**Западно-Казахстанский Инновационно-технологический Университет**

**Кафедра энергетики, автоматизации и вычислительной техники**

**Практическая работа**

**По дисциплине «Технологии баз данных»**

**Тема: Хранение мировых рекордов франшизы Touhou Project**

**Выполнил: студент 3-го курса Габдуллин Д.**

**Группа: ВТиИС-32О**

**Проверил: преподаватель Николаев А.А.**

**г. Уральск**  
**2025**

**СОДЕРЖАНИЕ**

1. **Введение…………………………………………….3**
2. **Техника безопасности во время практики……..5**
3. **Цель и задачи стажировки………………………..6**
4. **Задачи, выполняемые во время стажировки…..7**
5. **Используемые программы………………………..22**
6. **Результаты практики……………………………...22**
7. **Заключение………………………………………….23**
8. **Список материалов…………………………………24**

**Введение**

**Цель и задачи предметной области.**

Современные цифровые технологии помогают развитию игровых сообществ и формированию более удобных инструментов для работы с данными. Многие коммьюнити(сообщества) создают отдельные базы данных для записи их собственных достижений. В данной работе будет рассматриваться серия японских игр *Touhou Project*, известная своей высокой сложностью и наличием большого количества игровых значений, на которые можно ориентироваться – итоговый счёт, время прохождения, уровень сложности и персонаж, которого игрок контролировал. Все эти значения играют важные роли, так как методы и тактики, используемые в игре, могут сильно различаться в зависимости от всех этих факторов. Для сообщества Touhou каждый рекорд - достижение, которое побуждает игроков анализировать свои игровые сессии и использовать новые стратегии. Все это требует четкого и структурированного хранения.

**Целью** данного практического задания является разработка и проектирование реляционной базы данных в системе управления базами данных (СУБД), предназначенной для хранения и дальнейшего анализа мировых рекордов игроков по серии игр *Touhou Project*.

Для достижения поставленной цели необходимо решить следующие **задачи**:

* провести анализ предметной области и выявить основные сущности с их взаимосвязями;
* разработать реляционную схему и определить ключевые атрибуты для создания таблиц;
* нормализовать структуру базы данных для устранения избыточности и обеспечения целостности данных;
* реализовать даталогическую модель;
* обеспечить возможность выборки и сортировки данных в соответствии с запросами клиентов;
* разработать клиентское приложение для подключения и работы с базой данных.

**Краткое описание предметной области**  
Игровая франшиза *Touhou Project* представляет собой набор структурно схожих игр в жанре «данмаку-шутер» (bullethell). Основной характеристикой является фиксированная система уровней, персонажей и режимов сложности. Рекорды игроков выражаются числовыми значениями (на которые влияют множество факторов) и фиксируются для каждого забега, а также части отдельно. Поскольку с каждым рекордом объём данных постоянно увеличивается, а пользователи из разных стран добавляют результаты на регулярной основе, требуется создание централизованной базы данных, способной хранить, структурировать и что самое главное - анализировать информацию в соответствии с принципами реляционной модели.

**Обоснование актуальности**  
Разработка подобной базы данных актуальна по нескольким причинам. Во-первых, все больше игровых сообществ стремятся к строгому учёту достижений, а распределённое хранение данных (например в таблицах Excel) не обеспечивает целостности и доступности информации. Во-вторых, использование реляционной модели данных и механизмов СУБД позволяет гарантировать стабильность, устойчивость к сбоям и удобство масштабирования. В-третьих, рекорды продолжают устанавливать по сей день в связи с выходом 20-ой части данной франшизы, а также устанавливаются новые рекорды по старым частям благодаря анализу прошлых забегов и оптимизации стратегий.

Таким образом, можно применить основы о реляционных баз данных в игровой аналитике и связать их в одно единое для работы с данными.

1. **Анализ предметной области**

**Описание-бизнес-процессов.**  
Основной бизнес-процесс в рассматриваемой предметной области заключается в записи, верификации и хранении игровых результатов пользователей. После установки рекорда игрок может загрузить данные о своей игре (имя игрока, номер части, результат, выбранный персонаж и сложность). Администраторы или автоматизированная система проверяют достоверность предоставленных данных, после чего результат добавляется в общую таблицу рекордов. В дальнейшем пользователи могут осуществлять поиск по различным параметрам (игра, персонаж, сложность, дата рекорда и т.д.), формировать рейтинги и проводить анализ динамики игрока.

**Определение основных сущностей и атрибутов.**  
Для построения даталогической модели необходимо выделить ключевые сущности предметной области:

1. *Игрок (Player)* — характеризуется уникальным идентификатором, никнеймом и регионом(к этому мы вернёмся во время процесса нормализации).  
   *Основные атрибуты:* player\_id, nickname, region.
2. *Игровые данные* — несколько таблиц, указывающих информацию о частях, персонажах и уровнях определенных частей Touhou Project. Следует уточнить что фанатские и оригинальные названия персонажей отличаются, поэтому были созданы разные таблицы для этих название(позже они будут связаны при помощи представления)   
   *Основные атрибуты:* game\_id, title, stage\_id, character\_id, difficulty\_id и т.д.
3. **Р***екорды (Records)* — самая важная сущность, связывающая игрока, игру, персонажа, уровень сложности и дату установки рекорда.  
   *Основные атрибуты:* record\_id, player\_id, game\_id, char\_id, diff\_id, score, date.
4. *Очки уровней (Stage\_runs)* — позволит проанализировать динамику получения очков на определенных уровнях для сравнений.  
   *Основные атрибуты:* record\_id, , stage\_id, stage\_score.
5. *Ранги (Rankings)* — позволит сравнивать общие достижения игроков по количествам рекордов и полученных очков при установке рекордов.  
   *Основные атрибуты:* rank\_id, player\_id, last\_update\_date.

Между сущностями устанавливаются связи:

* Один игрок может иметь множество рекордов (*один-ко-многим*);
* Каждая игра содержит множество записей рекордов (*один-ко-многим*);
* Каждый рекорд определенно связывает игрока, игру, персонажа и сложность (*многие-ко-многим* реализуется через таблицу Records).

**Постановка задачи проектирования базы данных.**  
Основная задача проектирования базы данных - формализации предметной области. Необходимо разработать **реляционную схему**, отражающую взаимосвязи сущностей и обеспечивающую согласованность данных для непротиворечивости информации.

На этапе **проектирования** осуществляется построение диаграммы ER-модели, определяющей ключи и отношения между таблицами. Далее структура приводится к нормальной форме (вплоть до 3НФ), что обеспечивает устранение дублирования и аномалий в записях. Например, данные об игроке хранятся в отдельной таблице Player, а в таблице Record указывается только внешний ключ player\_id. То же самое и с регионами – они хранятся в совершенно отдельной таблице по правилам нормализации.

