МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(национальный исследовательский университет)» (МАИ)
Институт №8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Курсовой проект
по дисциплине «Базы данных»
Тема: «Видеоигровой портал»

Студент: Тарасов Е.Д.
Группа: М80-309Б-23
Преподаватель: Грубенко М.Д.
Оценка: _____
Дата: _____
Подпись: _____

Москва 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современных условиях цифровизации экономики и стремительного развития сферы развлечений, информационные технологии становятся фундаментом для эффективного управления контентом и взаимодействием с пользователями. Одной из наиболее динамично развивающихся областей является рынок видеоигр и цифровых платформ для геймеров. Рост популярности игр создает высокую потребность в надежных и масштабируемых цифровых платформах для хранения и анализа данных об играх, пользователях и их взаимодействии.

Объектом исследования в данной курсовой работе является процесс автоматизации деятельности игровых порталов. Данная сфера характеризуется высокой интенсивностью пользовательских взаимодействий, необходимостью строгого контроля прогресса и отзывов, а также обеспечения безопасности данных всех участников.

Предметом исследования является проектирование архитектуры и практическая реализация реляционной базы данных, а также интегрированного с ней серверного приложения для управления пользователями, играми, компаниями-разработчиками, жанрами, платформами, прогрессом пользователей и отзывами в рамках разрабатываемой информационной системы.

Актуальность данной работы обусловлена необходимостью решения ряда критических задач, таких как исключение конфликтов при одновременном доступе к данным, обеспечение прозрачности изменений данных через механизмы аудита и предоставление глубокой аналитики для разработчиков игр. Правильно спроектированная база данных позволяет не только эффективно хранить информацию, но и гарантирует целостность

данных в условиях конкурентного доступа.

Целью курсовой работы является создание полноценной информационной системы, демонстрирующей навыки проектирования сложных структур данных, реализации бизнес-логики на уровне СУБД с помощью триггеров и процедур, а также интеграции базы данных с современным backend-приложением.

Для достижения поставленной цели необходимо решить следующие задачи:

1) провести детальный анализ предметной области и выявить ключевые сущности и связи;
2) спроектировать логическую и физическую структуру реляционной базы данных, соответствующую требованиям нормализации;
3) реализовать механизмы обеспечения целостности данных с использованием ограничений и связей;
4) разработать процедурную логику на стороне сервера базы данных (функции, триггеры, хранимые процедуры) для автоматизации бизнес-процессов;
5) создать backend-приложение для взаимодействия с базой данных и обеспечения программного интерфейса (API);
6) провести оптимизацию производительности системы с использованием индексов и аналитических инструментов СУБД;
7) обеспечить контейнеризацию системы для упрощения развертывания и масштабирования.

Результатом работы является готовое к эксплуатации решение, сочетающее в себе строгость реляционной модели и гибкость современных программных архитектур.

1 АНАЛИТИЧЕСКАЯ ЧАСТЬ

1.1 Обзор предметной области

Предметная область информационной системы охватывает процессы взаимодействия между пользователями (игроками) и контентом игрового портала, включая игры, разработчиков, издателей, жанры и платформы. В условиях цифровой экономики данный сектор требует высокой степени автоматизации, так как традиционные способы ведения учета (ручные записи о прогрессе, бумажные журналы коллекций игр) не способны обеспечить необходимую скорость обработки данных и исключить ошибки человеческого фактора.

Основными бизнес-процессами в рассматриваемой области являются:

1) регистрация и профилирование участников: сбор и хранение контактных данных, аутентификационных данных и дополнительной информации. система четко определяет роль пользователя как игрока, отслеживающего прогресс и оставляющего отзывы;
2) управление каталогом игр: создание детальных описаний объектов (игр), включая название, описание, дату релиза, информацию о разработчике и издателе, а также связи с жанрами и платформами. важным аспектом является поддержание актуальной информации о созданных играх и их атрибутах;
3) отслеживание прогресса в играх: формирование записей о статусе игры для пользователя, часах игры, последней сессии и обновлениях. этот процесс является критическим, так как требует проверки связей между пользователями и играми без дубликатов;
4) аудит и мониторинг изменений: фиксация операций в журнале аудита

через триггеры, отслеживание изменений в таблицах и обеспечение целостности данных. система должна поддерживать каскадные обновления и удаления для связанных сущностей;

5) обратная связь и репутационный менеджмент: система отзывов и оценок позволяет формировать доверительную среду. рейтинг игры напрямую зависит от качества контента, зафиксированного в отзывах пользователей, с модерацией.

