следующие классы эквивалентности:

- **Для операции вставки:**
 - КЭ1: Успешная вставка пользователя с уникальным логином (покрывается тестом 1).
 - КЭ2: Вставка пользователя с уже существующим логином (покрывается тестом 3).

- Для операции удаления:
 - КЭЗ: Успешное удаление существующего пользователя (покрывается тестом 2).
 - КЭ4: Удаление несуществующего пользователя (покрывается тестом 4).

3.8.2 Тестовые случаи

Ниже приведены тестовые случаи для проверки работы триггера `trg_Users_AfterInsertDelete`.

1. Создание обычного пользователя

Входные данные: В таблице `Users` уже есть запись. Вставка записи с `UserID = '01B49FCE-3CAD-4D72-9680-21DCB2F2A974'`, `Name = 'Kyle22'`, `Password = 'UserPass123'`, `Email = 'user@example.com'`, `RegistrationDate = '2025-05-11'`, `Country = 'Russia'`, `IsBlocked = 0`, `Balance = 0.00`.

Ожидаемый результат: Новая запись добавлена в таблицу `Users`. Создан SQL-логин `Kyle22` с паролем `UserPass123`, пользователь добавлен в роль `UserRole`. Таблица `ErrorLog` не изменяется.

2. Удаление существующего пользователя

Входные данные: Удаление записи из таблицы `Users` с `Name = 'Kyle22'`.

Ожидаемый результат: Запись удалена из таблицы `Users`. SQL-логин `Kyle22` удален. Таблица `ErrorLog` не изменяется.

3. Попытка создания пользователя с уже существующим логином

Входные данные: Вставка записи с `UserID = 'C29FD498-1C80-4CF3-82FA-23010C08A06F'`, `Name = 'AdminUser'`, `Password = 'UserPass123'`, `Email = 'newuser@example.com'`, `RegistrationDate = '2025-05-11'`, `Country = 'Russia'`, `IsBlocked = 0`, `Balance = 0.00`, где логин `AdminUser` уже существует.

Ожидаемый результат: Запись добавлена в таблицу `Users`, но SQL-логин не создан. В таблице `ErrorLog` добавлена запись с сообщением об

ошибке: 'SQL login already exists for Name: AdminUser'.

4. Удаление несуществующего пользователя

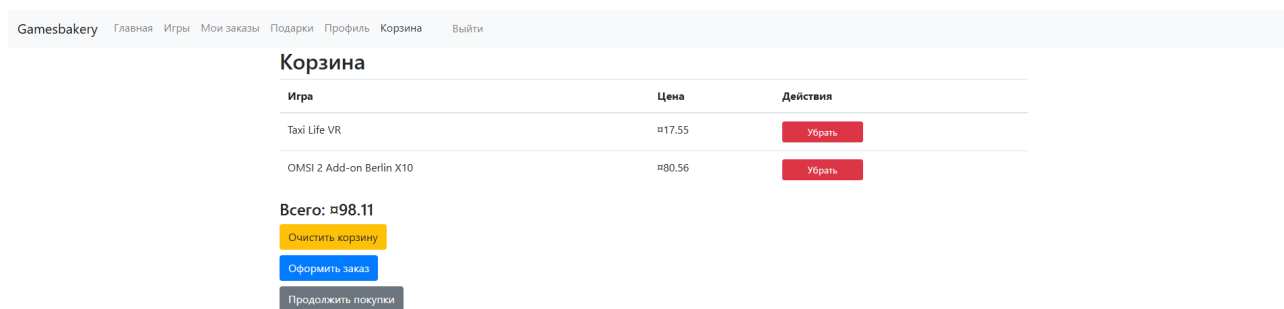
Входные данные: Удаление записи из таблицы `Users` с `Name = 'NonExistingUser'`, которая отсутствует.

Ожидаемый результат: Триггер не срабатывает, так как запись не существует. Таблицы `Users` и `ErrorLog` не изменяются.

Тестирование триггера `trg_Sellers_AfterInsertDelete` проводится аналогично, с учетом использования таблицы `Sellers` и роли `SellerRole`. Тестовые случаи включают создание и удаление продавцов, а также обработку ошибок при попытке создания продавца с уже существующим логином.