Выбор СУБД PostgreSQL обусловлен наличием опыта работы с данной технологией - знание синтаксиса, возможностей целостных ограничений и возможностей SQL (триггеры, представления и функции, которые необходимы для данной работы как обязательные условия).

В рамках проектирования базы данных выполняется:

* построение даталогической модели в соответствии с требованиями предметной области;
* создание таблиц с первичными(PK) и внешними ключами(FK);
* определение ограничений целостности (NOT NULL, UNIQUE, FOREIGN KEY);
* реализация механизмов вставки и обновления данных;
* реализация функций, представлений и триггеров.

В результате разработанная база данных обеспечивает централизованное, нормализованное и логически связанное хранение информации о мировых рекордах *Touhou Project*, что позволит использовать её для аналитики, публикации данных и автоматического формирования рейтингов.Для данных будет использоваться один сайт сообщества[1], который помимо некоторых советов и терминологии игры также содержит раздел, где хранятся рекорды очень большой давности и с некоторых локальных сообществ, что позволит нам вывести более точные данные, касающиеся рекордов.

## ****2. Даталогическая модель****

**Построение ER-диаграммы**

На этапе проектирования базы данных формируется **ER-диаграмма** – схема базы данных, отражающая структуру данной базы и взаимосвязи между её сущностями(элементами). .  
По итогу выделяются следующие сущности:

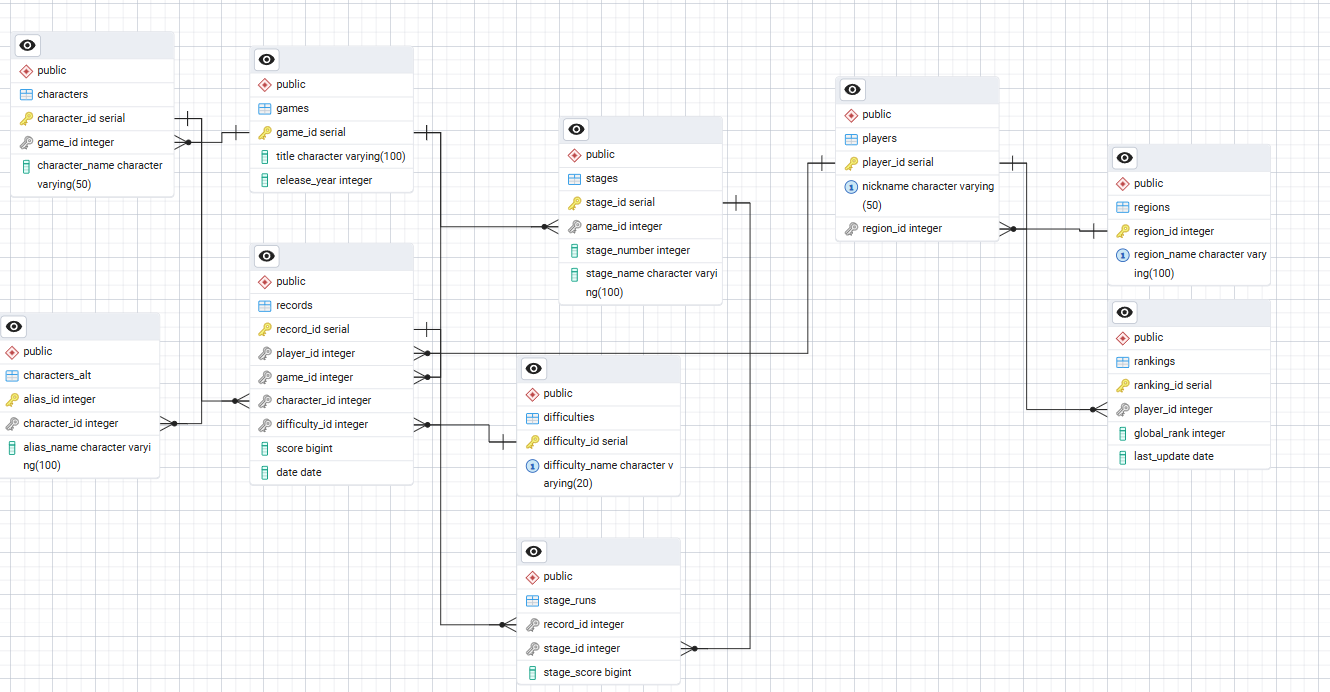


Рисунок 1.1- ER-диаграмма

**На рисунке 1.1 мы можем видеть все сущности диаграммы:**

* ***Players*** — игроки;
* ***Regions*** — регион, к которому принадлежит игрок.
* ***Games*** — части игры;
* ***Stages*** — уровни;
* ***Characters*** — официальные названия персонажей;
* ***Characters\_alt*** — фанатские названия персонажей;
* ***Difficulties*** — уровни сложности;
* ***Records*** — записи о рекордах;
* ***Stage\_runs*** — записи об информации рекорда(по уровням);
* ***Rankings*** — записи о рангах игроков.

**Определение связей между сущностями**

Связи между сущностями определяются на уровне внешних ключей(FK), что позволяет связать таблицы вместе и передавать значения. Основные зависимости диаграммы:

1. *Players→Regions*  
   Атрибут region\_id в *Players* – FK, указывающим на PK таблицы *Regions*.
2. *Stage\_runs→Records,Stages*  
   Имеет два FK ключа, указывающие на рекорд и уровни для составления счета данного рекорда.
3. *Records→Players, Games, Characters, Difficulties*Каждая запись рекорда связана с определенным игроком, частью, персонажем и сложностью.

Сущность Records – центральная часть реляционной схемы. Она связывает всю информацию в единое целое в своих записях. Помимо нее есть и Stage\_runs, позволяющая углубиться в детали набора очков рекорда.

**Нормализация данных**

Для обеспечения корректности и устранения аномалий избыточности данных используется процесс нормализации, который заключается в последовательном приведении структуры к нормальным формам. Для данной работы необходима 3НФ, из-за чего приняты следующие шаги:

*1НФ(Первая нормальная форма)*

Все атрибуты принимают атомарные значения, то есть – отсутствуют дубликаты(повторяющиеся записи).  
Пример: в таблице *Records* каждая строка представляет собой один рекорд, без хранения нескольких результатов в одной ячейке.

*2НФ (Вторая нормальная форма)*

Удаляются частичные зависимости от составного ключа, то есть – необходим только полный первичный ключ для создания зависимости.  
Пример - в таблице *Records* используется простой первичный ключ (record\_id), а значит что других зависимостей помимо него нет.  
Атрибуты, такие как score или date, зависят только от идентификатора записи.