1.2 Постановка задачи

Основной задачей данной работы является разработка программного комплекса, который объединяет в себе надежное хранилище данных и программный интерфейс для управления ими. в рамках постановки задачи необходимо решить проблему эффективного хранения больших объемов информации (тысячи записей) и обеспечения быстрого доступа к ним.

Ключевые проблемы, которые должна решать проектируемая система:

1) овербукинг: исключение возможности двойного бронирования объекта недвижимости на перекрывающиеся интервалы дат. решение данной задачи должно быть реализовано на уровне базы данных для обеспечения максимальной надежности;
2) целостность данных: гарантия того, что при удалении пользователя или объекта недвижимости все связанные с ними записи (бронирования, платежи) обрабатываются корректно в соответствии с заданными бизнес-правилами;
3) прослеживаемость изменений: создание механизма, позволяющего отследить, кем, когда и какие данные были изменены. это необходимо для разбора спорных ситуаций и повышения уровня безопасности;

4) аналитическая отчетность: предоставление сводных данных для оценки эффективности работы сервиса, расчет доходности хостов и популярности объектов.

С технической точки зрения система должна обеспечивать бесперебойную работу через REST API, поддерживать автоматическую миграцию структуры базы данных и быть готовой к развертыванию в изолированных контейнерах. это обеспечит переносимость решения и независимость от специфических настроек локальной среды разработчика или сервера.

2 ПРОЕКТНАЯ ЧАСТЬ

2.1 Архитектура системы

Проектируемая информационная система построена по принципу многоуровневой архитектуры, что обеспечивает разделение ответственности между компонентами и упрощает дальнейшую поддержку и масштабирование решения. в состав системы входят следующие уровни:

1) уровень базы данных: центральное звено системы, реализованное на базе СУБД PostgreSQL. на данном уровне сосредоточена не только информация, но и критически важная бизнес-логика, реализованная в виде ограничений целостности, триггеров и хранимых процедур;
2) уровень доступа к данным (repository layer): программный компонент на языке Python, который отвечает за формирование и выполнение запросов к базе данных. использование библиотеки SQLAlchemy позволяет исключить риск возникновения SQL-инъекций за счет обязательной параметризации всех входных данных;
3) уровень бизнес-логики (service layer): содержит алгоритмы обработки данных, специфичные для сервиса аренды. здесь происходит координация работы с репозиториями, управление транзакциями на уровне приложения и валидация данных перед их сохранением;
4) уровень программного интерфейса (api layer): обеспечивает взаимодействие внешних потребителей с системой через протокол HTTP. использование REST-архитектуры и формата JSON делает систему совместимой с любыми современными фронтенд-решениями и сторонними сервисами.

2.2 Проектирование структуры базы данных

Проектирование информационной системы базируется на концептуальной ER-модели (рисунок 2.1), которая описывает ключевые сущности предметной области и характер их взаимодействия. логическая и физическая структуры базы данных разработаны с учетом требований третьей нормальной формы, что гарантирует отсутствие аномалий обновления и минимизацию избыточности.

