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

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ (ИУ)
КАФЕДРА ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ (ИУ7)

КУРСОВАЯ РАБОТА

НА ТЕМУ:

Разработка базы данных для хранения и обработки данных сайта, посвященного сериалам

Студент ИУ7-66Б _____ Жаворонкова А.А.
(группа) (подпись, дата) (И.О. Фамилия)
Руководитель курсового проекта _____ Кострицкий А.С.
(подпись, дата) (И.О. Фамилия)

2024 г.

РЕФЕРАТ

Расчетно-пояснительная записка 40с., 10 рис., 16 табл., 19 источн., 1 прил.

Ключевые слова: база данных, веб-приложение, PostgreSQL, SQL, Golang.

Цель работы — разработка базы данных для хранения и обработки данных сайта, посвященного сериалам.

Результат работы — разработанная база данных, веб-приложение для доступа к базе данных.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Просмотр сериалов является популярным времяпрепровождением [1]. В связи с этим все больше людей обращаются к сайтам, посвященным сериалам, чтобы получить информацию о новинках, рейтингах, актерском составе и других интересующих их аспектах. Число выпускаемых сериалов увеличивается с каждым годом, и подобным сайтам необходимо хранить все больше информации и, как следствие, оптимизировать поиск информации в базах данных.

Целью данной работы является разработка базы данных для хранения и обработки данных сайта, посвященного сериалам.

Для достижения поставленной цели необходимо решить следующие задачи:

— провести сравнительный анализ известных решений;

— формализовать задачу и информацию, которая будет храниться в проектируемой базе данных;

— спроектировать базу данных;

— реализовать программное обеспечение, предоставляющее интерфейс для доступа к базе данных;

— исследовать влияние наличия индекса на время выполнения запроса поиска в базе данных.

1 Аналитический раздел

В данном разделе будет проведен анализ предметной области, анализ известных решений, будет формализована поставленная задача, а также данные, подлежащие хранению в проектируемой базе данных.

1.1 Анализ предметной области

Сериал — многосерийный фильм с несколькими сюжетными линиями. Существует широкий спектр жанров сериалов, включая драму, комедию, фантастику, ужасы, триллеры и многое другое. Каждый жанр имеет свои особенности и привлекает определенную аудиторию.

Некоторые сериалы становятся популярными и получают высокие оценки от зрителей и критиков. Рейтинг сериала может быть определен на основе оценок, просмотров, наград и обсуждений в СМИ. Помимо оценок, зрители могут оставлять отзывы, доступные другим зрителям, о просмотренных сериалах.

Сериалы могут иметь различное количество серий (эпизодов) и сезонов. Продолжительность отдельной серии и их количество в сезоне также различно для каждого сериала.

В сериалах задействованы актеры, которые играют роли различных персонажей. Некоторые сериалы имеют известных актеров, которые привлекают зрителей своим именем и талантом.

Сериалы требуют профессиональной работы съемочной группы, включая режиссера, оператора, художника по костюмам, декоратора и других специалистов. Технические аспекты, такие как качество съемки, монтаж, спецэффекты и звук, играют важную роль в создании атмосферы сериала.

1.2 Анализ известных решений

Для сравнения были выбраны следующие критерии:

1. поиск сериала с указанными параметрами;
2. наличие истории просмотров;
3. добавление сериала в избранное;

4. возможность сравнения выбранных сериалов. Под сравнением понимается просмотр таблицы, в которой столбцами являются названия выбранных сериалов, а строками — их характеристики, такие как жанр, дата выхода, рейтинг и т. д.;

5. просмотр общей длительности сериала.

В данной работе не рассматриваются такие решения как КиноПоиск, IVI, ОККО и т. д., поскольку они являются онлайн-кинотеатрами. Данные сервисы содержат информацию о сериалах, но их главная задача — обеспечение возможности просмотра видео. Поэтому для анализа существующих решений были выбраны сервисы наиболее схожие по функционалу с разрабатываемым решением.

В таблице 1.1 представлены результаты сравнения существующих решений.

Таблица 1.1 – Таблица сравнения существующих решений

Решение	Критерий				
	Поиск	История просмотров	Добавление в избранное	Сравнение	Общая длительность
myshows [2]	+	−	+	−	+
shikimori [3]	+	−	+	−	−
tvguru [4]	+	−	+	−	−
kinorium [5]	+	+	+	−	+

Из таблицы 1.1 видно, что ни одно из существующих решений не удовлетворяет всем критериям.

1.3 Формализация задачи

В рамках курсовой работы необходимо разработать базу данных для хранения и обработки данных сайта, посвященного сериалам, а также реализовать веб-приложение, обеспечивающее к ней доступ.

Приложение должно работать по модели клиент-сервер. Серверная часть должна обеспечивать обработку запросов от клиента, а также доступ к базе

данных. Клиентская часть должна предоставлять интерфейс для отправки таких запросов.

Разрабатываемая база данных должна содержать информацию о сериалах, их сезонах и эпизодах, актерах, режиссерах, а также о зарегистрированных пользователях.

На уровне базы данных должны быть реализованы 3 роли: гость (неавторизованный пользователь), авторизованный пользователь, администратор. На рисунке 1.1 представлены возможности каждой роли.

Рисунок 1.1 – Диаграмма вариантов использования

1.4 Формализация данных

Разрабатываемая база данных должна состоять из следующих сущностей:

— сериал;

— сезон;

— серия;

— актер;

— режиссер;

— пользователь;

— отзыв;

— избранное.

На рисунке 1.2 представлена ER-диаграмма сущностей проектируемой базы данных в нотации Чена.

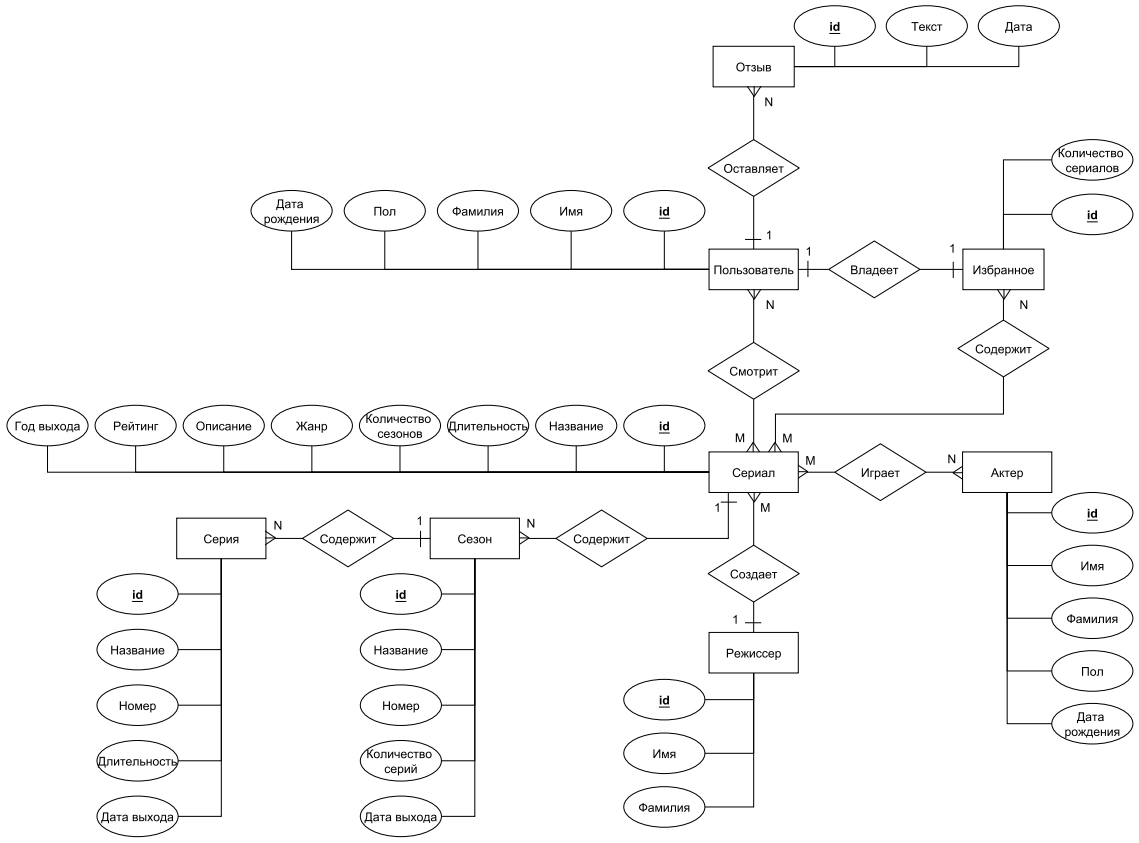

Рисунок 1.2 – ER-диаграмма сущностей в нотации Чена

1.5 Анализ существующих баз данных

Модель данных — это сововкупность правил порождения структур данных в базе данных, операций над ними, а также ограничений целостности, определяющих допустимые связи и значения данных, последовательность их изменения [6].