*3НФ (Третья нормальная форма)*

Устраняются транзитивные зависимости, то есть атрибуты не должны зависеть друг от друга, кроме как от первичного ключа.  
Из-за этой формы регион игрока (region\_id) не хранится в таблице *Players* напрямую, а вынесена в отдельную таблицу *Regions(рисунок 1.2)*, что предотвращает дублирование данных и повышает согласованность.

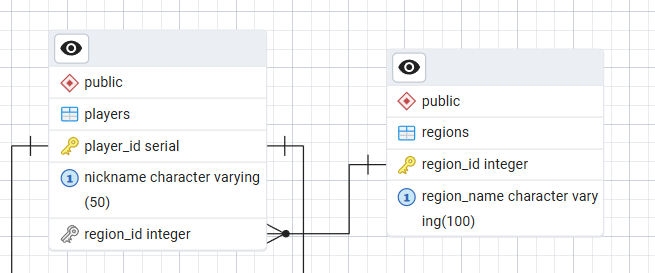


Рисунок 1.2 3НФ форма в ER-диаграмме

Результат - организованная структура, удовлетворяющая требованиям 3НФ, что обеспечивает целостность данных, то есть – меньше шансов теоретической потери данных и получения ошибки при работе с данными.

## ****3. Реализация базы данных****

**Список и описание таблиц**ы

Основные таблицы:

1. *Regions*
   * *region\_id* (PK, serial) - идентификатор региона;
   * *region\_name* (varchar) - название.
2. *players*
   * *player\_id* (PK, serial) – идентификатор игрока;
   * *nickname* (varchar(50)) - псевдоним игрока;
   * *region\_id* (FK) - ссылка на таблицу *regions*;
3. *games*
   * *game\_id* (PK, serial) – идентификатор игры;
   * *title* (varchar(100)) – название части;
   * *release\_year* (integer) – год выхода.
4. *characters*
   * *char\_id* (PK, serial) – идентификатор персонажа;
   * *char\_name* (varchar(50)) – его название;
   * *game\_id* (FK) – для определения из какой он части.
5. *characters\_alt*
   * *alias\_id* (PK, serial) – идентификатор альтернативного названия;
   * *character\_id* (FK) – ссылка на оффициального персонажа;
   * *alias\_name* (varchar(100)) – альтернативное название.
6. *difficulties*
   * *diff\_id* (PK, serial) – идентификатор сложности;
   * *name* (varchar(20)) — название сложности.
7. *stages*
   * *stage\_id* (PK, serial) – идентификатор уровня;
   * *game\_id* (FK) – ссылка на часть игры;
   * *stage\_name* (varchar(100)) – название стадии.
8. *records*
   * *record\_id* (PK) – идентификатор рекорда;
   * *player\_id* (FK) – ссылка на идентификатор игрока;
   * *game\_id* (FK) – ссылка на часть игры;
   * *char\_id* (FK) – ссылка на песронажа;
   * *diff\_id* (FK) – ссылка на сложность;
   * *score* (bigint) - итоговый счёт;
   * *date* (date) – дата рекорда.
9. *Stage\_runs*
   * *record\_id* (FK) – идентификатор рекорда;
   * *stage\_id* (FK) – ссылка на уровень;
   * *stage\_score* (bigint) – счёт уровня;
10. *rankings*
    * *rankings\_id* (PK) – идентификатор ранга;
    * *player\_id* (FK) – ссылка на идентификатор игрока;
    * *global\_rank* (int) – ранг игрока;
    * *last\_update\_date* (date) – дата обновления таблицы лидеров.

**Первичные и внешние ключи**

* *Первичные ключи (PK)* – уникальный номер каждой записи, который НЕ должен повторяться в таблице. В данной базе имеется:

player\_id, game\_id, record\_id, stage\_id, character\_id, ranking\_id, region\_id, difficulty\_id.

* *Внешние ключи (FK)* – ключи, которые ссылаются на другую таблицы. Так что все PK, которые были названы выше могут быть FK, но уже в другой таблице.

Благодаря этим ключам таблицы связаны между собой, а также при удалении записи в одной таблице удалятся и те, которые ссылались на эту запись.

**Индексы**

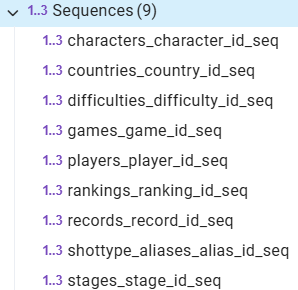
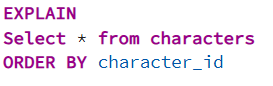


Рисунок 2.1 - Индексы таблиц

*Индексы* необходимы для более быстрого выполнения запроса. В данной таблице множество индексов ссылаются на PK таблиц(рисунок 2.1). При помощи EXPLAIN можно узнать время запроса(рисунок 2.2) для дальнешей его оптимизации, если это необходимо.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 2.2 Выполнение запроса с EXPLAIN

Индексы хорошо помогают в случае запросов с множеством условий а также если данных много. В нашей базе данных имеется 1000+ записей, поэтому использование индексов оправданно.

**Представления (VIEW)**

*Представления* позволяют объединить данные из нескольких таблиц. Пример – у меня имеются две таблицы с официальными и неофициальными названиями персонажей. При помощи представления я могу их объединить в одну таблицу:

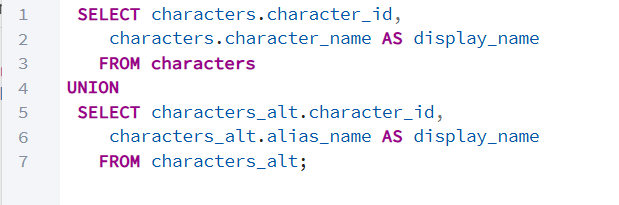


Рисунок 2.3 – Представление characters\_names

И теперь при запросе вне зависимости от того укажу ли я официальное название или нет(рисунок 2.3) – скрипт все равно найдет нужную запись и выведет ее.

**Процедуры, функции и триггеры**

Функции, процедуры и триггеры можно использовать для автоматизации.