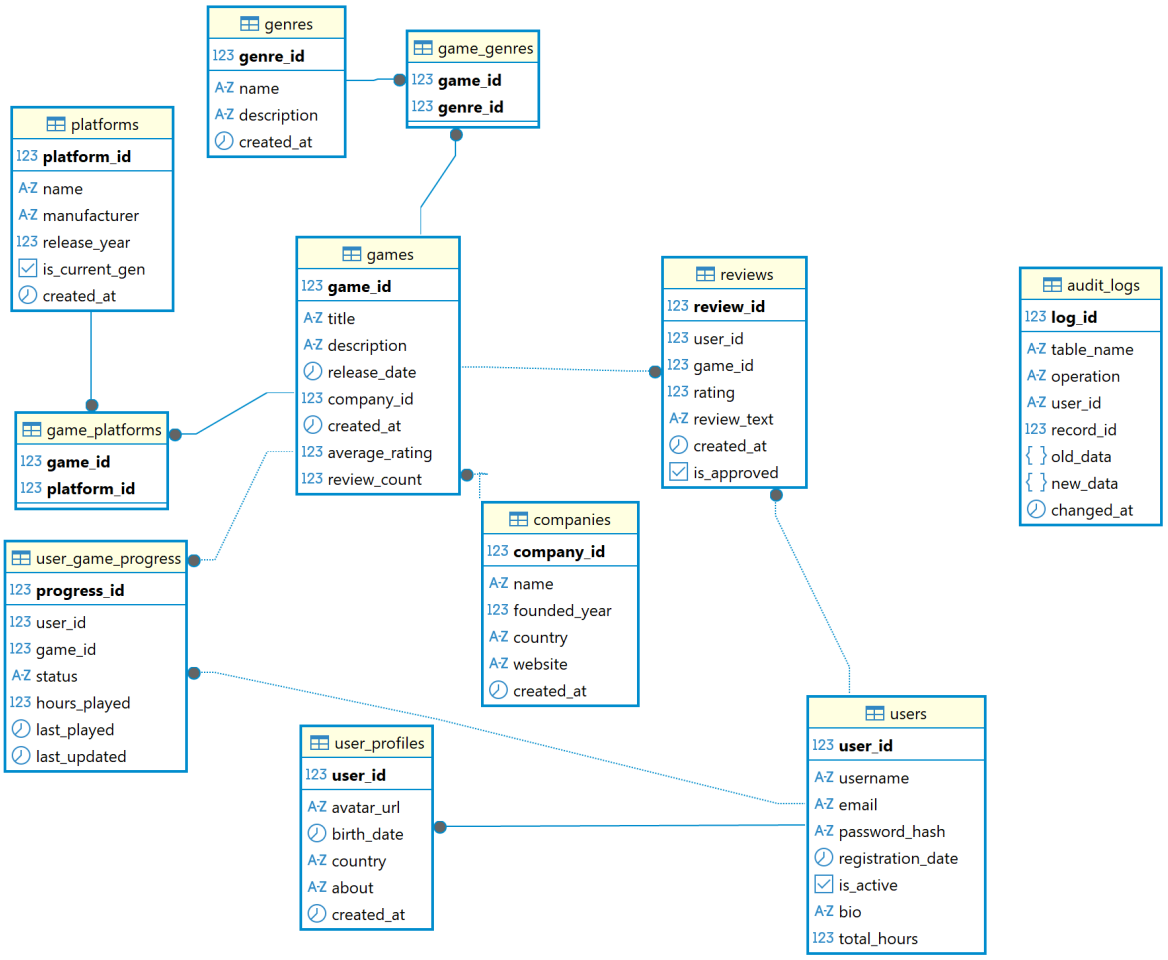


Рисунок 2.1 — ER-диаграмма

В основе системы лежат следующие механизмы обеспечения целостности данных:

1) первичные ключи (primary key): каждая таблица снабжена уникальным

идентификатором (например, user_id, game_id, review_id, company_id, genre_id и т.д.), обеспечивающим однозначную адресацию любой записи в системе;

2) внешние ключи (foreign key): связи между сущностями реализованы через ссылочную целостность. для большинства связей настроено каскадное удаление и обновление (on delete cascade, on update cascade), что позволяет автоматически очищать зависимые данные (например, при удалении пользователя автоматически удаляются его записи прогресса в user_game_progress и отзывы в reviews; при удалении игры — связи в game_genres, game_platforms, прогресс и отзывы);
3) ограничения уникальности (unique): гарантируют отсутствие дубликатов для критически важных полей, таких как username и email в таблице users, title в таблице games, name в таблицах companies, genres и platforms;
4) ограничения проверки (check): реализуют бизнес-логику на уровне схемы данных. например, проверка валидности founded_year в companies, hours_played в user_game_progress и другие.

В состав системы входят 10 основных таблиц (плюс дополнительные для связей и аудита), каждая из которых имеет строго определённую роль и набор ограничений:

1) users: содержит основные данные пользователей игрового портала. первичный ключ — user_id. на поля username и email наложены ограничения unique. ключевые поля (username, email, password_hash) имеют признак not null. дополнительно присутствует проверка формата email через check с регулярным выражением;
2) user_profiles: реализует связь один-к-одному с таблицей users и хранит дополнительную информацию о пользователях (avatar_url, birth_date,

country). первичный ключ — user_id (одновременно внешний ключ на users.user_id с on delete cascade). ограничение unique на user_id обеспечивает строгую связь 1:1;