Выделяют следующие модели баз данных [7]:

1. иерархические;
2. сетевые;
3. реляционные;
4. объектно-ориентированные.

1.5.1 Иерархическая модель

Организация данных в иерархических моделях определяется в следующих терминах [8]:

— атрибут — наименьшая единица структуры данных;

— запись — именованная совокупность атрибутов;

— групповое отношение — иерархическое отношение между записями двух типов.

В иерархической модели данные организованы в виде древовидного графа с записями в виде узлов и множествами в виде ребер [7]. У каждого потомка может быть только один родитель, в тоже время у родителя может быть большое количество потомков. В связи с этим данная модель не поддерживает отношение «многие ко многим».

Уникальное значение ключа обязательно должно присутствовать в корневой записи каждого дерева. Ключи записей, не являющихся корневыми, должны быть уникальными только в пределах своего родителя. Каждая запись идентифицируется полным сцепленным ключом, который представляет собой уникальную комбинацию ключей всех записей от корня по иерархическому пути [8].

Для каждой некорневой записи в базе данных должна быть определена родительская запись. При удалении родительской записи все ее дочерние записи будут автоматически удалены.

1.5.2 Сетевая модель

В сетевой модели данные организованы в виде сети, где каждый элемент может быть связан с несколькими другими элементами. Такой подход к

организации данных является расширением иерархического. Таким образом, сетевая модель позволяет организовывать базы данных, структура которых представляется графом общего вида [6].

Как и в иерархической модели обеспечивается только поддержание целостности по ссылкам [9].

1.5.3 Реляционная модель

В реляционной модели данные организованы в виде таблиц (отношений) и все операции над базой данных сводятся к манипулированию таблицами [7]. Отношение отражает тип объекта реального мира (сущность). Каждое отношение состоит из строк (кортежей) и столбцов (атрибутов).

Основными свойствами отношений являются [10]:

— в каждом кортеже отношения отсутствуют дубликаты, что обусловлено наличием у каждого кортежа первичного ключа. Для каждого отношения, по крайней мере, все его атрибуты вместе образуют первичный ключ;

— порядок атрибутов не имеет значения, поэтому для обращения к значениям атрибутов используются их имена;

— значения атрибутов являются атомарными, то есть не могут содержать в себе множества значений (отношения).

В реляционной модели данных существуют два основных требования целостности [11]:

1. целостность сущностей: каждый кортеж в отношении должен быть уникальным, то есть отличаться от всех остальных кортежей в этом отношении. Для этого каждое отношение должно иметь первичный ключ;

2. целостность ссылок: каждое значение внешнего ключа, присутствующее в дочернем отношении, должно иметь соответствующее значение первичного ключа в родительском отношении.

1.5.4 Объектно-ориентированная модель

Структура объектной модели описываются с помощью трех ключевых понятий [12]:

1. инкапсуляция: каждый объект содержит некоторое внутреннее состояние и методы доступа к этому состоянию. Объекты представляют собой автономные сущности, отделенные от внешнего мира;

2. наследование: возможность создания новых классов объектов на основе существующих классов, унаследовав их структуру и методы, при этом добавляя свои собственные черты;

3. полиморфизм: различные объекты могут по-разному обрабатывать одни и те же события в зависимости от их методов.

В объектно-ориентированной модели данные представлены в виде объектов с атрибутами и методами, что позволяет более гибко организовывать данные и использовать принципы объектно-ориентированного программирования [7].

Для обеспечения целостности объектно-ориентированный подход предлагает следующие средства [12]:

— автоматическое управление наследованием;

— возможность объявления некоторых полей данных и методов объекта доступными только самому объекту;

— создание процедур контроля целостности внутри объекта.

1.6 Вывод

Таким образом, была выбрана реляционная модель базы данных, поскольку реляционная модель позволяет хранить данные в виде таблиц, что обеспечивает четкую структуру и организацию данных. Также реляционные базы данных поддерживают механизмы целостности данных, такие как ограничения целостности и транзакции, что обеспечивает защиту данных от ошибок и нежелательных изменений.

2 Конструкторский раздел

В данном разделе будут описаны сущности, ролевая модель и хранимая функция проектируемой базы данных. Будет представлена диаграмма базы данных, а также описаны проектируемые ограничения целостности.

2.1 Описание сущностей проектируемой базы данных

На основании ER-диаграммы, представленной на рисунке 1.2, проектируемая база данных должна содержать следующие таблицы:

— serials: таблица сериалов;

— actors: таблица актеров;

— producers: таблица режиссеров;

— seasons: таблица сезонов;

— episodes: таблица серий;

— users: таблица пользователей;

— favourites: таблица избранного;

— comments: таблица комментариев;

— serials_users: таблица просмотренных пользователем сериалов;

— serials_actors: таблица актеров, учавствующих в сериале;

— serials_favourites: таблица сериалов в избранном.

На рисунке 2.1 представлена диаграмма проектируемой базы данных в нотации Мартина.

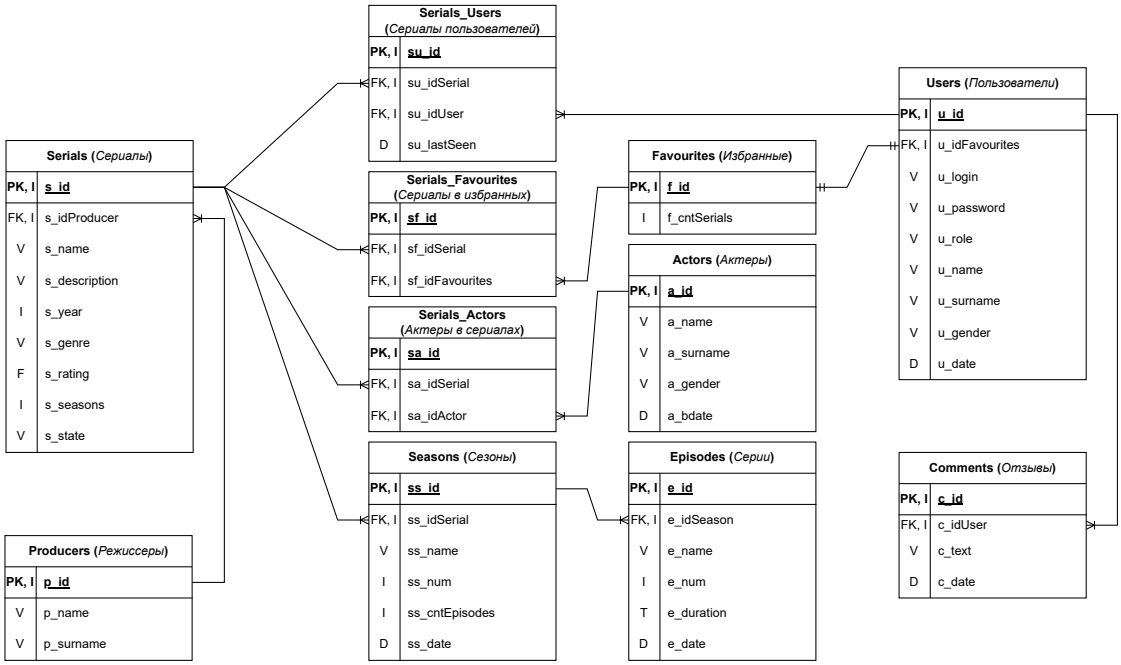

Рисунок 2.1 – Диаграмма базы данных

В таблицах 2.1 – 2.11 представлены имена столбцов каждой таблиц и описаны ограничения целостности. Поля всех таблиц не могут быть пустыми.