Пример триггера(рисунок 2.4) – при вставке данных игроков данные добавляются в подсчёт таблицы лидеров и автоматически обновляются:

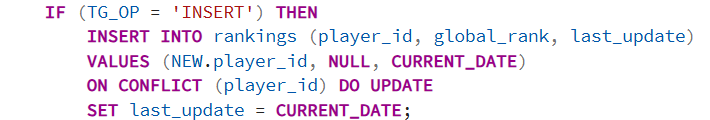


Рисунок 2.4 – Триггер update\_rankings

Пример функции(рисунок 2.5) – при вставке в аргумент псевдонима игрока выводятся все его рекорды(рисунок 2.6):

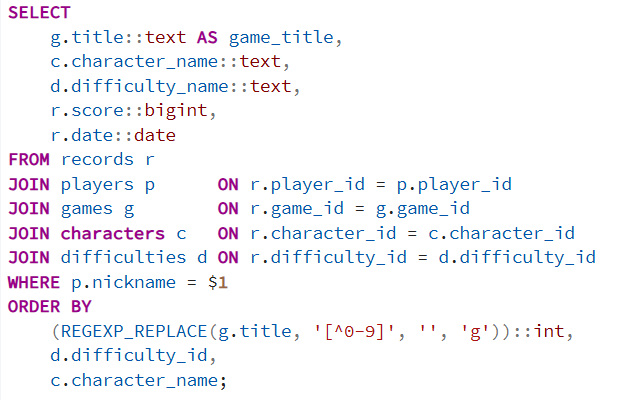


Рисунок 2.5 – Функция player\_records

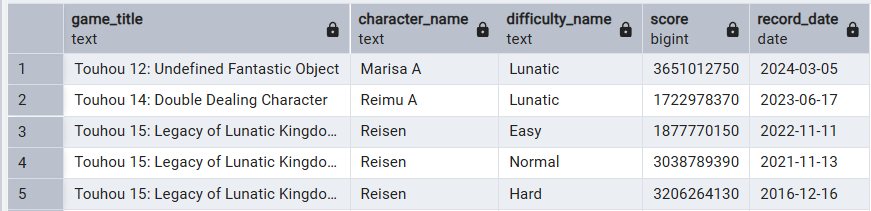


Рисунок 2.6 – Рекорды игрока RB

**Примеры SQL-запросов**

1. **Вывод списка рекордов 6-ой части:**

SELECT

g.title::text,

c.character\_name::text,

d.difficulty\_name::text,

p.nickname::text,

r.score::bigint,

r.date::date

FROM records r

JOIN games g ON r.game\_id = g.game\_id

JOIN characters c ON r.character\_id = c.character\_id

JOIN difficulties d ON r.difficulty\_id = d.difficulty\_id

JOIN players p ON r.player\_id = p.player\_id

WHERE g.title ILIKE 'Touhou 6%'

ORDER BY

(REGEXP\_REPLACE(g.title, '[^0-9]', '', 'g'))::int,

d.difficulty\_id,

c.character\_name;

1. **Вывод самого большого рекорда по каждой части:**

SELECT g.title, MAX(r.score) AS best\_score

FROM records r

JOIN games g ON r.game\_id = g.game\_id

GROUP BY g.title;

1. **Подсчёт количества рекордов по регионам:**

SELECT reg.region\_name, COUNT(r.record\_id) AS total\_records

FROM records r

JOIN players p ON r.player\_id = p.player\_id

JOIN regions reg ON p.region\_id = reg.region\_id

GROUP BY reg.region\_name

ORDER BY total\_records DESC;

1. **Выборка лучших результатов за текущий год:**

SELECT p.nickname, g.title, r.score, r.date

FROM records r

JOIN players p ON r.player\_id = p.player\_id

JOIN games g ON r.game\_id = g.game\_id

WHERE EXTRACT(YEAR FROM r.date) = EXTRACT(YEAR FROM CURRENT\_DATE)

ORDER BY r.score DESC

LIMIT 10;

1. **Клиентское приложение**

**Архитектура программы**

Разработанное клиентское приложение представляет собой систему для подключения к базе данных и дальнейшего выполнения запросов. Приложение могут использовать как и модераторы, которые могут внести необходимые правки в базу данных(рисунок 3.6), так и сами игроки, которые могут просматривать все рекорды, но не могут как-то изменять таблицу(рисунок 3.3). Приложение состоит из:

1. *Окно входа* – позволяет пользователю ввести название базы, порт, данные от своего аккаунта и подключится к базе данных (рисунок 3.1).
2. *Основное окно приложения* – имеет две готовые опции запроса - по игроку и по части с полем ввода данных(рисунок 3.4). Помимо этого, пользователь может сам написать запрос если ему необходимы свои условия выборки. Имеет кнопку сохранения результата в csv файл(рисунок 3.2).
3. *Окно вывода* – выводит пользователю результат запроса, который он может в дальнейшем сохранить(рисунок 3.5).

**Язык программирования и используемые библиотеки**

Клиентское приложение реализовано на языке *Python*, выбранным за счёт навыков работы с ним, а также его универсальности и широким выбором библиотек, совместимых с технологией PostgreSQL.

В проекте использовались следующие ключевые библиотеки:

* *SQLAlchemy* – обеспечивает саму работу с базой данных.  
  Позволяет подключится и в дальнейшем создавать запросы к базе(как автоматизированы так и ручные).
* *Pandas* — используется для отображения полученного результата, а также дальнейшего сохранения в csv файл.
* *Tkinter* — стандартная библиотека Python для создания интерфейсов.  
  Используется в приложении для окна авторизации, основного окна приложения и дополнительных элементов такие как кнопки и полей ввода.

В итоге приложение получилось довольно минималистичным и простым в использовании, чтобы каждый пользователь смог разобраться в его функционале.