3.9 Пример работы программы

На рисунке 3.1 показана страница корзины веб-приложения Gamesbakery. Вверху отображаются уведомления об ошибках или успехе. Таблица содержит игры с названием, ценой и кнопкой «Убрать». Итоговая сумма и кнопки «Очистить корзину» и «Оформить заказ» расположены ниже. Данные загружаются из таблицы `OrderItems`.



Игра	Цена	Действия
Taxi Life VR	≈17.55	Убрать
OMSI 2 Add-on Berlin X10	≈80.56	Убрать

Всего: ≈98.11

Очистить корзину

Оформить заказ

Продолжить покупки

Рисунок 3.1 – Страница корзины

3.10 Вывод

В разделе описана реализация базы данных и приложения Gamesbakery, обоснован выбор СУБД, ЯП и фреймворка. Представлены SQL-скрипты для таблиц, представлений, индексов, процедур и триггеров. Приведен пример интерфейса корзины. Тестирование триггеров подтвердило их корректность.

4 Исследовательский раздел

В данном разделе проведено исследование влияния индексов на время выполнения запросов в базе данных магазина игр. Цель исследования — определить оптимальный набор индексов для ускорения запросов с объединением таблиц и фильтрацией.

4.1 Технические характеристики

Замеры проводились для оценки производительности запросов и влияния индексов на скорость выполнения операций в локальной среде. Тестирование осуществлялось на следующем оборудовании:

- Процессор: Intel(R) Core(TM) i5-10300H CPU @ 2.60 ГГц [13];
- Оперативная память: 16 ГБ;
- ОС: Windows 11 [14];
- СУБД: Microsoft SQL Server.

4.2 Временные характеристики

Исследование направлено на оценку влияния индексов на скорость выполнения запроса, включающего объединение таблиц `OrderItems`, `Orders`, `Games` и `Sellers` с фильтрацией по `OrderID` и `SellerID`. Запрос представлен в листинге 4.1.

Листинг 4.1 – Запрос на поиск элементов заказа

```
1 SELECT oi.OrderItemID , o.OrderDate , g.Title , s.Name
2 FROM OrderItems oi
3 JOIN Orders o ON oi.OrderID = o.OrderID
4 JOIN Games g ON oi.GameID = g.GameID
5 JOIN Sellers s ON oi.SellerID = s.SellerID
6 WHERE oi.OrderID = (SELECT TOP 1 OrderID FROM Orders ORDER BY
   NEWID())
7 AND oi.SellerID = (SELECT TOP 1 SellerID FROM Sellers ORDER BY
   NEWID());
```


Для анализа были выбраны следующие индексы на таблице *OrderItems*:

- Без индекса;
- Индекс по **OrderID**;
- Индекс по **SellerID**;
- Составной индекс по **OrderID** и **SellerID**.

Замеры проводились при числе строк в таблице **OrderItems** от 10 до 50 тысяч с шагом 5 тысяч. Результат выполнения запроса считался как среднее арифметическое время за 100 замеров в миллисекундах. Результаты представлены в таблице 4.1.

Таблица 4.1 – Замеры времени выполнения запроса с различными индексами

Число строк, тыс.	Время, мс			
	Без индекса	OrderID	SellerID	Составной
10	7.8	3.7	4.02	3.53
15	6.49	8.19	4.88	8.77
20	11.0	5.82	5.73	7.7
25	11.04	6.07	15.96	2.89
30	10.66	4.13	7.63	3.28
35	8.06	4.1	11.06	2.71
40	16.36	4.24	9.45	8.76
45	20.06	2.73	9.2	4.44
50	26.48	3.78	11.65	3.95

Зависимость времени выполнения запроса от числа строк представлена на рисунке 4.1.