Таблица 2.1 – Таблица serials

Имя столбца	Описание	Ограничение
s_id	уникальный идентификатор сериала	является уникальным
s_idProducer	идентификатор режиссера сериала	—
s_name	название сериала	—
s_description	описание сериала	—
s_year	год выхода сериала	целое число, большее 1800
s_genre	жанр сериала	—
s_rating	рейтинг сериала	вещественное число от 0 до 10
s_seasons	количество сезонов в сериале	целое число, большее 0
s_state	статус сериала	принимает 2 возможных значения: «завершен», «продолжается»

Таблица 2.2 – Таблица actors

Имя столбца	Описание	Ограничение
a_id	уникальный идентификатор актера	является уникальным
a_name	имя актера	—
a_surname	фамилия актера	—
a_gender	пол актера	принимает 2 возможных значения: «мужской», «женский»
a_bdate	дата рождения актера	—

Таблица 2.3 – Таблица producers

Имя столбца	Описание	Ограничение
p_id	уникальный идентификатор режиссера	является уникальным
p_name	имя режиссера	—
p_surname	фамилия режиссера	—

Таблица 2.4 – Таблица seasons

Имя столбца	Описание	Ограничение
ss_id	уникальный идентификатор сезона	является уникальным
ss_idSerial	идентификатор сериала	—
ss_name	название сезона	—
ss_num	номер сезона	целое число, большее 0
ss_cntEpisodes	количество эпизодов в сезоне	целое число, не меньшее 0
ss_date	дата выхода сезона	—

Таблица 2.5 – Таблица episodes

Имя столбца	Описание	Ограничение
e_id	уникальный идентификатор серии	является уникальным
e_idSeason	идентификатор сезона	—
e_name	название серии	—
e_num	номер серии	целое число, большее 0
e_duration	продолжительность серии	—
e_date	дата выхода серии	—

Таблица 2.6 – Таблица users

Имя столбца	Описание	Ограничение
u_id	уникальный идентификатор пользователя	является уникальным
u_idFavourites	идентификатор избранного	—
u_login	логин пользователя	—
u_password	хеш пароля пользователя	—
u_role	роль пользователя	принимает 2 возможных значения: «user», «admin»
u_name	имя пользователя	—
u_surname	фамилия пользователя	—
u_gender	пол пользователя	принимает 2 возможных значения: «мужской», «женский»
u_date	дата рождения пользователя	—

Таблица 2.7 – Таблица favourites

Имя столбца	Описание	Ограничение
f_id	уникальный идентификатор избранного	является уникальным
f_cntSerials	количество сериалов в избранном	целое число, не меньшее 0

Таблица 2.8 – Таблица comments

Имя столбца	Описание	Ограничение
c_id	уникальный идентификатор комментария	является уникальным
c_idUser	идентификатор пользователя	—
c_text	текст комментария	—
c_date	дата оставления комментария	—

Таблица 2.9 – Таблица serials_users

Имя столбца	Описание	Ограничение
su_id	уникальный идентификатор сериалов, просмотренных пользователем	является уникальным
su_idSerial	идентификатор сериала	—
su_idUser	идентификатор пользователя	—
su_lastSeen	дата последнего просмотра	—

Таблица 2.10 – Таблица serials_actors

Имя столбца	Описание	Ограничение
sa_id	уникальный идентификатор актеров, учавствующих в сериале	является уникальным
sa_idSerial	идентификатор сериала	—
sa_idActor	идентификатор актера	—

Таблица 2.11 – Таблица serials_favourites

Имя столбца	Описание	Ограничение
sf_id	уникальный идентификатор сериалов в избранном	является уникальным
sf_idSerial	идентификатор сериала	—
sf_idFavourites	идентификатор избранного	—

2.2 Описание проектируемой ролевой модели

В предыдущем разделе были выделены 3 роли: гость (неавторизованный пользователь), авторизованный пользователь, администратор. В таблице 2.12 описаны возможности каждой роли. Введены следующие обозначения: C — CREATE, R — READ, U — UPDATE, D — DELETE.

Таблица 2.12 – Возможности ролей

Таблица	Роль		
	Гость	Авторизованный пользователь	Администратор
serials	R	R	C, R, U, D
actors	R	R	C, R, U, D
producers	R	R	C, R, U, D
seasons	R	R	C, R, U, D
episodes	R	R	C, R, U, D
users	C	R, U, D	R, U, D
favourites	-	R, U, D	R, U, D
comments	-	C, R, U, D	R
serials_users	-	C, R, D	-
serials_actors	-	-	C, R, U, D
serials_favourites	-	C, R, D	-

2.3 Описание проектируемой хранимой функции

В аналитическом разделе одним действий пользователя была возможность посмотреть сериалы из избранного, которые можно посмотреть за

выходные. Для реализации данного функционала было принято решение спроектировать функцию.

На рисунке 2.2 представлена схема алгоритма поиска сериалов из избранного, которые можно посмотреть за выходные.

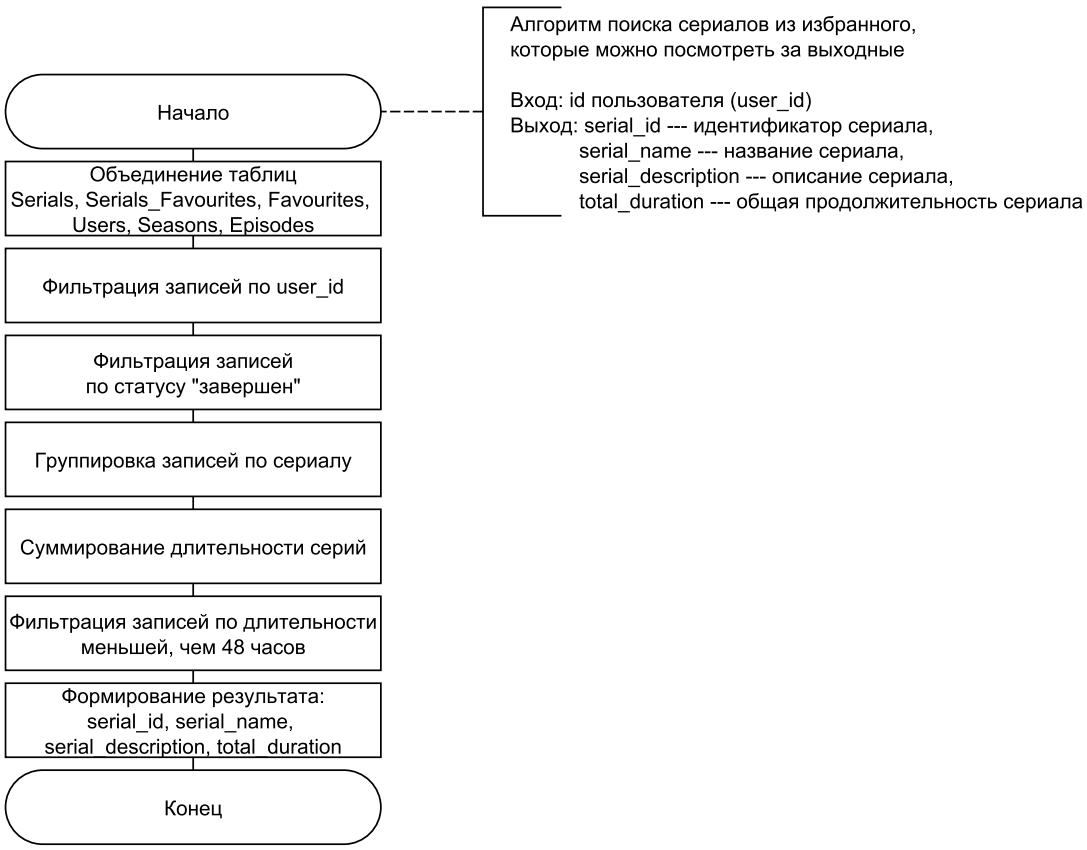

Рисунок 2.2 – Схема алгоритма поиска сериалов из избранного, которые можно посмотреть за выходные

2.4 Вывод

В данном разделе были описаны сущности, ролевая модель и хранимая функция проектируемой базы данных. Была представлена диаграмма базы данных, а также описаны проектируемые ограничения целостности.