**Скриншоты интерфейса**

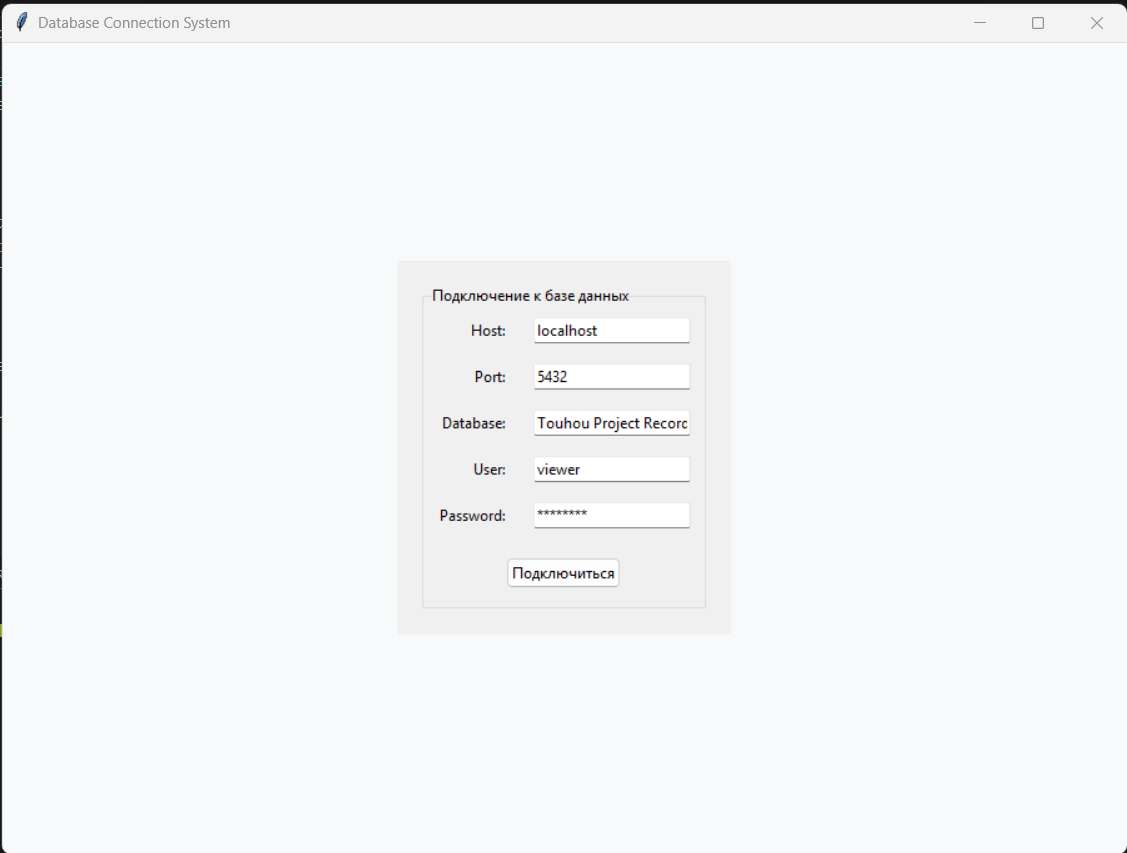


Рисунок 3.1 – Окно входа

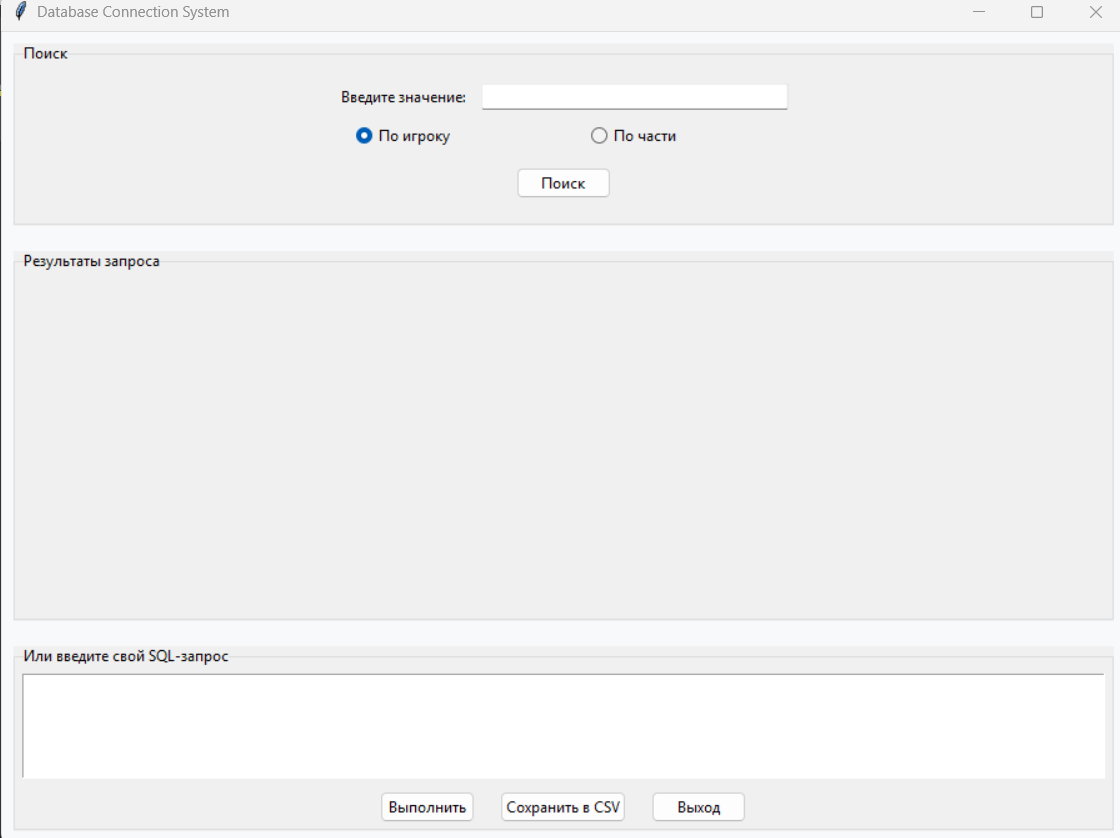


Рисунок 3.2 – Основное окно приложения

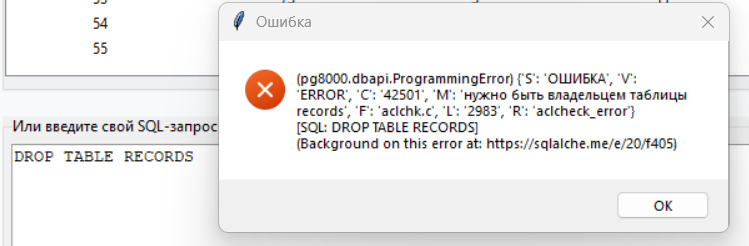


Рисунок 3.3 – Попытка изменения за игрока

**Примеры взаимодействия с базой данных**

* 1. Вход за игрока и просмотр рекордов моих рекордов:

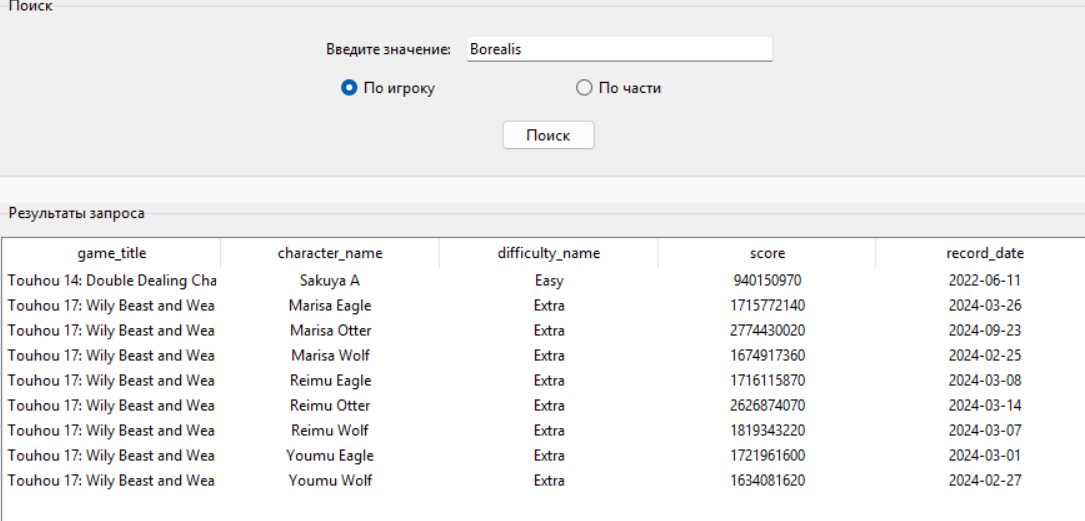
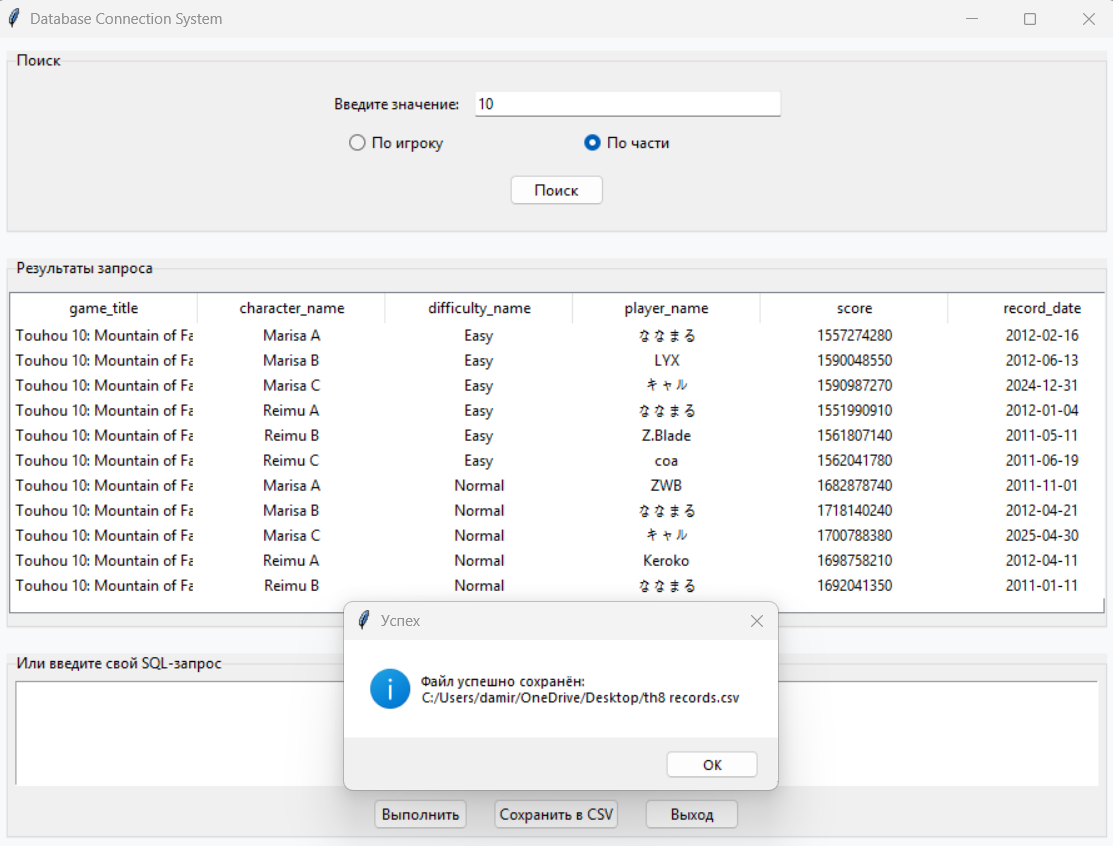


Рисунок 3.4 – Запрос по игроку Borealis

* 1. Сохранение таблицы в csv:



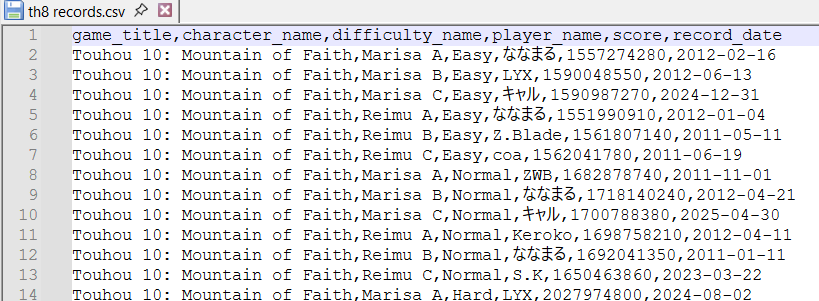
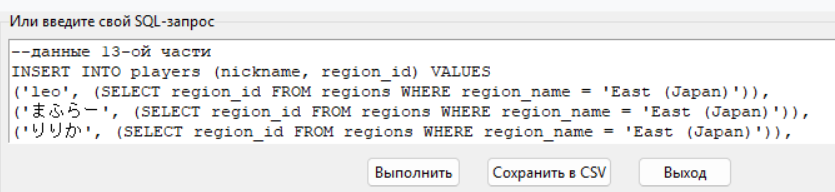


Рисунок 3.5 – Результат выборки и csv файл рекордов 10-ой части

* 1. Внесение правок за модератора:

Имеется скрипт для вставки новых рекордов 13-ой части(в конце стоит RETURNING \* для вывода результата запроса):



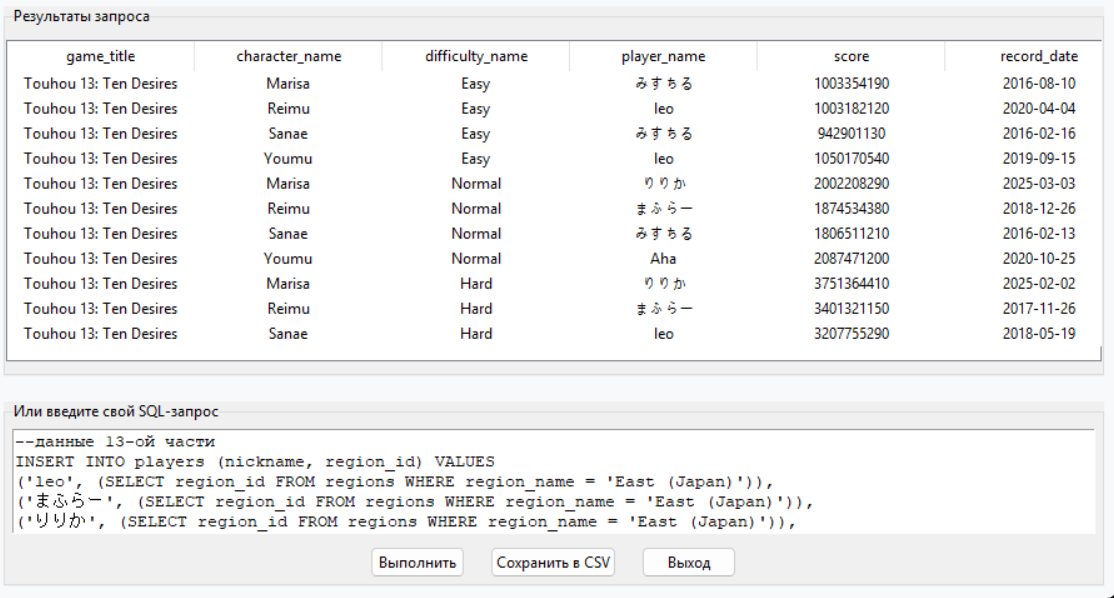


Рисунок 3.6 – Вставка рекордов 13-ой части

Любой запрос который можно выполнить в PostgreSQL можно выполнить в клиентском приложении.