3) companies: описывает компании-разработчиков и издателей. первичный ключ — company_id. поле name уникально (unique). реализованы ограничения check: founded_year > 1900 и <= текущего года, role в списке ('developer', 'publisher', 'both');
4) genres: хранит жанры игр. первичный ключ — genre_id. поле name уникально (unique);
5) platforms: содержит информацию о игровых платформах. первичный ключ — platform_id. поле name уникально (unique). ограничение check на release_year > 1900;
6) games: центральная таблица каталога игр. первичный ключ — game_id. поле title уникально (unique). содержит внешние ключи developer_id и publisher_id (могут быть null). поля description и title not null;
7) game_genres и game_platforms: реализуют связи «многие-ко-многим» между играми и жанрами/платформами соответственно. в обеих таблицах определён составной первичный ключ ((game_id, genre_id) и (game_id, platform_id)), что исключает дублирование связей. внешние ключи с on delete cascade и on update cascade;
8) user_game_progress: обеспечивает отслеживание прогресса пользователей в играх. первичный ключ — progress_id. содержит внешние ключи user_id и game_id (оба not null) с каскадным удалением. ограничение unique на пару (user_id, game_id) для предотвращения дублирования записей одного пользователя по одной игре. срусл-констрейнты: status в списке ('playing', 'completed', 'planned', 'dropped'), hours_played >= 0;
9) reviews: хранит отзывы пользователей об играх. первичный ключ —

review_id. внешние ключи user_id и game_id с каскадным удалением. ограничение unique на пару (user_id, game_id) — один пользователь может оставить только один отзыв на игру. Поле rating — not null, is_approved по умолчанию true;

10) audit_logs: реализован через триггеры (например, trigger_audit_review и аналогичные для других таблиц), записывающие изменения в отдельную таблицу audit_logs (или аналогичную структуру), содержащую информацию об операции, старых и новых значениях, времени изменения и, при возможности, идентификаторе пользователя.

Взаимодействие сущностей характеризуется различными типами связей:

1) один-к-одному (1:1): связь между users и user_profiles (каждый пользователь может иметь не более одного профиля, и каждый профиль принадлежит ровно одному пользователю);
2) один-ко-многим (1:N): пользователь — множество записей прогресса и отзывов; игра — множество записей прогресса, отзывов, связей с жанрами и платформами; компания — множество игр (как разработчик или издатель);
3) многие-ко-многим (N:M): игра — множество жанров и платформ (реализовано через промежуточные таблицы game_genres и game_platforms)..

2.3 Описание API и взаимодействия с базой данных

Взаимодействие между серверным приложением и системой управления базами данных организовано через ORM-библиотеку SQLAlchemy, которая обеспечивает безопасный и структурированный

доступ к информации. Работа с данными строится на следующих принципах:

1) использование библиотеки SQLAlchemy: для выполнения операций с базой данных применяется мощный ORM-инструмент SQLAlchemy версии 2.0. это позволяет автоматизировать сопоставление моделей Python (классов, таких как User, Game, Review) с таблицами базы данных, значительно сокращает объём вспомогательного кода и снижает вероятность возникновения ошибок при обработке данных;
2) параметризация запросов и защита от инъекций: все запросы, формируемые через ORM, автоматически параметризуются. входные данные от пользователей (через API) валидируются с помощью Pydantic-схем (BaseModel), что полностью исключает возможность атак типа SQL-инъекция и обеспечивает высокий уровень безопасности системы;
3) управление сессиями и транзакциями: для выполнения операций используется механизм сессий (SessionLocal), предоставляемый SQLAlchemy. все изменения в рамках одного запроса происходят в одной транзакции: при успешном завершении вызывается commit(), при ошибке — автоматический rollback. это гарантирует атомарность действий, особенно при создании или обновлении связанных сущностей;
4) реализация REST-архитектуры: программный интерфейс на базе FastAPI предоставляет набор конечных точек (endpoints), сгруппированных по функциональному назначению в отдельных роутерах (users.py, games.py, reviews.py, batch.py, views.py). каждая точка поддерживает стандартные методы HTTP (get, post, put, delete),

что соответствует концепции CRUD-операций для основных сущностей системы (пользователи, игры, отзывы, прогресс). сложные аналитические запросы реализованы через материализованные представления (views) и прямое выполнение SQL.