3 Технологический раздел

В данном разделе будут обоснован выбор средств реализации, представлены листинги создания таблиц и ограничений целостности базы данных, создания ролей, реализации хранимой процедуры и способа ее тестирования, а также будут приведены примеры работы программы.

3.1 Выбор системы управления базами данных

Существует большое количество реляционных СУБД, и самые популярные из них это [13] — Oracle [14], MySQL [15], Microsoft SQL Server [16], PostgreSQL [17].

Для сравнения выбранных СУБД выделены следующие критерии:

— бесплатное распространение СУБД;

— производительность (на основе источника [18]);

— наличие опыта работы с СУБД.

Результаты сравнения выбранных СУБД по заданным критериям представлены в таблице 3.1.

Таблица 3.1 – Сравнение выбранных СУБД

СУБД	Критерий		
	Является бесплатной	Производительность	Наличие опыта работы с СУБД
Oracle	—	2	—
MySQL	+	3	—
Microsoft SQL Server	—	4	—
PostgreSQL	+	1	+

В качестве системы управления базами данных была выбрана PostgreSQL. Данная СУБД поддерживает все необходимые типы данных для реализации описанной выше базы данных, предоставляет возможность создания хранимых функций. Кроме того, PostgreSQL является свободным программным обеспечением (open source), доступна бесплатно и присутствует опыт работы с данной СУБД.

3.2 Средства реализации

Для написания серверной части приложения был выбран язык программирования Golang [19], так как он обладает всеми необходимыми инструментами для выполнения поставленных задач. Golang предоставляет функционал для создания простых одностраничных серверов, а также инструменты для взаимодействия с базой данных, разработанной с использованием PostgreSQL. В частности были использованы такие пакеты как «gorilla/mux», реализующий маршрутизатор запросов, «gorilla/sessions» для управления сессиями, «jmoiron/sqlx» для работы с базой данных, «sirupsen/logrus», обеспечивающий логирование, «x/crypto/bcrypt», позволяющий хранить пароли в зашифрованном виде, «net/http», предоставляющий клиентскую и серверную реализации HTTP, «BurntSushi/toml», использующийся для декодирования TOML файлов.

Для реализации клиентских веб-страниц выбран язык разметки HTML [20].

Для реализации хранимой процедуры использовалось процедурное расширение PL/pgSQL [21].

Для автоматизации развертывания и управления приложением использовался Docker [22].

В качестве среды разработки использовался Visual Studio Code [23].

3.3 Создание таблиц

В листингах 3.1 – 3.11 представлены создание таблиц и ограничений целостности базы данных.

Листинг 3.1 – Создание таблицы Serials

```
1  drop table if exists Serials cascade;
2  create table Serials (
3      s_id serial not null primary key,
4      s_idProducer int not null,
5    s_img text not null,
6      s_name text not null,
7      s_description text not null,
8      s_year int not null check (s_year > 1800),
9      s_genre text not null,
10     s_rating float not null check (s_rating between 0 and 10),
11     s_seasons int not null check (s_seasons >= 0),
12     s_state text not null check (s_state = 'завершен' or s_state
```

```
    = 'продолжается'),
  s_duration interval not null
);
alter table Serials add foreign key (s_idProducer) references
    Producers(p_id);
```

Листинг 3.2 – Создание таблицы Actors

```

drop table if exists Actors cascade;
create table Actors (
    a_id serial not null primary key,
    a_name text not null,
    a_surname text not null,
    a_gender text not null check (a_gender = 'мужской' or
    a_gender = 'женский'),
    a_bdate date not null
);
```

Листинг 3.3 – Создание таблицы Producers

```

drop table if exists Producers cascade;
create table Producers (
    p_id serial not null primary key,
    p_name text not null,
    p_surname text not null
);
```

Листинг 3.4 – Создание таблицы Seasons

```

drop table if exists Seasons cascade;
create table Seasons (
    ss_id serial not null primary key,
    ss_idSerial int not null,
    ss_name text not null,
    ss_num int not null check (ss_num > 0),
    ss_cntEpisodes int not null check (ss_cntEpisodes > 0),
    ss_date date not null
);
alter table Seasons add foreign key (ss_idSerial) references
    Serials(s_id);
```

Листинг 3.5 – Создание таблицы Episodes

```

drop table if exists Episodes cascade;
create table Episodes (
    e_id serial not null primary key,
    e_idSeason int not null,
```

```
    e_name text not null,
    e_num int not null check (e_num > 0),
    e_duration interval not null,
    e_date date not null
);
alter table Episodes add foreign key (e_idSeason) references
   Seasons(ss_id);
```

Листинг 3.6 – Создание таблицы Users

```

drop table if exists Users cascade;
create table Users (
    u_id serial not null primary key,
    u_idFavourites int not null,
    u_login text not null,
    u_password text not null,
    u_role text not null check (u_role = 'user' or u_role = '
   admin'),
    u_name text not null,
    u_surname text not null,
    u_gender text not null check (u_gender = 'мужской' or
   u_gender = 'женский'),
    u_bdate date not null
);
```

Листинг 3.7 – Создание таблицы Favourites

```

drop table if exists Favourites cascade;
create table Favourites (
    f_id serial not null primary key,
    f_cntSerials int not null check (f_cntSerials >= 0)
);
alter table Users add foreign key (u_idFavourites) references
   Favourites(f_id);
```

Листинг 3.8 – Создание таблицы Comments

```

drop table if exists Comments cascade;
create table Comments (
    c_id serial not null primary key,
    c_idUser int not null,
  c_idSerial int not null,
    c_text text not null,
    c_date date not null
);
alter table Comments add foreign key (c_idUser) references Users(
   u_id);
```

Листинг 3.9 – Создание таблицы Serials_Users

```
1
2 drop table if exists Serials_Users cascade;
3 create table Serials_Users (
4     su_id serial not null primary key,
5     su_idSerial int not null,
6     su_idUser int not null,
7     su_lastSeen date not null
8 );
9 alter table Serials_Users add foreign key (su_idSerial)
     references Serials(s_id);
10 alter table Serials_Users add foreign key (su_idUser) references
     Users(u_id);
```

Листинг 3.10 – Создание таблицы Serials_Actors

```
1
2 drop table if exists Serials_Actors cascade;
3 create table Serials_Actors (
4     sa_id serial not null primary key,
5     sa_idSerial int not null,
6     sa_idActor int not null
7 );
8 alter table Serials_Actors add foreign key (sa_idSerial)
     references Serials(s_id);
9 alter table Serials_Actors add foreign key (sa_idActor)
     references Actors(a_id);
```

Листинг 3.11 – Создание таблицы Serials_Favourites

```
1
2 drop table if exists Serials_Favourites cascade;
3 create table Serials_Favourites (
4     sf_id serial not null primary key,
5     sf_idSerial int not null,
6     sf_idFavourite int not null
7 );
8 alter table Serials_Favourites add foreign key (sf_idSerial)
     references Serials(s_id);
9 alter table Serials_Favourites add foreign key (sf_idFavourite)
     references Favourites(f_id);
```

3.4 Создание ролей базы данных

В листингах 3.12 – 3.14 представлены создание ролей базы данных.