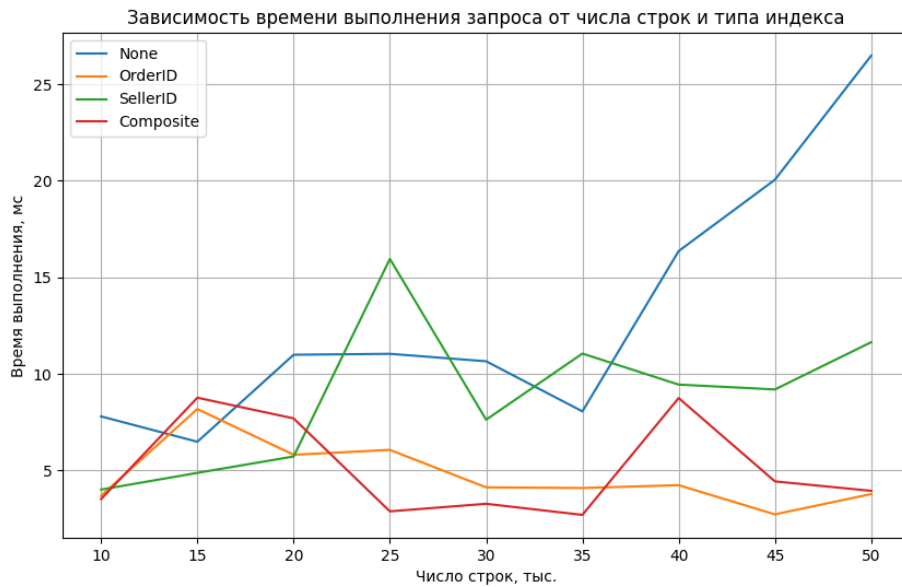


Рисунок 4.1 – Зависимость времени выполнения запроса от числа строк

Для выбора лучшего индекса были сравнены индекс по **OrderID** и составной индекс (таблица 4.2).

Таблица 4.2 – Сравнение времени выполнения запроса для индекса по **OrderID** и составного индекса

Число строк, тыс.	Индекс по OrderID, мс	Составной индекс, мс
5	3.55	2.38
10	4.64	2.24
15	6.99	2.38
20	8.67	2.00
25	9.19	2.23
30	10.35	2.65
35	10.08	2.34
40	8.65	1.85
45	7.96	2.40
50	9.34	1.90

Зависимость времени выполнения запроса для этих индексов (с шагом 1 тыс.) представлена на рисунке 4.2.

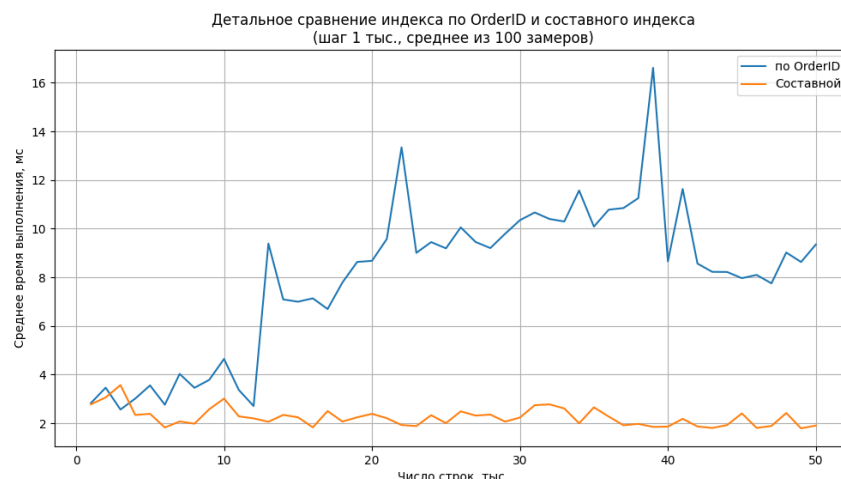


Рисунок 4.2 – Сравнение времени выполнения запроса для индекса по OrderID и составного индекса

4.3 Вывод

В данном разделе было проведено исследование зависимости времени выполнения запроса на поиск элементов заказа, путем объединения 4 таблиц и применения дополнительной фильтрации разными способами: без индексов, с простыми индексами и с составным индексом.