Функциональные возможности api включают в себя следующие группы операций:

1) управление пользователями и каталогом игр: регистрация пользователей, получение списка пользователей, добавление, редактирование и удаление игр , получение информации об отдельной игре;
2) работа с прогрессом и отзывами: добавление отзывов о играх, получение отзывов по игре с фильтром по is_approved = true), отслеживание прогресса пользователей в играх (через таблицу user_game_progress, доступную косвенно через связанные сущности);
3) доступ к аналитическим данным: выполнение запросов к представлениям базы данных для получения сводной отчётности;
4) пакетная обработка данных: специальный интерфейс для массового добавления игр, принимающий список объектов и вставляющий только новые записи (с проверкой на дубли по title), с подсчётом количества успешно добавленных игр и автоматическим commit в рамках одной транзакции.

2.4 Описание процедурной логики базы данных

Одной из ключевых особенностей проектируемой системы является перенос части критически важной бизнес-логики непосредственно на

сторону сервера базы данных. это обеспечивает инкапсуляцию правил обработки данных и гарантирует их выполнение независимо от того, через какой программный интерфейс происходит обращение к СУБД. процедурная логика реализована с использованием языка PL/pgSQL и включает в себя следующие компоненты:

1) ограничения и каскадные действия для управления целостностью: реализованы механизмы ссылочной целостности через foreign key с on delete cascade и on update cascade. например, при удалении пользователя автоматически удаляются все его записи прогресса (user_game_progress) и отзывы (reviews), а при удалении игры — все связанные записи в game_genres, game_platforms, user_game_progress и reviews. это исключает риск возникновения «висячих» ссылок и обеспечивает атомарность связанных операций на уровне базы данных;
2) представления (views) и функции для аналитических расчётов: разработаны материализованные или обычные представления (game_ratings_view, user_stats_view, popular_games_view), а также функции (например, get_game_rating(game_id), get_user_total_hours(user_id), get_top_players_by_genre(genre_name)). представления используются для агрегации данных — расчёта среднего рейтинга игр, количества отзывов, статистики пользователей (общее время игры, завершённые игры) и популярности игр по количеству игроков. функции позволяют оперативно получать сложные показатели за один вызов;
3) триггеры для автоматизации аудита: для обеспечения прослеживаемости изменений реализованы триггеры на ключевых таблицах (например, trigger_audit_review для таблицы reviews, а также

аналогичные триггеры для users, games, user_game_progress). при вставке, обновлении или удалении записи триггер автоматически фиксирует информацию в таблице аудита (audit_logs или аналогичной), сохраняя тип операции (insert/update/delete), старые и новые значения данных (в формате jsonb или текстовом), время изменения и, при возможности, идентификатор пользователя;

4) триггеры и ограничения для поддержания целостности и уникальности: для предотвращения дублирования реализованы unique-констрейнты на критических парах (например, (user_id, game_id) в user_game_progress и reviews). триггеры также могут использоваться для автоматического обновления агрегатных полей (например, average_rating и review_count в таблице games при добавлении/удалении/обновлении отзыва), что повышает производительность при частых выборках каталога игр без необходимости выполнения ресурсоёмких агрегатных запросов каждый раз.

2.5 Представления

Для упрощения доступа к наиболее часто используемым наборам данных и сокрытия сложности внутренних соединений в базе данных реализованы представления:

1) game_ratings_view: объединяет данные из таблиц games и reviews для предоставления полной информации о рейтинге каждой игры — средний рейтинг (average_rating), количество отзывов (review_count), название игры и дату релиза. используется для быстрого получения отсортированного списка игр по популярности и оценкам;
2) user_stats_view: предоставляет сводную статистику по активности

каждого пользователя, включая общее количество игр в прогрессе (total_games), количество завершённых игр (completed_games), суммарное время игры (total_hours), имя пользователя и дату регистрации;

3) popular_games_view: служит инструментом для аналитики популярности игр, объединяя данные из games, user_game_progress и reviews. для каждой игры рассчитывается количество игроков (players_count) и средний рейтинг (average_rating), с сортировкой по убыванию популярности (топ-10 или полный список).

2.6 Оптимизация и анализ производительности

Для обеспечения стабильно высокой скорости работы системы при масштабировании данных была проведена оптимизация запросов:

1) индексирование: созданы индексы для всех полей, наиболее часто участвующих в условиях поиска, операциях соединения и сортировки. особое внимание уделено внешним ключам и полям, по которым осуществляется фильтрация в аналитических отчетах;
2) анализ планов выполнения: с помощью инструментов профилирования запросов был проведен сравнительный анализ производительности. использование индексов позволило значительно сократить время выполнения операций поиска по сравнению с полным сканированием таблиц, что подтверждено результатами выполнения команды анализа планов запросов.