Листинг 3.12 – Создание роли guest

```
1 create role guest with login password 'guest';
2 grant select on table Serials to guest;
```

```
grant select on table Seasons to guest;
grant select on table Episodes to guest;
grant select on table Actors to guest;
grant select on table Producers to guest;
grant insert on table Users to guest;
```

Листинг 3.13 – Создание роли regUser

```
create role regUser with login password 'regUser';
grant select on table Serials to regUser;
grant select on table Seasons to regUser;
grant select on table Episodes to regUser;
grant select on table Actors to regUser;
grant select on table Producers to regUser;
grant select, insert, update on table Users to regUser;
grant select, insert, update on table Comments to regUser;
grant select, insert, update on table Favourites to regUser;
grant select, insert on table Serials_Users to regUser;
```

Листинг 3.14 – Создание роли adminUser

```
create role adminUser with login password 'adminUser';
grant select, insert, update on table Actors to adminUser;
grant select, insert, update on table Comments to adminUser;
grant select, insert, update on table Episodes to adminUser;
grant select, insert, update on table Favourites to adminUser;
grant select, insert, update on table Producers to adminUser;
grant select, insert, update on table Seasons to adminUser;
grant select, insert, update on table Serials to adminUser;
grant select, insert, update on table Serials_Actors to adminUser
    ;
grant select, insert, update on table Serials_Favourites to
    adminUser;
grant select, insert, update on table Serials_Users to adminUser;
grant select, insert, update on table Users to adminUser;
```

3.5 Реализация хранимой функции

В листинге 3.15 представлена реализация хранимой функции, обеспечивающей поиск сериалов из избранного, которые можно посмотреть за выходные.

Листинг 3.15 – Реализация хранимой функции, обеспечивающей поиск сериалов из избранного, которые можно посмотреть за выходные

```
CREATE OR REPLACE FUNCTION get_weekend_serials(user_id INT)
RETURNS TABLE (
    serial_id INT,
    serial_name TEXT,
```

```
    serial_description TEXT,
    total_duration INTERVAL
) AS $$
BEGIN
    RETURN QUERY
    SELECT
        s.s_id,
        s.s_name,
        s.s_description,
        sum(e.e_duration) as total_duration
    FROM
        Serials s
        JOIN Serials_Favourites sf ON s.s_id = sf.sf_idSerial
        JOIN Favourites f ON sf.sf_idFavourite = f.f_id
        JOIN Users u ON u.u_idFavourites = f.f_id
        JOIN Seasons ss ON ss.ss_idSerial = s.s_id
        JOIN Episodes e ON e.e_idSeason = ss.ss_id
    WHERE
        u.u_id = user_id
        AND s.s_state = 'завершен'
    GROUP BY
        s.s_id, s.s_name, s.s_description
    HAVING
        sum(e.e_duration) <= INTERVAL '48 hours';
END;
$$ LANGUAGE plpgsql;
```

Тестирование функции выполнялось по алгоритму, представленному на рисунке 3.1.

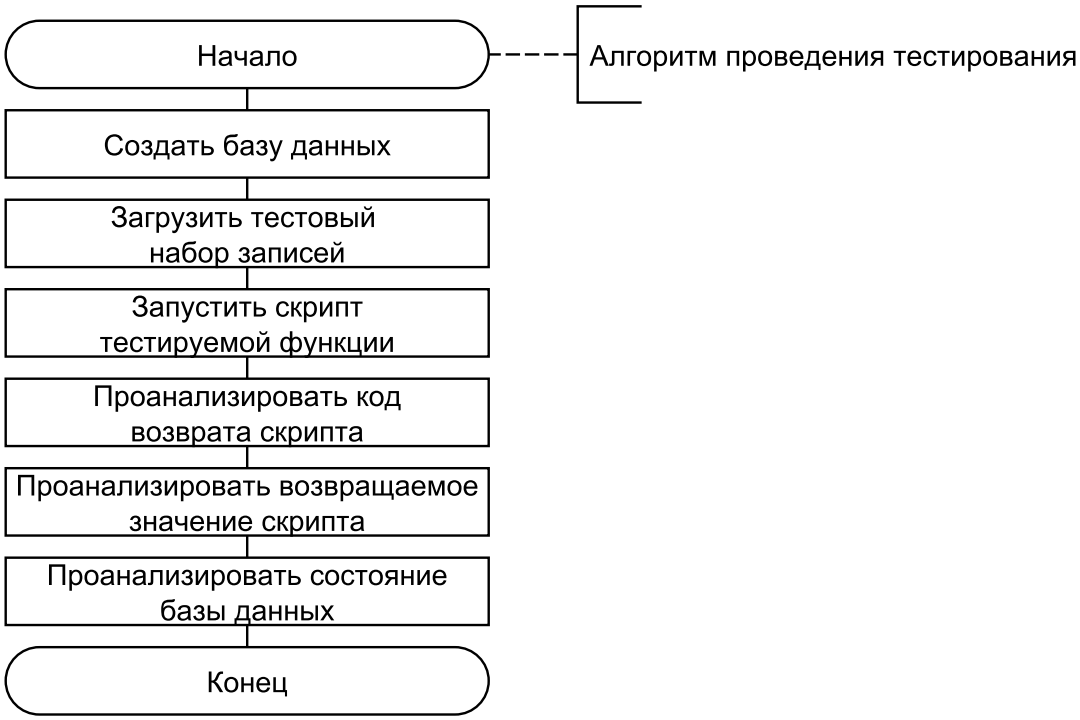

Рисунок 3.1 – Алгоритм тестирования хранимой функции

Код программы представлен в листинге 3.16.

