

Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина «Модели данных и системы управления базами данных»

«К ЗАЩИТЕ ДОПУСТИТЬ»

Руководитель курсового проекта
ассистент кафедры информатики

_____.А.В.Давыдчик
_____._____.2024

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему
**ПРОГРАММНОЕ СРЕДСТВО, РЕАЛИЗУЮЩЕЕ БИБЛИОТЕКУ
ФИЛЬМОВ И СЕРИАЛОВ**

БГУИР КП 1-40 04 01

Выполнил студент группы 153501
Жур Вадим Дмитриевич

(подпись студента)

Курсовой проект представлен на
проверку _____._____.2024

(подпись студента)

Минск 2024

СОДЕРЖАНИЕ

1	Анализ предметной области	5
1.1	Обзор веб-сайта о кинематографе IMDb.....	5
1.2	Обзор социальной сети для любителей кино Letterboxd.....	10
1.3	Обзор стримингового сервиса фильмов и сериалов Netflix	15
2	Формирование функциональных требований.....	19
2.1	Функциональные требования к разрабатываемой базе данных.....	19
2.2	Анализ существующих подходов к разработке баз данных.....	20
2.3	Выбор базы данных для разработки предметной области.....	29
3	Проектирование базы данных.....	31
3.1	Описание сущностей.....	31
3.2	Разработка даталогической модели базы данных.....	33
4	Разработка базы данных	34
4.1	Создание исходных таблиц, индексов и ограничений	34
4.2	Создание хранимых процедур	34
4.3	Реализация триггеров.....	38
5	Разработка программного средства.....	40
5.1	Описание разработанного программного средства	40
	Заключение	43
	Список литературных источников	44
	Приложение А (обязательное) Листинг кода программы.....	46
	Приложение Б (обязательное) ER-диаграмма базы данных	55

ВВЕДЕНИЕ

Кинематограф давно стал неотъемлемой частью современной культуры и повседневной жизни. Он не только развлекает, но и формирует взгляды, расширяет кругозор и служит источником вдохновения. С ростом популярности фильмов и сериалов возросла и потребность в удобных системах, которые позволяют организовать, хранить и управлять коллекциями видеоконтента. Цифровые библиотеки фильмов и сериалов помогают пользователям систематизировать материалы, предоставлять легкий доступ к информации и поддерживать порядок в личных коллекциях.

С развитием интернет-технологий и цифровых устройств значительно вырос интерес к онлайн-платформам и приложениям для хранения видеоконтента. Такие системы предоставляют пользователям удобные функции для управления коллекциями, поиска и фильтрации контента по жанру, режиссеру, году выпуска и другим параметрам. Однако, несмотря на обилие существующих решений, многие из них не удовлетворяют всем потребностям пользователей, что создает необходимость в разработке улучшенных и более функциональных систем для работы с видеоархивами.

Целью данной курсовой работы является разработка концепции и прототипа программного обеспечения для создания цифровой библиотеки фильмов и сериалов. Программное обеспечение позволит пользователям управлять коллекциями видеоконтента, осуществлять удобный поиск и фильтрацию, а также поддерживать актуальность и структуру базы данных.

Для достижения поставленной цели необходимо решить следующие задачи:

Подведем основные цели на данный курсовой проект:

- провести анализ предметной области и изучить существующие аналоги;
- описать основные сущности проектируемой базы данных и связи между ними;
- разработать структуру базы данных, включая индексы для оптимизации поиска;
- создать триггеры и хранимые процедуры для обеспечения корректной работы системы;
- разработать программное обеспечение, использующее данные, которые хранятся в созданной базе данных.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Современные цифровые библиотеки фильмов и сериалов являются популярным инструментом для организации, хранения и доступа к видеоконтенту. С ростом интереса к фильмам и сериалам возникла потребность в более удобных и функциональных решениях для управления коллекциями. Среди множества приложений и сервисов можно выделить несколько популярных платформ, каждая из которых имеет уникальные особенности и функции. В рамках данного раздела будет проведен обзор основных конкурентов в области библиотек фильмов и сериалов:

- веб-сайт с крупнейшей в мире базой данных о кинематографе IMDb;
- социальная сеть для любителей кино Letterboxd;
- стриминговый сервис фильмов и сериалов Netflix.

1.1 Обзор веб-сайта о кинематографе IMDb

IMDb (Internet Movie Database) – одна из крупнейших и наиболее популярных платформ для поиска информации о фильмах, сериалах, актерах и событиях в киноиндустрии. Основанная в 1990 году, IMDb изначально представляла собой небольшую базу данных о фильмах, но со временем превратилась в полноценный интернет-портал для любителей кино и профессионалов. Сегодня IMDb входит в состав компании Amazon и охватывает десятки тысяч фильмов и сериалов, а также предлагает пользователям обширный функционал для взаимодействия с контентом. На рисунке 1.1 представлен графический интерфейс главной страницы веб-сайта:

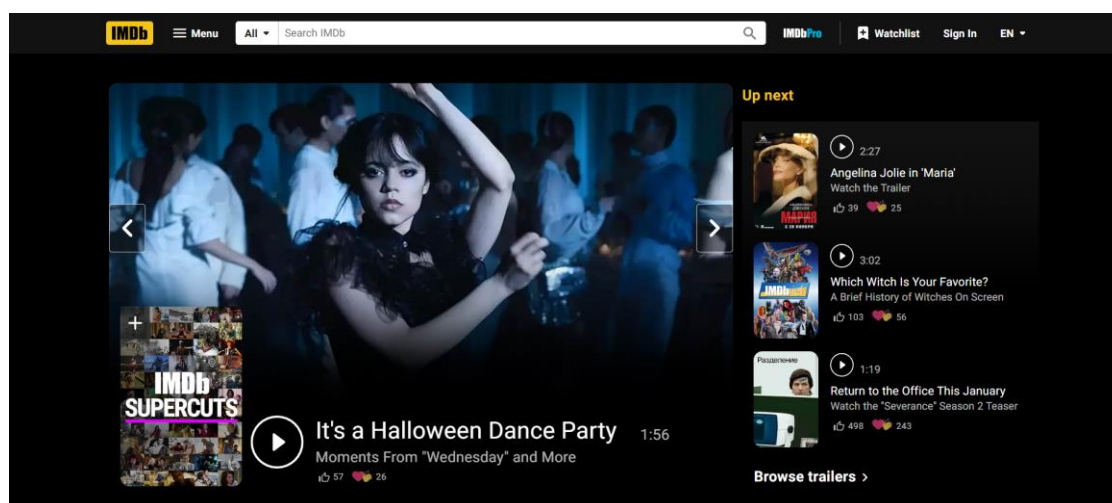


Рисунок 1.1 – Графический интерфейс главной страницы IMDb

IMDb предлагает пользователям мощный инструмент поиска и фильтрации, который позволяет находить фильмы и сериалы по множеству параметров, таких как жанр, год выпуска, страна производства, рейтинг и другие. Поиск можно осуществлять как по названию, так и по конкретным участникам съемочной группы, таким как актеры, режиссеры и сценаристы.

Пример фильтров и параметров для поиска представлен на рисунке 1.2:

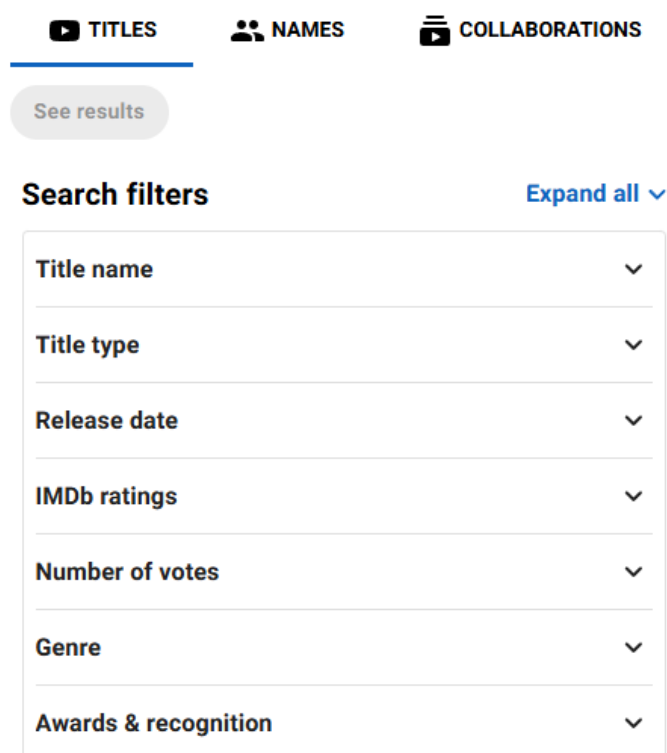


Рисунок 1.2 – Фрагмент списка фильтров в IMDb

Каждая единица контента на IMDb представлена в виде подробной карточки, где пользователи могут найти следующую информацию:

- краткое описание сюжета, постер и трейлер;
- основные актеры и съемочная группа, включая режиссера, сценариста и продюсера;
- жанр, продолжительность, возрастной рейтинг и страна производства;
- оценка пользователей IMDb и возможность просмотреть отзывы и рецензии.

Эта структура карточек позволяет быстро получить полное представление о любом фильме или сериале. Карточка также часто включает трейлеры, галерею изображений и ссылки на похожие фильмы.

На рисунке 1.3 продемонстрирован пример такой карточки для фильма *Catch Me If You Can*:

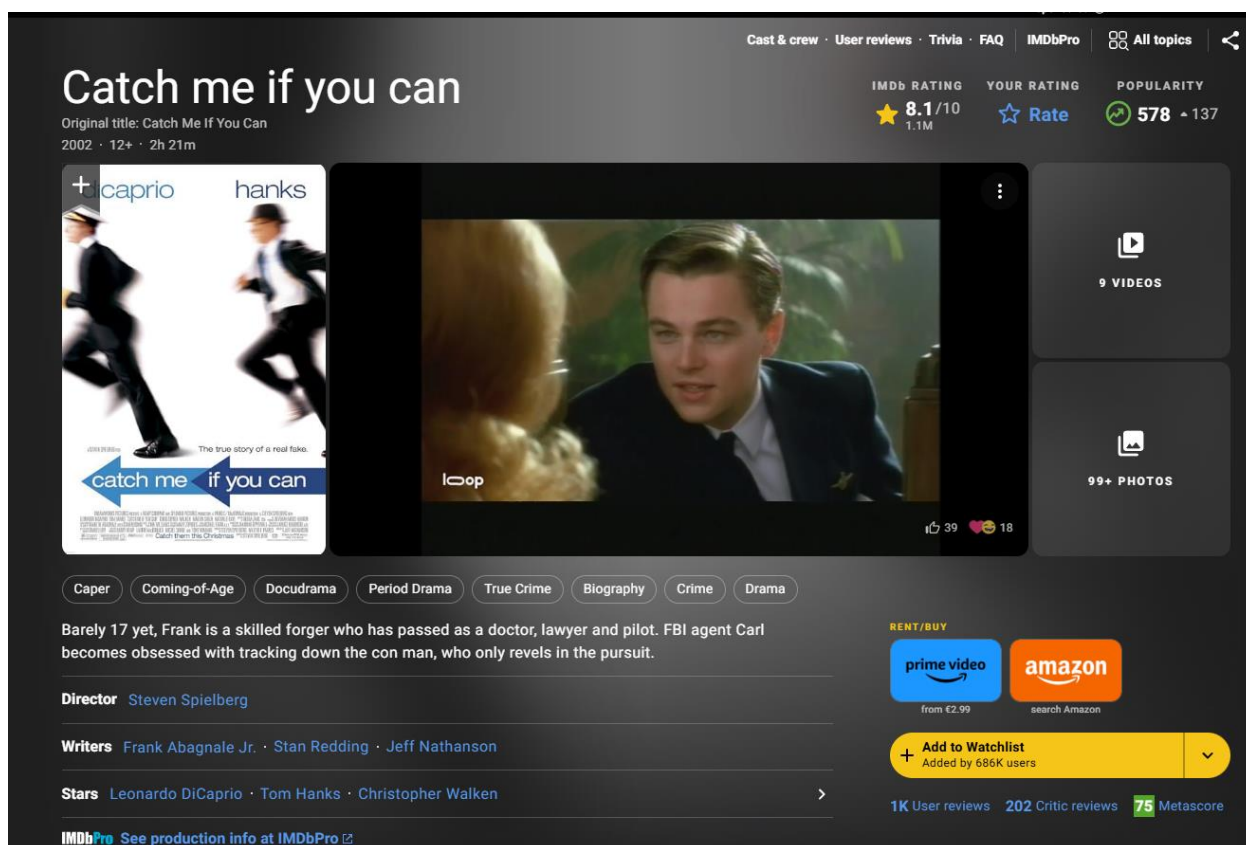


Рисунок 1.3 – Карточка с описанием выбранного фильма

IMDb предоставляет пользователям возможность оценивать фильмы и сериалы, а также оставлять отзывы. Рейтинг фильмов и сериалов на IMDb является одним из наиболее популярных и используется в качестве показателя качества и популярности контента. Пользователи могут голосовать, ставя оценку от 1 до 10, что позволяет создавать средний балл, который отображается на странице каждого фильма.

На рисунке 1.4 представлена страница с отзывами пользователей на выбранный фильм:

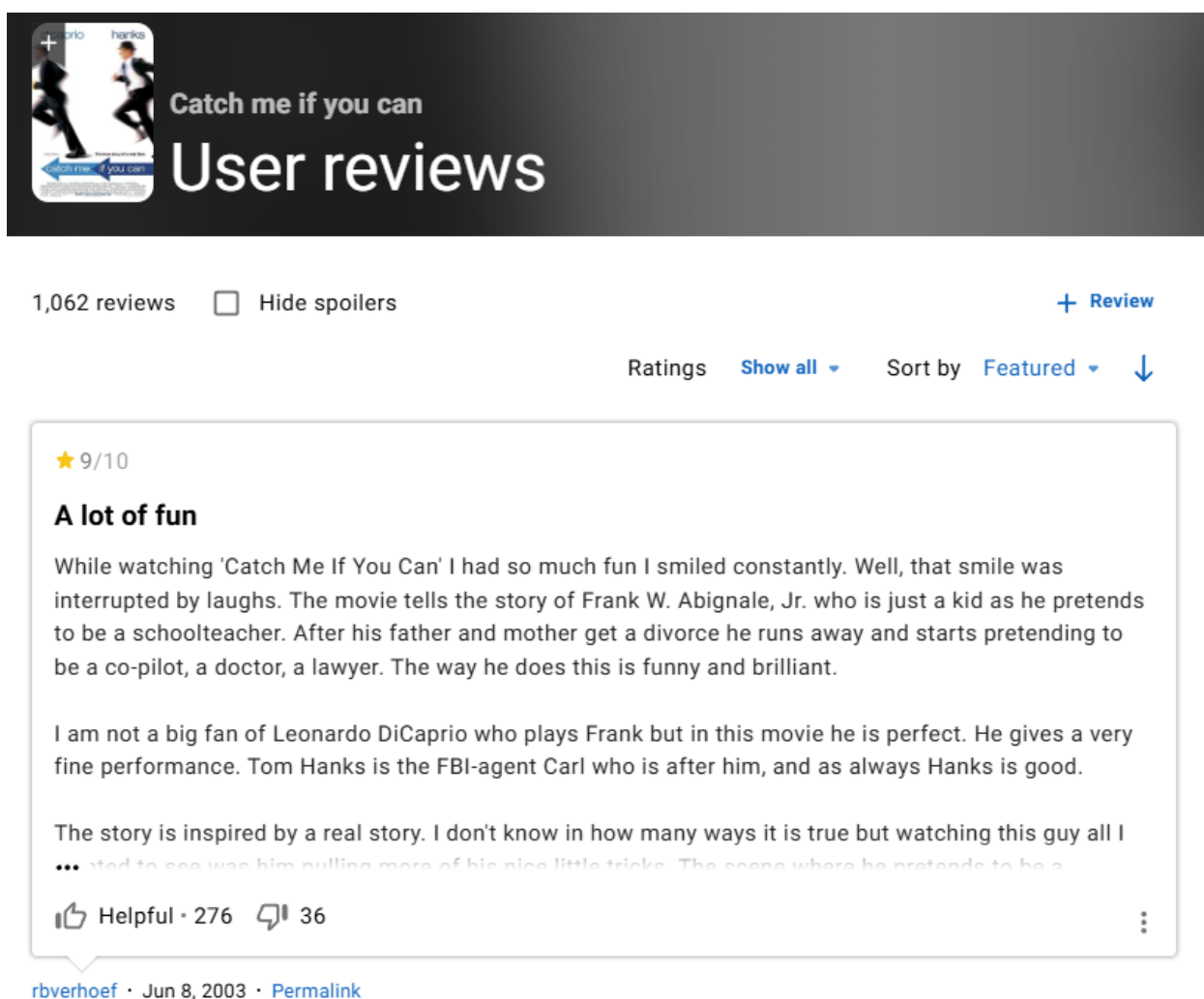


Рисунок 1.4 – Страница с отзывами на выбранный фильм

IMDb позволяет зарегистрированным пользователям создавать свои собственные списки фильмов и сериалов, добавляя в них контент по различным критериям. Это может быть полезно для создания персонализированных подборок, таких как «Лучшие фильмы по жанру», «Фильмы для просмотра позже» или «Избранные сериалы».

IMDb регулярно публикует новости о новинках киноиндустрии, а также анонсы грядущих фильмов и сериалов. Пользователи могут оставаться в курсе последних событий, таких как премьеры, награды и фестивали, что делает IMDb не просто базой данных, но и новостным ресурсом для киноманов.

Пример страницы новостей изображен на рисунке 1.5:

Movie News

Tom Holland's 'Spider-Man 4' Sets July 2026 Release, Following 'Avengers: Doomsday' [✉](#)



Tom Holland's Spider-Man will swing back into theaters on July 24, 2026.