Анализ результатов замеров показал, что отсутствие индексов приводит к значительному увеличению времени выполнения запроса: с 7.8 мс при 10 тысячах строк до 26.48 мс при 50 тысячах строк. Индекс по `OrderID` снижает время до 3.55–10.35 мс, демонстрируя улучшение, но с ростом объема данных время увеличивается. Индекс по `SellerID` оказался менее эффективным (4.02–15.96 мс), так как этот столбец реже используется в фильтрации. Составной индекс по `OrderID` и `SellerID` показал наилучшие результаты, стабильно удерживая время выполнения в диапазоне 1.85–2.65 мс даже при увеличении объема данных до 50 тысяч строк, что на 30–78% быстрее по сравнению с индексом по `OrderID`.

Детальное сравнение (таблица 4.2) подтвердило превосходство составного индекса: его время выполнения варьируется от 1.85 мс (при 40 тысяч строк) до 2.65 мс (при 30 тысяч строк), тогда как индекс по `OrderID` показывает 3.55–10.35 мс. Таким образом, для всех рассмотренных объемов данных рекомендуется использовать составной индекс, так как он обеспечивает стабильное ускорение и предсказуемое время выполнения запросов.

ЗАКЛЮЧЕНИЕ

В ходе курсовой работы была успешно достигнута цель — разработана базы данных для хранения и обработки информации магазина видеоигр.

Для достижения были решены следующие задачи:

- изучена предметная область магазинов видеоигр;
- сформулированы требования и ограничения к разрабатываемой базе данных и приложению;
- спроектированы сущности базы данных, определены ограничения целостности, а также разработаны функции и ролевая модель на уровне базы данных;
- реализованы сущности базы данных магазина видеоигр с учетом ограничений целостности и описан интерфейс доступа к базе данных;
- проведено исследование влияния индексов на скорость выполнения запросов разработанной системы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Роб П., Коронел К. Системы баз данных: проектирование, реализация и управление. БХВ-Петербург, 2004. с. 20.
2. Кон С. OLTP and OLAP Data Integration. IEEE SoutheastCon, 2005. с. 515.
3. Кон С. OLTP and OLAP Data Integration. IEEE SoutheastCon, 2005. с. 516.
4. Тарасов С. В. СУБД для программиста. Базы данных изнутри. 2015.
5. Кузнецов С.Д. Основы организации современных баз данных [Электронный ресурс], 2015. URL: <http://www.lcard.ru/nail/database/osbd/contents.htm> (дата обращения 13.03.2025).
6. Steampay. Официальный сайт [Электронный ресурс]. 2025. URL: <https://steampay.com> (дата обращения: 13.03.2025).
7. Kupikod. Официальный сайт [Электронный ресурс]. 2025. URL: <https://kupikod.com> (дата обращения: 13.03.2025).
8. Zaka-Zaka. Официальный сайт [Электронный ресурс]. 2025. URL: <https://zaka-zaka.com> (дата обращения: 13.03.2025).
9. Парфенов Ю., Папуловская Н. Постреляционные хранилища данных. Litres, 2017. с. 14.
10. Нотации диаграммы прецедентов [Электронный ресурс], 2018. URL: <https://circle.visual-paradigm.com/docs/uml-and-sysml/use-case-diagram/use-case-diagram-notations-guide/> (дата обращения 13.03.2025).
11. Chen P.P.S. The entity-relationship model—toward a unified view of data. 1976. с. 9-36
12. Microsoft. Официальная документация ASP.NET Core [Электронный ресурс]. 2025. URL: <https://docs.microsoft.com/ru-ru/aspnet/core/> (дата обращения: 18.04.2025).
13. Intel. Intel® Core™ i5-10300H Processor [Электронный ресурс]. 2025. URL: <https://ark.intel.com/content/www/us/en/ark/products/201839/intel-core-i5-10300h-processor-8m-cache-up-to-4-50-ghz.html> (дата обращения: 18.04.2025).
14. Microsoft. Windows 11 Home [Электронный ресурс]. 2025. URL: <https://www.microsoft.com/ru-ru/windows/> (дата обращения: 18.04.2025).

ПРИЛОЖЕНИЕ А

Презентация к курсовой работе

Презентация к курсовой работе состоит из 14 слайдов.