3 ТЕХНОЛОГИЧЕСКАЯ ЧАСТЬ

3.1 Средства разработки и технологии

Для реализации информационной системы был выбран современный стек технологий, обеспечивающий высокую производительность и надежность:

1) язык программирования backend-части: Python 3.12 с фреймворком FastAPI. выбор обусловлен высокой скоростью разработки, автоматической генерацией документации API (OpenAPI/Swagger), асинхронной поддержкой, отличной интеграцией с SQLAlchemy и Pydantic, а также встроенной валидацией данных и обработкой ошибок;
2) система управления базами данных: PostgreSQL. данная СУБД была выбрана за ее развитые возможности работы с реляционными данными, поддержку JSONB для реализации аудита и мощный язык процедурного программирования;
3) средства контейнеризации: Docker и Docker Compose. использование контейнеров позволяет полностью изолировать среду исполнения приложения от операционной системы, гарантируя идентичность работы сервиса на различных серверах.

3.2 Процесс развертывания и заполнения данными

Для подготовки системы к работе был реализован автоматизированный процесс инициализации:

1) инициализация структуры базы данных: при запуске контейнера PostgreSQL автоматически выполняется init.sql (монтируется через volume в docker-compose.yml), создающий таблицы, ограничения,

индексы, представления, функции и триггеры.

2) массовая загрузка демонстрационных данных: реализован специализированный модуль для генерации реалистичных данных. процесс загрузки происходит пакетами, что позволяет за короткое время наполнить базу тысячами записей о пользователях, играх и обзорах. это необходимо для полноценного тестирования аналитических отчетов и проверки производительности системы под нагрузкой.

3.3 Тестирование

Для тестирования механизма работы приложения есть пользовательский интерфейс, реализованный с помощью Swagger-документации (рисунок 3.1). Он позволяет обращаться к api приложения и возвращает ответ в формате JSON.

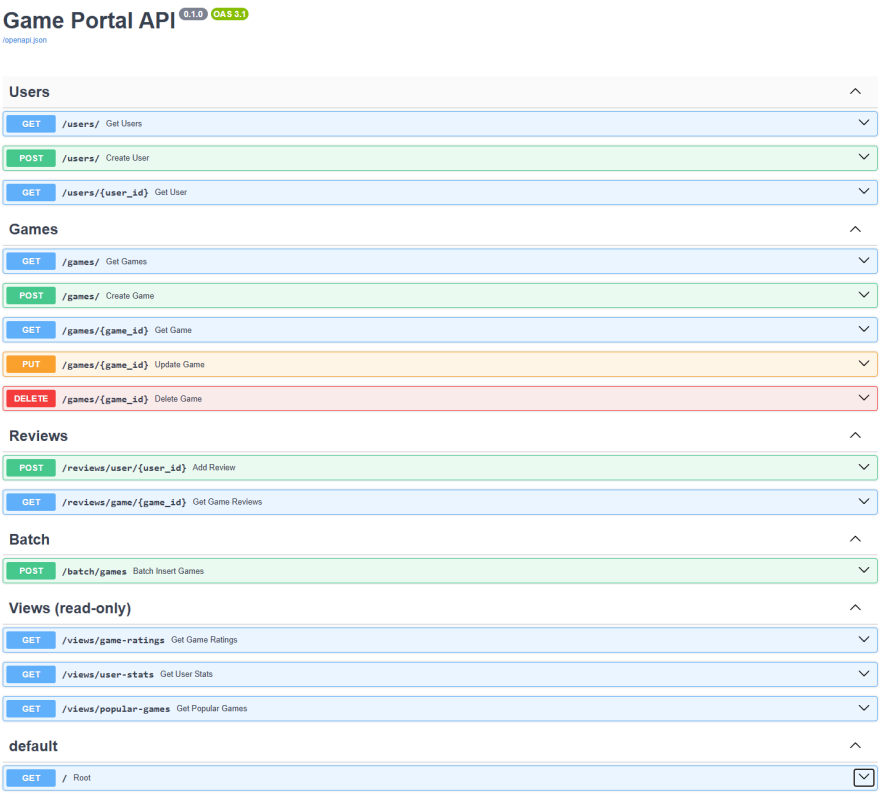


Рисунок 3.1 — Пример интерфейса Swagger

В качестве примера обратимся к api получения списка пользователей, которые есть в базе данных (рисунок 3.2). Результат имеет код ответа, а также текст ответа в формате JSON.