The untitled Columbia Pictures project from director Destin Daniel Cretton will debut shortly after "Avengers: Doomsday," which comes out on May 1, 2026. "Spider-Man: Far From Home" was similarly released just two months after "Avengers: Endgame," and the third Spidey film grossed over \$1 billion globally.

During an appearance on "The Tonight Show Starring Jimmy Fallon" on Tuesday night, Holland confirmed that his fourth "Spider-Man" movie will begin production in mid-2025.

...

10/25/2024 · by Katcy Stephan · [Variety - Film News](#) [✉](#)



Рисунок 1.5 – Страница с новостями о фильмах

IMDb также предлагает платную услугу IMDb Pro, которая предоставляет дополнительные возможности для профессионалов в киноиндустрии. Эта подписка открывает доступ к расширенной информации о съемочной группе, продюсерах, а также позволяет создавать и управлять профессиональными профилями. IMDb Pro активно используется актерами, режиссерами и кастинг-менеджерами.

IMDb обладает рядом преимуществ, которые сделали его одной из ведущих платформ в своей области:

1 Широкий охват информации. IMDb охватывает практически все аспекты кино и телевидения, включая фильмы, сериалы, анимацию и документальные проекты.

2 Пользовательские рейтинги и отзывы. Рейтинг IMDb является одним из самых цитируемых показателей качества фильмов, благодаря огромной пользовательской базе.

3 Обширная база данных. IMDb включает множество фильмов и сериалов, что позволяет пользователям найти почти любой контент.

Однако IMDb также имеет некоторые ограничения:

1 Отсутствие функций для просмотра контента. IMDb предоставляет информацию, но не является стриминговым сервисом.

2 Ограниченная функциональность в создании коллекций. Платформа не рассчитана на ведение личных коллекций с углубленным управлением контентом.

IMDb является одной из самых известных и востребованных платформ для поиска информации о фильмах и сериалах. Благодаря обширной базе данных, рейтинговой системе и возможности создавать списки, IMDb привлекает как любителей кино, так и профессионалов в этой индустрии. Однако для создания полнофункциональной библиотеки видеоконтента может потребоваться более персонализированное решение, которое позволит пользователям эффективно управлять собственными коллекциями. [1]

1.2 Обзор социальной сети для любителей кино Letterboxd

Letterboxd – это социальная платформа для киноманов, позволяющая пользователям вести учет просмотренных фильмов, создавать списки, оставлять рецензии и делиться своими предпочтениями с другими. Основанная в 2011 году, Letterboxd предлагает удобный и интуитивный интерфейс для отслеживания истории просмотров и формирования коллекций, а также обладает социальной функцией, что делает его популярным среди поклонников кино. Главная страница Letterboxd представлена следующим образом на рисунке 1.6:

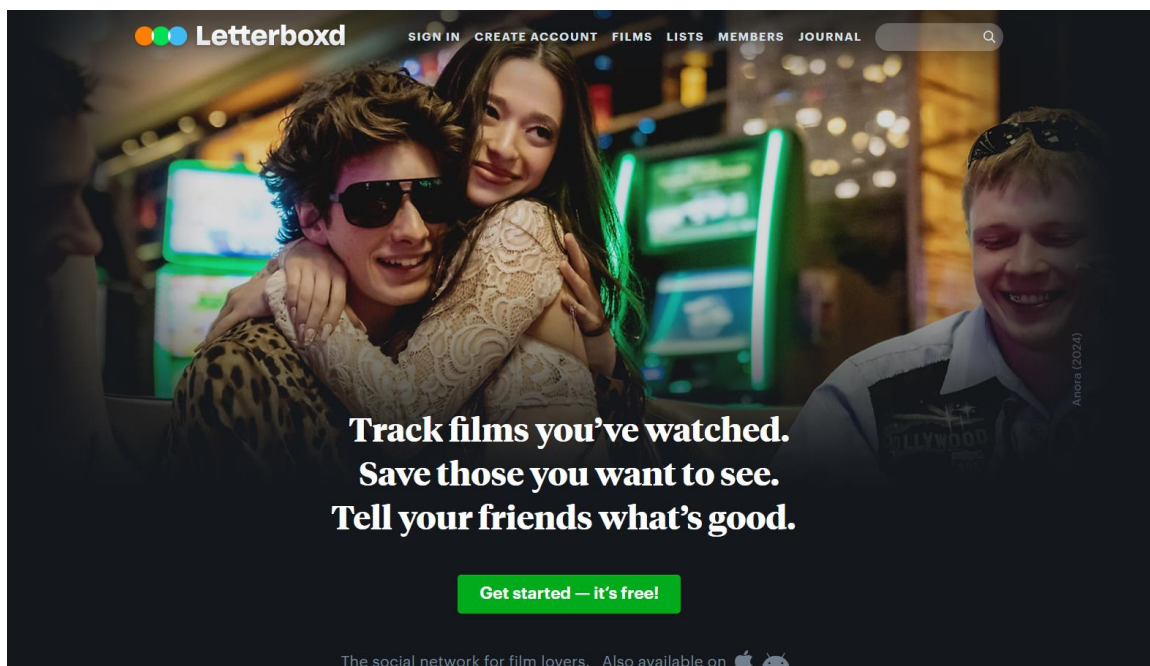


Рисунок 1.6 – Главная страница Letterboxd

Letterboxd сочетает в себе функциональность дневника для фильмов и элементы социальной сети, что позволяет пользователям не только отслеживать свои просмотры, но и взаимодействовать с сообществом. Ниже описаны ключевые возможности платформы.

Основная функция Letterboxd – ведение личного дневника, где пользователи могут отмечать просмотренные фильмы. Для этого достаточно найти фильм через поиск и добавить его в дневник, выбрав дату просмотра. На странице каждого фильма можно оставить отзыв, указать рейтинг и сохранить его в списках или коллекциях.

Пример страницы, демонстрирующей данную возможность, а также подробную информацию о фильме показан на рисунке 1.7.