**5. Использование искусственного интеллекта как помощника**

**Этапы применения ИИ**

В процессе разработки проекта искусственный интеллект применялся на нескольких этапах:

1. *Этап проектирования даталогической модели* **–** в основном только для проверки на нормализацию – благодаря нему выделили таблицу регионов в отдельную, иначе записи о регионах зависели бы от записей игроков.
2. *Заполнение таблиц данными –* несмотря на то что все данные были реальными, это все равно заняло бы очень много времени заполнения вручную. Поэтому ИИ выступал как приложение парсинг(рисунок 7.2) для получения данных из сайта и дальнейшей их записи.
3. *Отладка кода клиентского приложения -* ИИ помог в корректировке синтаксиса SQLAlchemy и устранении ошибок при подключении к PostgreSQL(рисунок 7.3). Кроме того, были предложены решения для автоматизации обновления рейтингов через триггеры и функции на PL/pgSQL.

**Конкретные примеры подсказок**

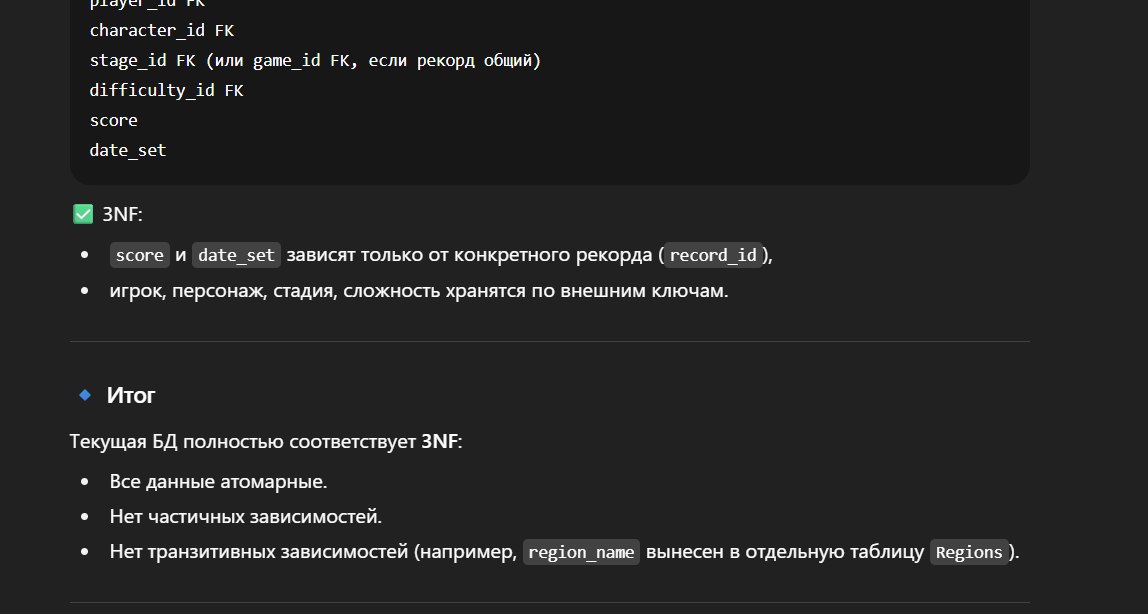


Рисунок 4.1 – Проверка на нормализацию 3НФ

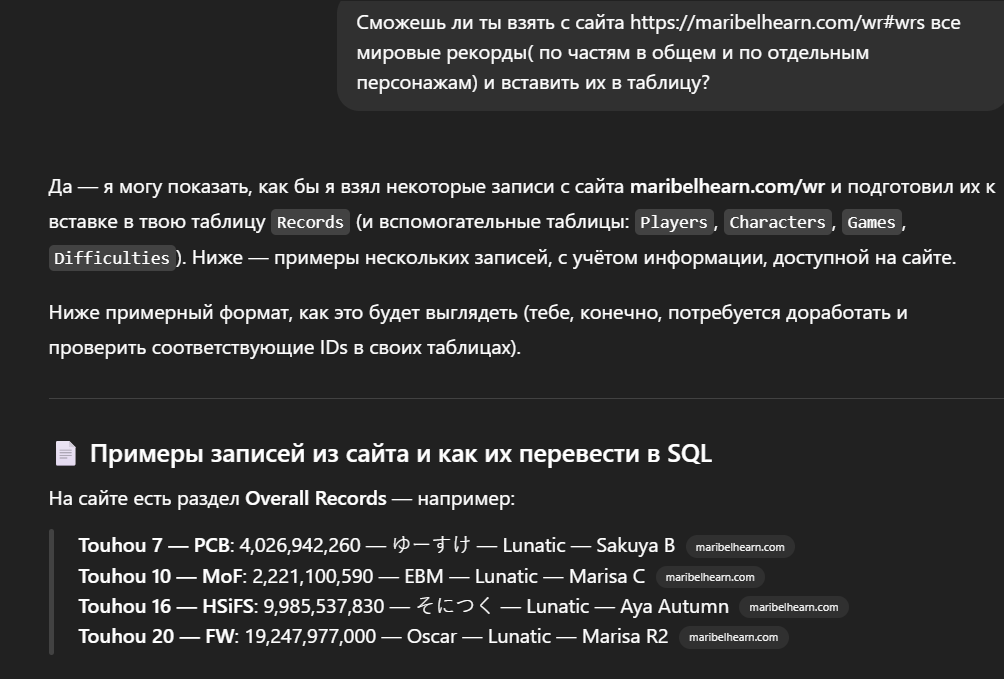
**

Рисунок 4.2(а) – Поставлен вопрос о скрапинге

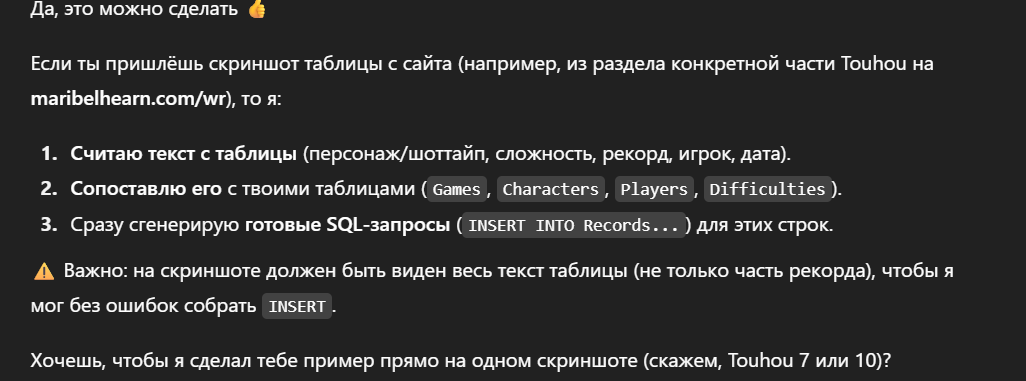


Рисунок 4.2(б) – Использование ИИ для скрапинга

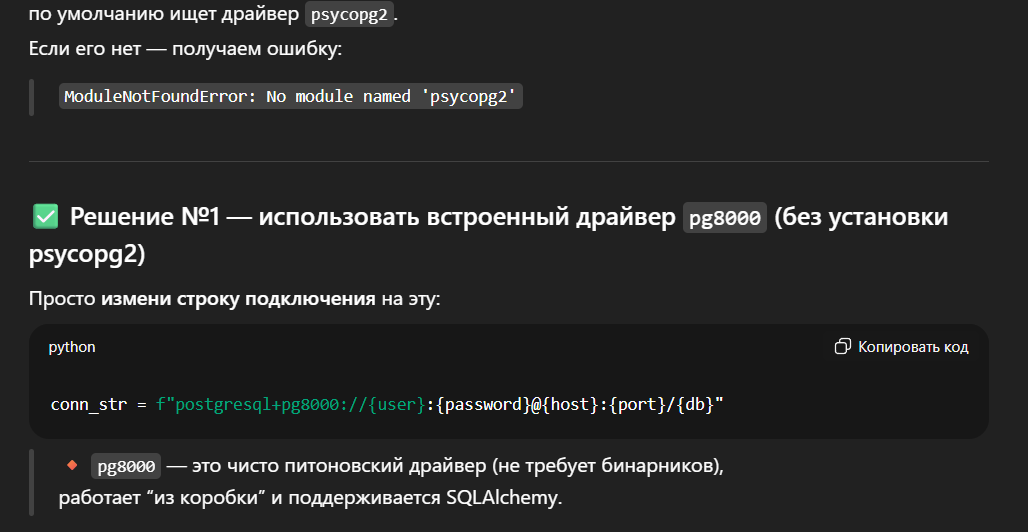


Рисунок 4.3 – Отладка ошибки клиентского приложения

**Результаты применения**

Применение искусственного интеллекта позволило значительно уменьшить потраченное время на работу, повысить эффективность работы и его решение.  
В частности:

* Была проведена успешная проверка нормализации 3НФ;
* ОЧЕНЬ большое время работы было уменьшено из-за использования ИИ как скрапера изображений;
* Решилась появившаяся ошибка при подключении к базе данных;
* Предоставила несколько стилистических подсказок при написании отчёта.

В данной работе ИИ выступил скорее не как автономный инструмент, задуманный сделать всю работу, а как современный интерактивный помощник, повышающий скорость и качество итоговой работы. Помимо этого, развились навыки prompt engineering и появился опыт работы с актуальными технологиями.

**Выводы об эффективности**

В ходе работы была доказана практическая эффективность использования искусственного интеллекта при разработке базы данных и клиентского приложения при наличии технического опыта работы с нейросетями.  
ИИ способствует:

* повышению производительности работы;
* сокращению числа ошибок при проектировании и программировании;
* формирование и проверка отчёта в научном стиле.

Интеграция ИИ является довольно серьезным процессом при разработке и практической работе. Однако не стоит забывать что на одном искусственном интеллекте работа сама себя не завершит, необходим определенный контроль над его работой и дополнительная проверка. Самое оптимальное решение – применение его возможностей в меру, но и не полностью отказываться от него.  