Листинг 3.16 – Тестирование хранимой функции

```
import psycopg2
import pandas as pd
from faker import Faker
import random
from datetime import timedelta

def connect_to_db():
    try:
        conn = psycopg2.connect(
            dbname="serials",
            user="postgres",
            password="5454038",
            host="localhost",
            port="5432"
        )
        return conn
    except Exception as e:
        print(f"Error connecting to the database: {e}")
        return None

def setup_database(conn):
    try:
        cursor = conn.cursor()
```

```
24          setup_query = """
25          DROP TABLE IF EXISTS Serials CASCADE;
26          CREATE TABLE Serials (
27              s_id SERIAL NOT NULL PRIMARY KEY,
28              s_idProducer INT NOT NULL,
29              s_img TEXT NOT NULL,
30              s_name TEXT NOT NULL,
31              s_description TEXT NOT NULL,
32              s_year INT NOT NULL CHECK (s_year > 1800),
33              s_genre TEXT NOT NULL,
34              s_rating FLOAT NOT NULL CHECK (s_rating BETWEEN 0 AND
     10),
35              s_seasons INT NOT NULL CHECK (s_seasons >= 0),
36              s_state TEXT NOT NULL CHECK (s_state = 'завершен' OR
    s_state = 'продолжается'),
37              s_duration INTERVAL NOT NULL
38          );
39          DROP TABLE IF EXISTS Producers CASCADE;
40          CREATE TABLE Producers (
41              p_id SERIAL NOT NULL PRIMARY KEY,
42              p_name TEXT NOT NULL,
43              p_surname TEXT NOT NULL
44          );
45          ALTER TABLE Serials ADD FOREIGN KEY (s_idProducer)
    REFERENCES Producers(p_id);
46
47          DROP TABLE IF EXISTS Seasons CASCADE;
48          CREATE TABLE Seasons (
49              ss_id SERIAL NOT NULL PRIMARY KEY,
50              ss_idSerial INT NOT NULL,
51              ss_name TEXT NOT NULL,
52              ss_num INT NOT NULL CHECK (ss_num > 0),
53              ss_cntEpisodes INT NOT NULL CHECK (ss_cntEpisodes >
    0),
54              ss_date DATE NOT NULL
55          );
56          ALTER TABLE Seasons ADD FOREIGN KEY (ss_idSerial)
    REFERENCES Serials(s_id);
57
58          DROP TABLE IF EXISTS Episodes CASCADE;
59          CREATE TABLE Episodes (
60              e_id SERIAL NOT NULL PRIMARY KEY,
61              e_idSeason INT NOT NULL,
62              e_name TEXT NOT NULL,
63              e_num INT NOT NULL CHECK (e_num > 0),
64              e_duration INTERVAL NOT NULL,
65              e_date DATE NOT NULL
66          );
```

```
67          ALTER TABLE Episodes ADD FOREIGN KEY (e_idSeason)
    REFERENCES Seasons(ss_id);
68
69          DROP TABLE IF EXISTS Users CASCADE;
70          CREATE TABLE Users (
71              u_id SERIAL NOT NULL PRIMARY KEY,
72              u_idFavourites INT NOT NULL,
73              u_login TEXT NOT NULL,
74              u_password TEXT NOT NULL,
75              u_role TEXT NOT NULL CHECK (u_role = 'user' OR u_role
     = 'admin'),
76              u_name TEXT NOT NULL,
77              u_surname TEXT NOT NULL,
78              u_gender TEXT NOT NULL CHECK (u_gender = 'мужской' OR
     u_gender = 'женский'),
79              u_bdate DATE NOT NULL
80          );
81
82          DROP TABLE IF EXISTS Favourites CASCADE;
83          CREATE TABLE Favourites (
84              f_id SERIAL NOT NULL PRIMARY KEY,
85              f_cntSerials INT NOT NULL CHECK (f_cntSerials >= 0)
86          );
87          ALTER TABLE Users ADD FOREIGN KEY (u_idFavourites)
    REFERENCES Favourites(f_id);
88
89          DROP TABLE IF EXISTS Serials_Favourites CASCADE;
90          CREATE TABLE Serials_Favourites (
91              sf_id SERIAL NOT NULL PRIMARY KEY,
92              sf_idSerial INT NOT NULL,
93              sf_idFavourite INT NOT NULL
94          );
95          ALTER TABLE Serials_Favourites ADD FOREIGN KEY (
    sf_idSerial) REFERENCES Serials(s_id);
96          ALTER TABLE Serials_Favourites ADD FOREIGN KEY (
    sf_idFavourite) REFERENCES Favourites(f_id);
97          """
98          cursor.execute(setup_query)
99          conn.commit()
100         cursor.close()
101         print("Database setup completed.")
102     except Exception as e:
103         print(f"Error setting up the database: {e}")
104
105 def populate_database(conn):
106     fake = Faker()
107     try:
108         cursor = conn.cursor()
```

```
109
110         # Create producers
111         cursor.execute(
112             "INSERT INTO Producers (p_name, p_surname) VALUES (%s
    , %s) RETURNING p_id",
113             (fake.first_name(), fake.last_name())
114         )
115         producer_id = cursor.fetchone()[0]
116
117         # Create favourites
118         cursor.execute(
119             "INSERT INTO Favourites (f_cntSerials) VALUES (%s)
    RETURNING f_id",
120             (0,)
121         )
122         favourites_id = cursor.fetchone()[0]
123
124         # Create user
125         cursor.execute(
126             "INSERT INTO Users (u_idFavourites, u_login,
    u_password, u_role, u_name, u_surname, u_gender, u_bdate)
    VALUES (%s, %s, %s, %s, %s, %s, %s, %s) RETURNING u_id",
127             (favourites_id, fake.user_name(), fake.password(), '
    user', fake.first_name(), fake.last_name(), random.choice(['
    мужской', 'женский']), fake.date_of_birth(minimum_age=18,
    maximum_age=80))
128         )
129         user_id = cursor.fetchone()[0]
130
131         expected_results = []
132
133         # Create serials for each class of equivalence
134         # Class 1: 1 сериал
135         serial_id = create_serial(cursor, producer_id,
    favourites_id, fake, serial_name="Class 1 Serial", num_seasons
    =1, num_episodes=[5], duration_per_episode=[timedelta(minutes
    =20)])
136         expected_results.append((serial_id, "Class 1 Serial", "
    Class 1 Serial Description", timedelta(minutes=100)))
137
138         # Class 2: несколько сериалов
139         serial_id1 = create_serial(cursor, producer_id,
    favourites_id, fake, serial_name="Class 2 Serial 1",
    num_seasons=1, num_episodes=[5], duration_per_episode=[
    timedelta(minutes=20)])
140         serial_id2 = create_serial(cursor, producer_id,
    favourites_id, fake, serial_name="Class 2 Serial 2",
    num_seasons=2, num_episodes=[3, 3], duration_per_episode=[
```

```
    timedelta(minutes=30), timedelta(minutes=30)])
141         expected_results.append((serial_id1, "Class 2 Serial 1",
    "Class 2 Serial 1 Description", timedelta(minutes=100)))
142         expected_results.append((serial_id2, "Class 2 Serial 2",
    "Class 2 Serial 2 Description", timedelta(minutes=180)))
143
144         # Class 3: нет сериалов
145         # No action needed, the user has no serials that fit in
    the weekend time
146
147         # Class 4: сериал из 1 сезона и 1 серии
148         serial_id = create_serial(cursor, producer_id,
    favourites_id, fake, serial_name="Class 4 Serial", num_seasons
    =1, num_episodes=[1], duration_per_episode=[timedelta(minutes
    =40)])
149         expected_results.append((serial_id, "Class 4 Serial", "
    Class 4 Serial Description", timedelta(minutes=40)))
150
151         # Class 5: сериал из 1 сезона
152         serial_id = create_serial(cursor, producer_id,
    favourites_id, fake, serial_name="Class 5 Serial", num_seasons
    =1, num_episodes=[8], duration_per_episode=[timedelta(minutes
    =30)])
153         expected_results.append((serial_id, "Class 5 Serial", "
    Class 5 Serial Description", timedelta(minutes=240)))
154
155         conn.commit()
156         cursor.close()
157         print("Database population completed.")
158         return user_id, expected_results
159     except Exception as e:
160         print(f"Error populating the database: {e}")
161         return None, None
162
163 def create_serial(cursor, producer_id, favourites_id, fake,
    serial_name, num_seasons, num_episodes, duration_per_episode):
164     cursor.execute(
165         "INSERT INTO Serials (s_idProducer, s_img, s_name,
    s_description, s_year, s_genre, s_rating, s_seasons, s_state,
    s_duration) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
    RETURNING s_id",
166         (
167             producer_id,
168             fake.image_url(),
169             serial_name,
170             f"{serial_name} Description",
171             random.randint(1900, 2023),
172             random.choice(['Drama', 'Comedy', 'Action']),
```

```
173             round(random.uniform(0, 10), 1),
174             num_seasons,
175             'завершен',
176             '00:00:00'
177         )
178     )
179     serial_id = cursor.fetchone()[0]
180 
181     total_duration = timedelta()
182     for season_num in range(1, num_seasons + 1):
183         cursor.execute(
184             "INSERT INTO Seasons (ss_idSerial, ss_name, ss_num,
    ss_cntEpisodes, ss_date) VALUES (%s, %s, %s, %s, %s) RETURNING
     ss_id",
185             (
186                 serial_id,
187                 f"Season {season_num}",
188                 season_num,
189                 num_episodes[season_num - 1],
190                 fake.date_between(start_date='-5y', end_date='
    today')
191             )
192         )
193         season_id = cursor.fetchone()[0]
194 
195         for episode_num in range(1, num_episodes[season_num - 1]
    + 1):
196             episode_duration = duration_per_episode[season_num -
    1]
197             total_duration += episode_duration
198             cursor.execute(
199                 "INSERT INTO Episodes (e_idSeason, e_name, e_num,
     e_duration, e_date) VALUES (%s, %s, %s, %s, %s)",
200                 (
201                     season_id,
202                     f"Episode {episode_num}",
203                     episode_num,
204                     str(episode_duration),
205                     fake.date_between(start_date='-5y', end_date
    ='today')
206                 )
207             )
208 
209     cursor.execute(
210         "UPDATE Serials SET s_duration = %s WHERE s_id = %s",
211         (str(total_duration), serial_id)
212     )
213 
```

```
214     cursor.execute(
215         "INSERT INTO Serials_Favourites (sf_idSerial,
    sf_idFavourite) VALUES (%s, %s)",
216         (serial_id, favourites_id)
217     )
218 
219     return serial_id
220 
221 def export_db_to_excel(conn, filename):
222     try:
223         query = """
224         SELECT Serials.s_id, Serials.s_name, Serials.
    s_description, Serials.s_duration, Favourites.f_id
225         FROM Serials
226         JOIN Serials_Favourites ON Serials.s_id =
    Serials_Favourites.sf_idSerial
227         JOIN Favourites ON Serials_Favourites.sf_idFavourite =
    Favourites.f_id
228         """
229         df = pd.read_sql_query(query, conn)
230         df.to_excel(filename, index=False)
231         print(f"Database exported to {filename}")
232     except Exception as e:
233         print(f"Error exporting database to Excel: {e}")
234 
235 def call_stored_procedure(conn, user_id):
236     try:
237         cursor = conn.cursor()
238         cursor.callproc('get_weekend_serials', [user_id])
239         result = cursor.fetchall()
240         conn.commit()
241         cursor.close()
242         return result
243     except Exception as e:
244         print(f"Error calling stored procedure: {e}")
245         return None
246 
247 def compare_excel_files(file1, file2):
248     try:
249         df1 = pd.read_excel(file1)
250         df2 = pd.read_excel(file2)
251         comparison = df1.equals(df2)
252         return comparison
253     except Exception as e:
254         print(f"Error comparing Excel files: {e}")
255         return False
256 
257 def main():
```

```
258     conn = connect_to_db()
259     if not conn:
260         return False
261 
262     setup_database(conn)
263     user_id, expected_results = populate_database(conn)
264     if user_id is None:
265         return False
266 
267     export_db_to_excel(conn, 'old.xlsx')
268 
269     result = call_stored_procedure(conn, user_id)
270     if result is None:
271         print("Failed to call stored procedure.")
272         return False
273 
274     expected_serials = {str(res[0]): res[1:] for res in
    expected_results}
275     returned_serials = {str(serial[0]): serial[1:] for serial in
    result}
276 
277     print(expected_serials)
278     print(returned_serials)
279 
280     if expected_serials != returned_serials:
281         print(f"Test failed. Expected: {expected_serials}, Got: {
    returned_serials}")
282         return False
283 
284     export_db_to_excel(conn, 'new.xlsx')
285 
286     if compare_excel_files('old.xlsx', 'new.xlsx'):
287         print("The files old.xlsx and new.xlsx are identical.")
288         return True
289     else:
290         print("The files old.xlsx and new.xlsx are different.")
291         return False
292 
293 if __name__ == "__main__":
294     success = main()
295     print(f"Operation successful: {success}")
```