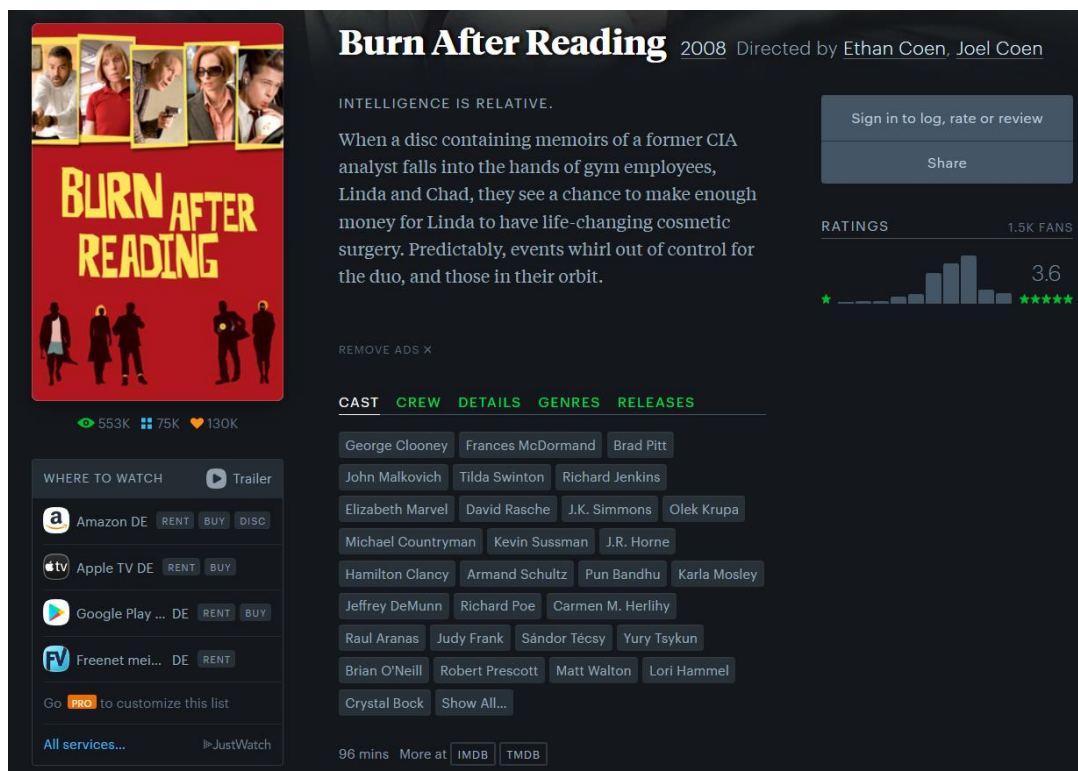


Рисунок 1.7 – Страница с информацией о фильмах

Пользователи могут создавать собственные списки фильмов по любым категориям – от лучших фильмов года до тематических подборок. Списки могут быть публичными или приватными, и они отображаются на странице пользователя, позволяя другим видеть его предпочтения и интересы. Этот инструмент особенно полезен для составления рекомендаций или для планирования будущих просмотров. Пример тематических списков фильмов представлен на рисунке 1.8:

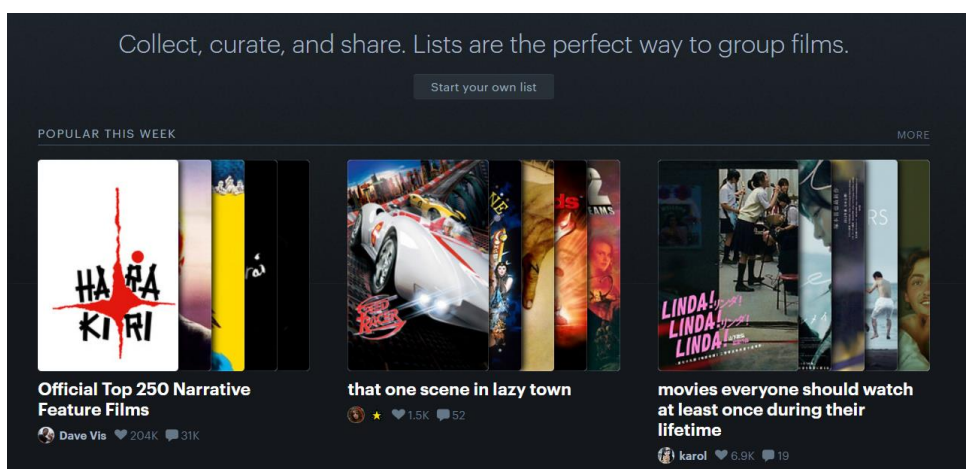


Рисунок 1.8 – Тематические списки фильмов

Letterboxd выделяется среди других платформ за счет своих социальных возможностей. Пользователи могут подписываться на других, оставлять комментарии под рецензиями, лайкать чужие отзывы и делиться своими впечатлениями. Каждый пользователь имеет профиль, где отображаются его списки, рейтинги и история просмотров.

На Letterboxd у пользователей есть возможность оставлять подробные рецензии на фильмы и оценивать их по 5-звездочной шкале. Рецензии можно читать на странице фильма, что позволяет пользователям ознакомиться с мнениями других людей перед просмотром. Платформа также показывает популярные и наиболее высоко оцененные рецензии для каждого фильма, что делает ее ресурсом для рекомендаций и анализа.