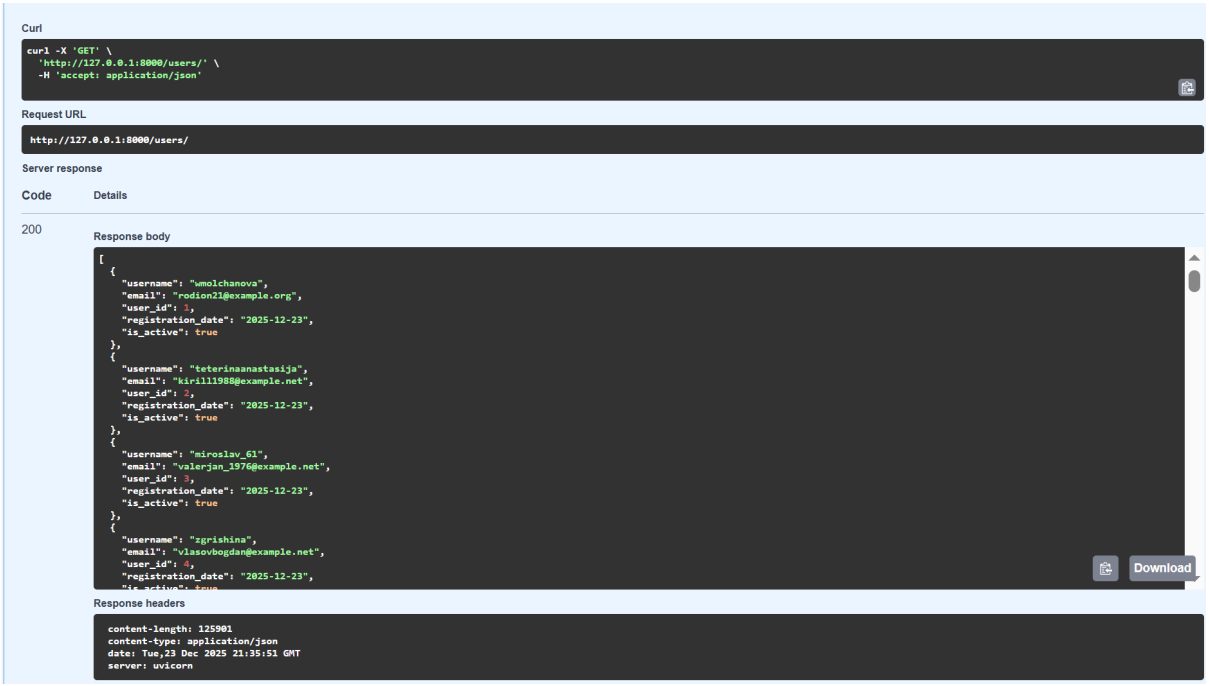
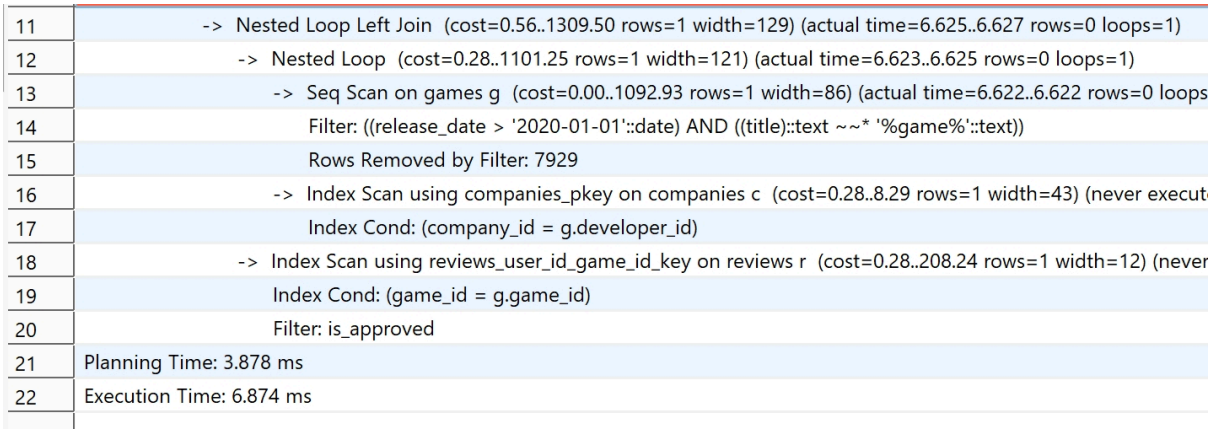