3.6 Примеры работы

На рисунках 3.2 – 3.5 представлены примеры работы программы.

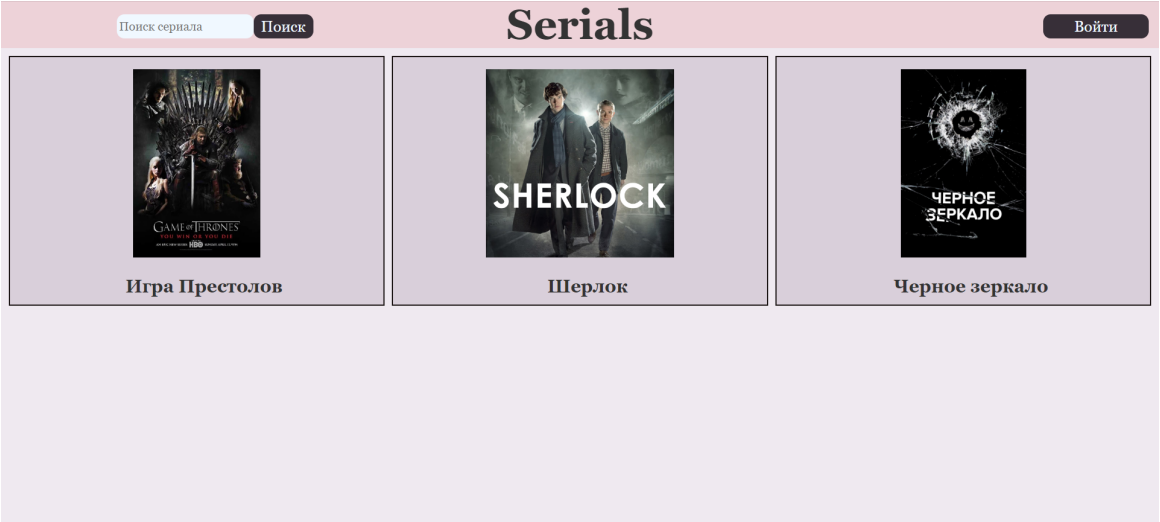

Рисунок 3.2 – Главная страница

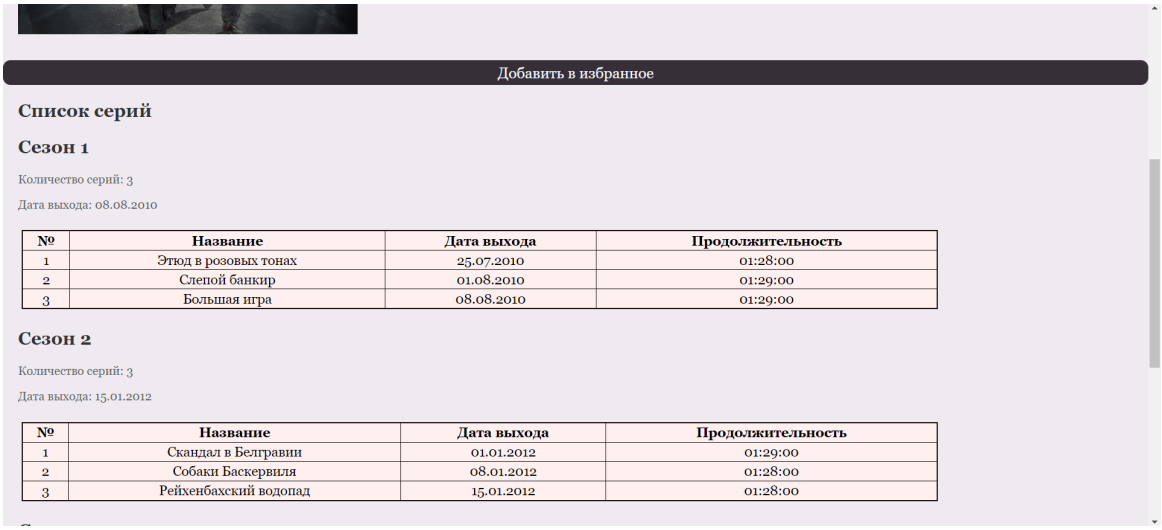

Рисунок 3.3 – Страница сериала: список серий

Рисунок 3.4 – Страница кабинета администратора

Рисунок 3.5 – Страница сравнения сериалов

Вывод

В данном разделе был обоснован выбор средств реализации, представлены листинги создания таблиц и ограничений целостности базы данных, создания ролей, реализации хранимой процедуры и способа ее тестирования, а также были приведены примеры работы программы.

4 Исследовательский раздел

4.1 Описание исследования

Целью является исследование влияния наличия индекса на время выполнения запроса к базе данных.

Для проведения исследования была выбрана таблица *Users*, для индексации — поле *u_name*. Время выполнения запроса измерялось с помощью втроенной в PostgreSQL команды EXPLAIN [24].

В листинге 4.1 представлен скрипт для анализа времени выполнения запроса.

Листинг 4.1 – Скрипт для анализа времени выполнения запроса

```
1 EXPLAIN ANALYSE
2 SELECT * FROM users
3 ORDER BY u_name;
```

В листинге 4.2 приведен скрипт для создания индекса.