Данные возможности сервиса проиллюстрированы на рисунке 1.9:

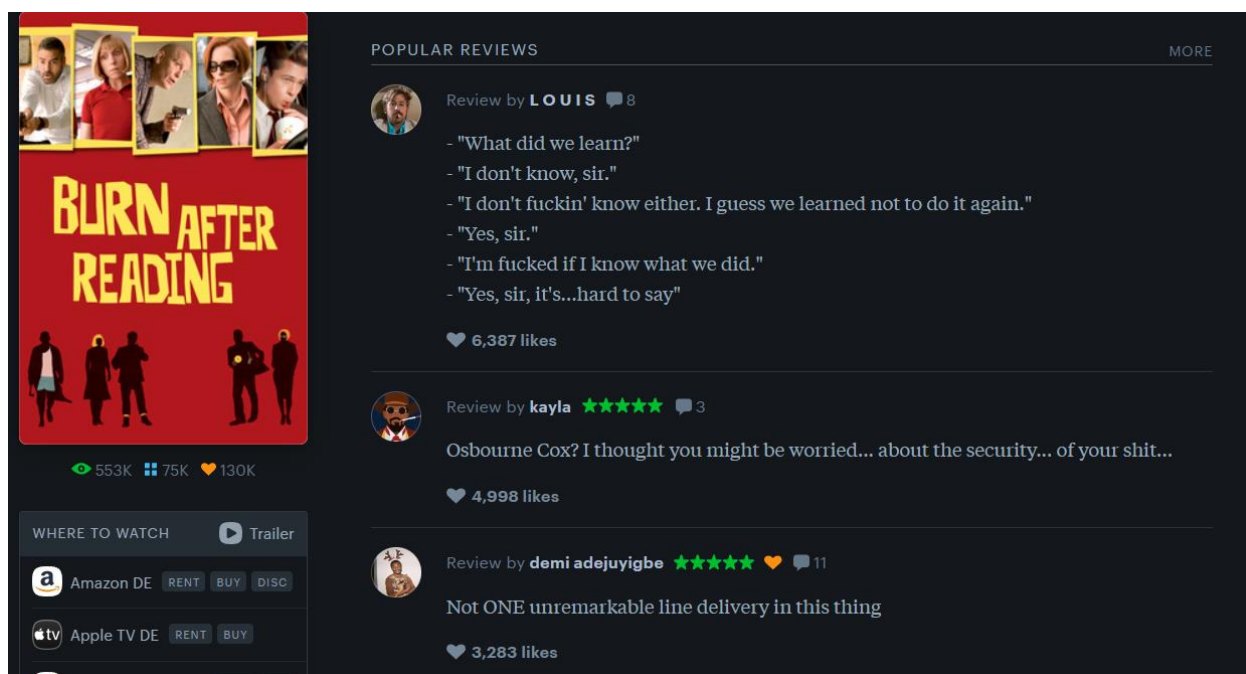


Рисунок 1.9 – Отзывы к фильмам от пользователей

Также Letterboxd имеет удобный инструмент для добавления фильмов в раздел «Посмотреть позже», позволяя пользователям сохранять интересные их фильмы для будущих просмотров. Это особенно полезно для отслеживания фильмов, которые только что вышли в прокат или недавно были рекомендованы другими пользователями.

Letterboxd предлагает пользователям возможность создавать и просматривать статистику их кинопросмотров, включая годовые итоги. Эти отчеты включают общее количество просмотренных фильмов, предпочтения

по жанрам, наиболее популярные режиссеры и другие интересные данные. Персональные сводки помогают пользователям лучше понять свои вкусы и привычки, что делает использование платформы более персонализированным.

Помимо бесплатного доступа, Letterboxd предлагает два уровня подписки — Pro и Patron. Подписка Pro предоставляет дополнительные функции, такие как расширенная статистика, отсутствие рекламы и возможность интеграции с другими сервисами для импорта фильмов. Подписка Patron включает все функции Pro и добавляет возможность поддержки проекта, что отмечается на странице пользователя особой меткой.

Пример персональной страницы пользователя данной сети представлен на рисунке 1.10:

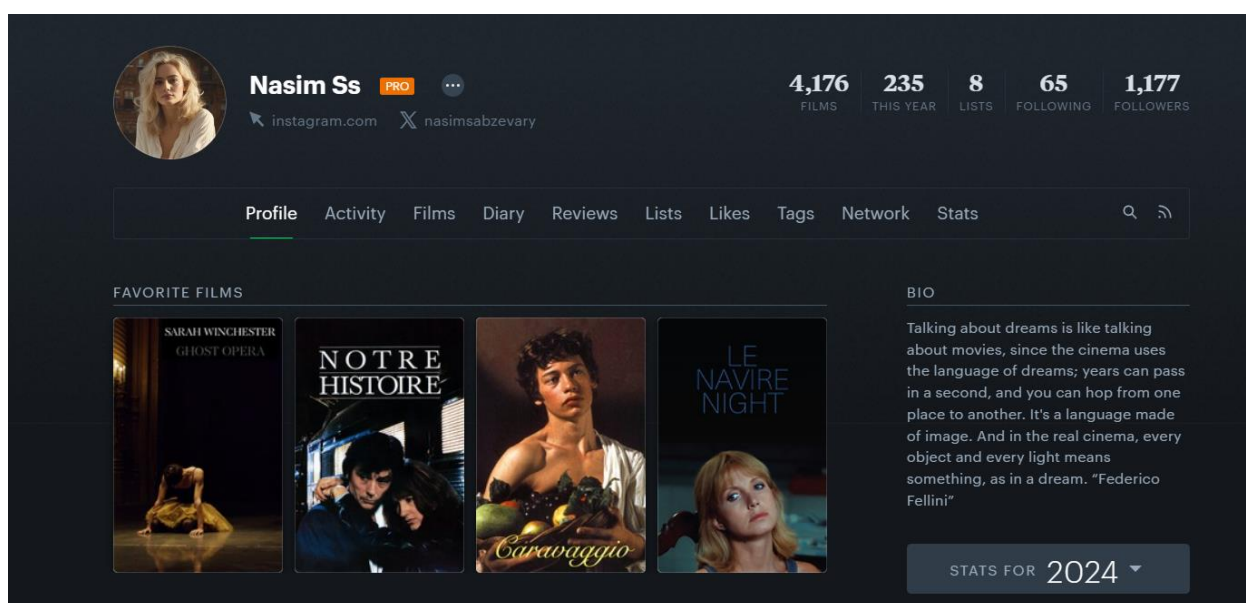


Рисунок 1.10 – Персональная страница пользователя сети

К преимуществам Letterboxd относятся:

1 Социальные функции. Возможность подписываться на других пользователей, оставлять комментарии и делиться списками делает Letterboxd привлекательным для активного взаимодействия с другими киноманами.

2 Простой интерфейс для учета просмотров. Удобное добавление фильмов в дневник и создание списков упрощает отслеживание истории просмотров.