Использование искусственного интеллекта должно не заменять , а дополнять возможности разработчика.

**6. Заключение**

**Итоги работы**

По итогам практической работы была создана система хранения и обработки данных, предназначенная для ведения и анализа мировых рекордов по серии игр *Touhou Project*. Проект позволил продемонстрировать применение методов проектирования баз данных и практическое использование языка SQL в совокупности с некоторыми библиотеками Python для взаимодействия с PostgreSQL. База данных стабильно функционирует, как и клиентское приложение. Все необходимое было отправлено в соответствующий репозиторий[6], и соответствующим образом оформлено.

**Достижение целей и задач**

После выполнения поставленной цели можно заявить, что выполнены следующие задачи:

* проведен анализ и построена даталогическая модель;
* определены ключевые атрибуты таблиц;
* нормализована структура базы данных;
* реализованы основные представления, функции и триггеры;
* распределены права доступа между пользователями;
* разработано клиентское приложение для работы с базой данных.

Цели и задачи работы были достигнуты в полном объёме. Полученная база данных обеспечивает надёжное хранение, структурированный доступ, а также имеет возможность анализа и обработки информации о рекордах игроков.

**Перспективы дальнейшего развития проекта**

Проект можно расширить, так как не все потенциальные идеи были реализованы. Помимо этого появляется все больше рекордов и создаются новые части со своими категориями. Возможные направления проекта:

1. Продолжить запись рекордов по данной франшизе, но с более углубленными возможностями – показ того какие стратегии использовал игрок, допущенные ошибки и моменты которые можно было оптимизировать.
2. Расширить базу уже на другие франшизы, оставив текущую структуру базы данных и адаптации новых записей под нее.

Оба направления требуют гораздо больше времени на реализацию и имеют свою масштабируемость – первое будет увеличиваться в сообществе Touhou Project, а другое – среди остальных франшиз.

**7. Список источников**

Оформлено по ГОСТ 7.1–2003.

1. **MaribelHearn** – данные о рекордах по всем частям франшизы: [https://www.maribelhearn.com/wr/](https://www.postgresql.org/docs/)
2. **PostgreSQL Documentation –** для создания таблиц, заполнения и запросов: <https://www.postgresql.org/docs/>
3. **SQLAlchemy Documentation** – для подключения к базе данных и выполнения запросов: https://docs.sqlalchemy.org/
4. **pandas Documentation** – для экспорта данных из клиентского приложения: https://pandas.pydata.org/docs
5. **Git Documentation –** для работы с GitHub и создания репозитория:  
   <https://git-scm.com/doc>
6. **Репозиторий выполненный работы –** хранение бэкапа, клиентского приложения, отчёта: https://github.com/Damirbek45/TH-WR-project