Листинг 4.2 – Скрипт для создания индекса

```
1 CREATE INDEX users_name_index
2 ON users USING btree(u_name);
```

Технические характеристики устройства, на котором выполнялось исследование представлены далее:

— операционная система: Windows 11, x64;

— оперативная память: 8 Гб;

— процессор: AMD Ryzen 5 5500U с видеокартой Radeon Graphics 2.10 ГГц.

4.2 Результаты исследования

В таблице 4.1 представлены результаты замеров времени в миллисекундах. Каждый результат замеров является усредненным значением времени выполнения запроса к базе данных.

Таблица 4.1 – Результаты замеров времени

Количество записей	Без индексации, мс	С индексацией, мс
1	0.0265	0.0170
5001	17.1516	2.2535
10001	33.4603	4.2925
15001	48.8944	6.5388
20001	66.3264	8.9383
25001	79.9716	11.1433
30001	101.5194	15.2185
35001	143.2783	17.4035
40001	159.8446	18.7491
45001	171.9279	20.6471
50001	191.5169	25.4068
55001	211.8095	25.9988
60001	227.1767	28.1881
65001	247.2775	33.2508
70001	301.8896	34.5799
75001	285.8743	37.7099
80001	304.3699	39.7253
85001	327.2190	43.0365
90001	341.1442	47.4883
95001	369.6379	48.2742
100001	413.4842	53.0227

Визуализация результатов замеров представлена на рисунке 4.1.

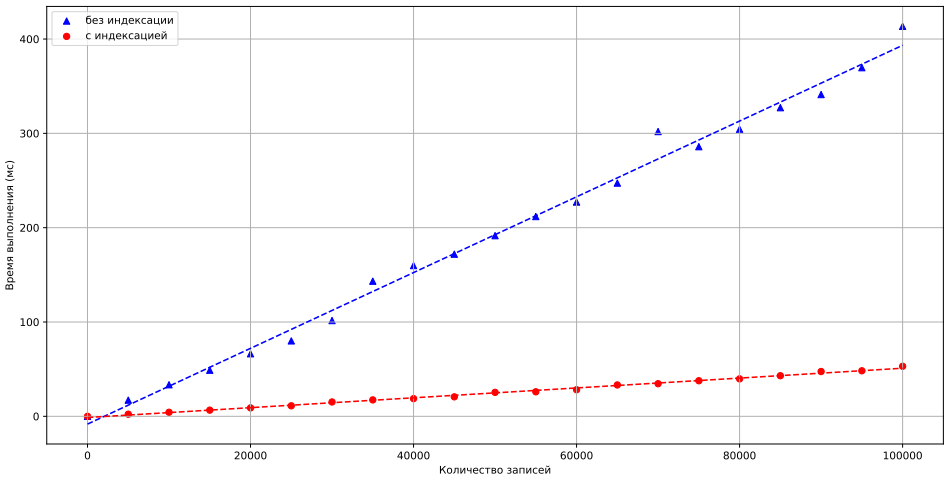

Рисунок 4.1 – Зависимость времени выполнения запроса к базе данных от количества записей

Была проведена интерполяция полиномом первой степени. Полученные интерполянты:

$$y_1 = 0.0040x - 8.25, y_2 = 0.0005x - 1.28$$

Вывод

В результате исследования было получено, что наличие индекса уменьшает время выполнения запроса к базе данных.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы цель была достигнута: разработана база данных для хранения и обработки данных сайта, посвященного сериалам. Все задачи были решены:

1. проведен сравнительный анализ известных решений;
2. формализована задача и информация, хранящаяся в базе данных;
3. спроектирована база данных;
4. реализовано программное обеспечение, предоставляющее интерфейс для доступа к базе данных;
5. исследовано влияние наличия индекса на время выполнения запроса в базе данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Левада-Центр : Досуг и развлечения [Электронный ресурс]. — 2019. — URL: https://www.levada.ru/2019/07/01/dosug-i-razvlecheniya/ (дата обращения: 8.3.2024).
2. Рейтинг сериалов: список сериалов на MyShows.me [Электронный ресурс]. — URL: https://myshows.me/ (дата обращения: 8.3.2024).
3. Шикимори - энциклопедия аниме и манги [Электронный ресурс]. — URL: https://shikimori.one/ (дата обращения: 8.3.2024).
4. TVGuru – новости сериалов, сенсации, трейлеры и обзоры новинок [Электронный ресурс]. — URL: https://tvguru.ru/ (дата обращения: 8.3.2024).
5. Кинориум [Электронный ресурс]. — URL: https://ru.kinorium.com/ (дата обращения: 8.3.2024).
6. *Карпова И. П.* Базы данных. Учебное пособие. — "Издательский дом Питер", 2021.
7. *Гущин А. Н.* Базы данных. — Directmedia, 2015.
8. Иерархическая модель данных [Электронный ресурс]. — URL: https://dit.isuct.ru/IVT/BOOKS/DBMS/DBMS14/ch_3_1.html (дата обращения: 12.3.2024).
9. Сетевая модель данных [Электронный ресурс]. — URL: https://dit.isuct.ru/IVT/BOOKS/DBMS/DBMS14/ch_3_2.html (дата обращения: 12.3.2024).
10. Организация данных в реляционной модели [Электронный ресурс]. — URL: https://dit.isuct.ru/IVT/BOOKS/DBMS/DBMS14/ch_4_1.html (дата обращения: 12.3.2024).
11. Ограничения целостности в реляционной модели данных [Электронный ресурс]. — URL: https://dit.isuct.ru/IVT/BOOKS/DBMS/DBMS14/ch_4_3.html (дата обращения: 12.3.2024).
12. Объектно-ориентированные СУБД [Электронный ресурс]. — URL: https://dit.isuct.ru/IVT/BOOKS/DBMS/DBMS14/ch_6_3.html (дата обращения: 12.3.2024).

13. Ranking of the most popular database management systems worldwide, as of February 2023 [Электронный ресурс]. — URL: https://www.statista.com/statistics/809750/worldwide-popularity-ranking-database-management-systems/ (дата обращения: 11.5.2024).

14. Oracle database [Электронный ресурс]. — URL: https://www.oracle.com/database/ (дата обращения: 11.5.2024).

15. MySQL [Электронный ресурс]. — URL: https://www.mysql.com/ (дата обращения: 11.5.2024).

16. SQL Server 2022 [Электронный ресурс]. — URL: https://www.microsoft.com/ru-ru/sql-server/sql-server-2022 (дата обращения: 11.5.2024).

17. PostgreSQL: Documentation [Электронный ресурс]. — URL: https://www.postgresql.org/docs/ (дата обращения: 11.5.2024).

18. Oracle, MySQL, PostgreSQL, SQLite, SQL Server: Performance based competitive analysis [Электронный ресурс]. — URL: http://dspace.daffodilvarsity.edu.bd:8080/handle/123456789/4437 (дата обращения: 11.5.2024).

19. The Go Programming Language [Электронный ресурс]. — URL: https://go.dev/ (дата обращения: 11.5.2024).

20. HTML Documentation [Электронный ресурс]. — URL: https://html-doc.vercel.app/ (дата обращения: 11.5.2024).

21. PostgreSQL: Documentation: 16: Chapter 43. PL/pgSQL — SQL Procedural Language [Электронный ресурс]. — URL: https://www.postgresql.org/docs/current/plpgsql.html (дата обращения: 11.5.2024).

22. Docker: Accelerated Container Application Development [Электронный ресурс]. — URL: https://www.docker.com/ (дата обращения: 11.5.2024).

23. Visual Studio Code - Code Editing. Redefined [Электронный ресурс]. — URL: https://code.visualstudio.com/ (дата обращения: 11.5.2024).

24. PostgreSQL: Documentation: 16: EXPLAIN [Электронный ресурс]. — URL: https://www.postgresql.org/docs/current/sql-explain.html (дата обращения: 11.5.2024).

ПРИЛОЖЕНИЕ А

Приложена презентация к курсовой работе.