3 Персонализированные сводки. Статистика просмотров и годовые отчеты помогают пользователям лучше понять свои предпочтения в фильмах.

К недостаткам данной социальной сети можно отнести:

1 Ограниченная база данных сериалов. Letterboxd ориентирован

преимущественно на фильмы и содержит ограниченную информацию о сериалах, что может ограничивать пользователей, интересующихся этим видом контента.

2 Платные функции. Для доступа к расширенной статистике и другим функциям требуется подписка Pro или Patron, что не всегда удобно для обычных пользователей.

Letterboxd – это уникальная платформа, объединяющая киноманов по всему миру и предоставляющая удобный инструмент для ведения дневника просмотренных фильмов. Сочетание социальных функций с возможностью создания списков и просмотра персонализированных отчетов делает Letterboxd идеальным выбором для пользователей, желающих не просто организовать коллекцию фильмов, но и активно взаимодействовать с сообществом. [2]

1.3 Обзор стримингового сервиса фильмов и сериалов Netflix

Netflix – одна из самых популярных и влиятельных платформ для потокового воспроизведения фильмов и сериалов. Основанный в 1997 году как сервис для аренды DVD, Netflix позже перешел к онлайн-стримингу и создал собственное производство контента, известное как Netflix Originals. Сегодня Netflix предоставляет миллионам пользователей по всему миру доступ к обширной библиотеке фильмов, сериалов и документальных фильмов, а также предлагает интуитивный интерфейс и мощные рекомендации для улучшения пользовательского опыта.

Главное окно при входе на платформу представлено на рисунке 1.11:

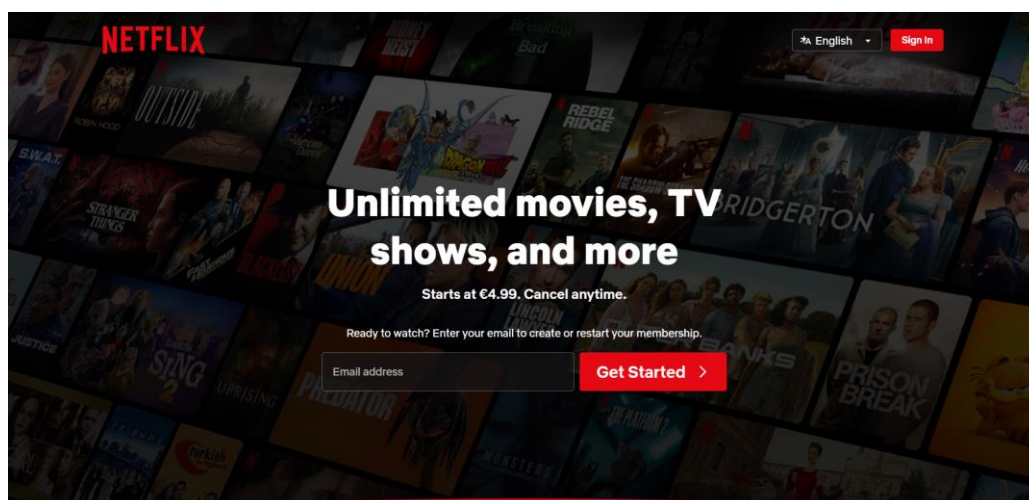


Рисунок 1.11 – Главная страница Netflix

Netflix объединяет в себе простоту использования, гибкость настроек и персонализированные рекомендации. Ниже описаны основные возможности, которые делают Netflix столь востребованным среди зрителей.

Netflix предлагает обширный каталог, включающий фильмы и сериалы различных жанров, стран и времен. Каталог Netflix разделен на категории, такие как «Тренды», «Новинки», «Популярные», а также персонализированные подборки на основе предпочтений пользователя. Netflix активно лицензирует контент и добавляет собственные оригинальные фильмы и сериалы, что делает его привлекательным для зрителей по всему миру.

Пример страницы каталога продемонстрирован на рисунке 1.12:

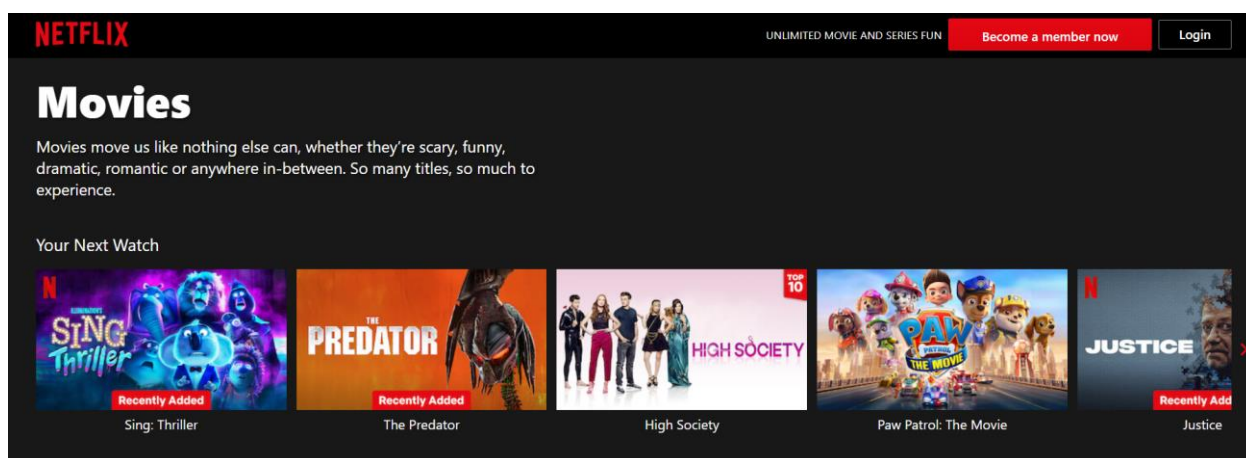


Рисунок 1.12 – Страница каталога фильмов

Одним из ключевых преимуществ Netflix является его мощная система рекомендаций, которая на основе предпочтений, истории просмотров и оценок пользователя предлагает релевантный контент. Рекомендации включают списки «Похожие на...», «Популярные сейчас» и другие подборки, что помогает пользователям находить интересные фильмы и сериалы быстро и эффективно.