Рисунок 3.2 — Результат вызова метода из API

3.4 Анализ производительности

В качестве примера для анализа производительности будет использоваться фрагмент запроса получения среднего рейтинга игры по дате и названию, так как именно в нём заметно, что индексы оправдывают свое использование. На рисунке 3.3 представлен запрос и его время работы без индекса, которое составляет 6.874 миллисекунд



```
11  ->  Nested Loop Left Join  (cost=0.56..1309.50 rows=1 width=129) (actual time=6.625..6.627 rows=0 loops=1)
12      ->  Nested Loop  (cost=0.28..1101.25 rows=1 width=121) (actual time=6.623..6.625 rows=0 loops=1)
13          ->  Seq Scan on games g  (cost=0.00..1092.93 rows=1 width=86) (actual time=6.622..6.622 rows=0 loops
14              Filter: ((release_date > '2020-01-01'::date) AND ((title)::text ~~* '%game%'::text))
15              Rows Removed by Filter: 7929
16          ->  Index Scan using companies_pkey on companies c  (cost=0.28..8.29 rows=1 width=43) (never execut
17              Index Cond: (company_id = g.developer_id)
18      ->  Index Scan using reviews_user_id_game_id_key on reviews r  (cost=0.28..208.24 rows=1 width=12) (never
19          Index Cond: (game_id = g.game_id)
20          Filter: is_approved
21  Planning Time: 3.878 ms
22  Execution Time: 6.874 ms
```

Рисунок 3.3 — Результат выполнения запроса до создания индекса

Далее на рисунке 3.4 изображен результат работы запроса после создания индекса. Время выполнения запроса составило 1.1936 миллисекунд, что в 5.76 раз быстрее чем запрос без индекса.

.

```
13      ->  Bitmap Heap Scan on games g  (cost=26.07..1071.55 rows=1 width=86) (actual time=1.
14          Recheck Cond: (release_date > '2020-01-01'::date)
15          Filter: ((title)::text ~~* '%game%'::text)
16          Rows Removed by Filter: 1306
17          Heap Blocks: exact=759
18          ->  Bitmap Index Scan on idx_games_release_date  (cost=0.00..26.07 rows=1305 width=
19              Index Cond: (release_date > '2020-01-01'::date)
20      ->  Index Scan using companies_pkey on companies c  (cost=0.28..8.29 rows=1 width=43) (
21          Index Cond: (company_id = g.developer_id)
22  ->  Index Scan using idx_reviews_game_approved on reviews r  (cost=0.28..8.30 rows=1 width=
23      Index Cond: (game_id = g.game_id)
24  Planning Time: 1.296 ms
25  Execution Time: 1.936 ms
```

Рисунок 3.4 — Результат выполнения запроса после создания индекса

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была спроектирована и реализована полноценная информационная система для игрового портала. Разработанное решение демонстрирует комплексный подход к проектированию баз данных, сочетающий в себе классические принципы нормализации и современные методы обеспечения целостности данных на стороне СУБД.

В процессе работы были достигнуты следующие результаты:

1) спроектирована структура реляционной базы данных, включающая 10 взаимосвязанных таблиц с настроенными ограничениями целостности;
2) реализована сложная процедурная логика, обеспечивающая автоматизацию аудита, защиту от пересекающихся бронирований и расчет финансовых показателей;
3) разработано backend-приложение, предоставляющее программный интерфейс для выполнения всех необходимых операций с данными;
4) проведена оптимизация производительности системы за счет внедрения стратегии индексирования и использования предварительно агрегированных данных.

Созданная система является масштабируемой и надежной основой для игрового портала, обеспечивающей целостность данных, быстрый доступ к аналитике и удобное взаимодействие пользователей с каталогом игр. Практическая значимость работы заключается в создании готового к развертыванию решения , которое может быть использовано как прототип реального сервиса или адаптировано под другие предметные области, требующие сложной реляционной структуры и аналитики.