Netflix позволяет создавать до пяти отдельных профилей в рамках одной учетной записи, что удобно для семей или групп пользователей. Каждый профиль сохраняет уникальные настройки и рекомендации, а также персональную историю просмотров, позволяя каждому пользователю получать персонализированный опыт.

Пример страницы с описанием фильма представлен на рисунке 1.13:



Рисунок 1.13 – Страница с описанием фильма

Пользователи могут скачивать фильмы и сериалы для просмотра в автономном режиме, что особенно полезно при ограниченном доступе к интернету. Эта функция доступна для большинства контента и позволяет выбрать качество видео, что помогает экономить место на устройстве.

Netflix предоставляет родительские настройки, позволяя ограничивать доступ к контенту по возрастным категориям, что особенно важно для семей с детьми. Родители могут настроить детские профили с контентом, подходящим для младшей аудитории, и установить PIN-код для ограничения доступа к взрослому контенту.

В каталоге Netflix также представлены интерактивные фильмы и сериалы, где зрители могут принимать решения за персонажей, влияя на ход сюжета. Такие проекты, как «Черное зеркало: Брандашмыг», стали популярными и добавили интерактивный элемент в обычный процесс просмотра.

Netflix активно инвестирует в производство собственного контента, известного как Netflix Originals. Это включает в себя фильмы, сериалы, документальные и анимационные проекты, которые доступны исключительно на платформе. Netflix Originals отличаются высоким качеством и зачастую собирают миллионы зрителей, а также получают признание критиков и награды на международных фестивалях.

Среди преимуществ Netflix стоит отметить:

- 1 Широкий каталог и уникальный контент. Наличие разнообразного контента, включая Netflix Originals, делает платформу привлекательной для всех категорий зрителей.

2 Персонализированные рекомендации. Благодаря алгоритмам Netflix пользователи могут находить контент, который соответствует их вкусам, что экономит время на поиски.

3 Мультипрофильность и офлайн-доступ. Поддержка нескольких профилей и возможность загружать контент делают Netflix удобным для использования семьей или друзьями.

Однако присутствуют и недостатки у данного стримингового сервиса:

1 Региональные ограничения. Библиотека Netflix варьируется в зависимости от региона, и некоторые фильмы или сериалы могут быть недоступны в определенных странах.

2 Изменяющийся каталог. Лицензионные соглашения означают, что контент может временно исчезнуть из библиотеки Netflix, что неудобно для пользователей, которые хотели бы пересмотреть любимые фильмы или сериалы.

3 Зависимость от интернет-соединения. Качество потокового воспроизведения зависит от скорости интернета, и пользователи с медленным интернетом могут испытывать задержки и ухудшение качества видео.

Netflix является лидером среди стриминговых сервисов благодаря широкому каталогу контента, мощной системе рекомендаций и качественным оригинальным проектам. Эти особенности позволяют пользователям быстро находить и просматривать интересующие их фильмы и сериалы. Однако для создания полноценной библиотеки видеоконтента с сохранением коллекций Netflix может не подходить, поскольку акцентирован на потоковом воспроизведении, а не на хранении или управлении коллекцией. [3]

2 ФОРМИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

2.1 Функциональные требования к разрабатываемой базе данных

На основе анализа существующих аналогов выдвинем функциональные требования к разрабатываемой библиотеке фильмов и сериалов, в которой предусмотрены функции авторизации пользователя, просмотра истории действий и информации о фильмах:

1 Авторизация пользователя. Пользователи должны иметь возможность авторизоваться в системе, используя свои учетные данные, и получать доступ к расширенным функциям библиотеки в зависимости от своей роли.

2 Управление пользователями (CRUD). Система должна позволять создание, чтение, обновление и удаление учетных записей пользователей, обеспечивая гибкость и контроль для администраторов.

3 Ролевая система. Необходимо реализовать систему ролей, где предусмотрены роли администратора и обычного пользователя. Разграничение прав доступа позволит обеспечить безопасность и функциональность системы.

4 История действий пользователей. Система должна вести журнал всех действий пользователей для отслеживания истории взаимодействий. Пользователи смогут просматривать свои действия, а администраторы — действия всех пользователей.

После авторизации обычные пользователи должны иметь следующие возможности:

1 Поиск фильмов. Возможность находить фильмы по ключевым словам или другим фильтрам.

2 Получение информации о фильме. Доступ к детальной информации, включая описание, жанр, актеров и другие характеристики фильма.

3 Оценка фильма. Возможность поставить фильму оценку, которая будет использоваться для расчета средней оценки.

4 Просмотр истории действий. Доступ к журналу собственных действий в системе, позволяющий отслеживать взаимодействие с контентом.

Пользователи с ролью администратора будут иметь доступ к расширенным функциям:

1 Доступ к функциям пользователей. Возможность выполнять все те же действия, что и обычные пользователи (поиск фильмов, оценка и т.д.).

2 Получение информации о пользователях. Доступ к информации о

зарегистрированных пользователей для управления и мониторинга.

3 Управление пользователями (CRUD). Администраторы могут добавлять, редактировать и удалять учетные записи пользователей, а также назначать или изменять роли.

Эти требования обеспечат надежное управление данными, удобство использования для пользователей и функциональные возможности для администраторов при поддержке контроля доступа и безопасности системы.

2.2 Анализ существующих подходов к разработке баз данных

В современной разработке баз данных выделяют два основных типа: SQL и NoSQL. Их отличие связано не только с использованием SQL-запросов, но и с принципами организации данных. SQL базы данных, основанные на реляционной модели, представляют данные в виде таблиц с четко определенными связями, что обеспечивает целостность данных и удобную обработку с помощью SQL. NoSQL базы данных, напротив, не придерживаются реляционной теории и могут хранить данные в различных форматах (документы, графы, колонки и др.), что позволяет гибко адаптироваться к задачам масштабируемых систем. [4]

Для лучшего понимания структуры и подвидов этих баз данных обратимся к рисунку 2.1, который иллюстрирует основные категории SQL и NoSQL баз.

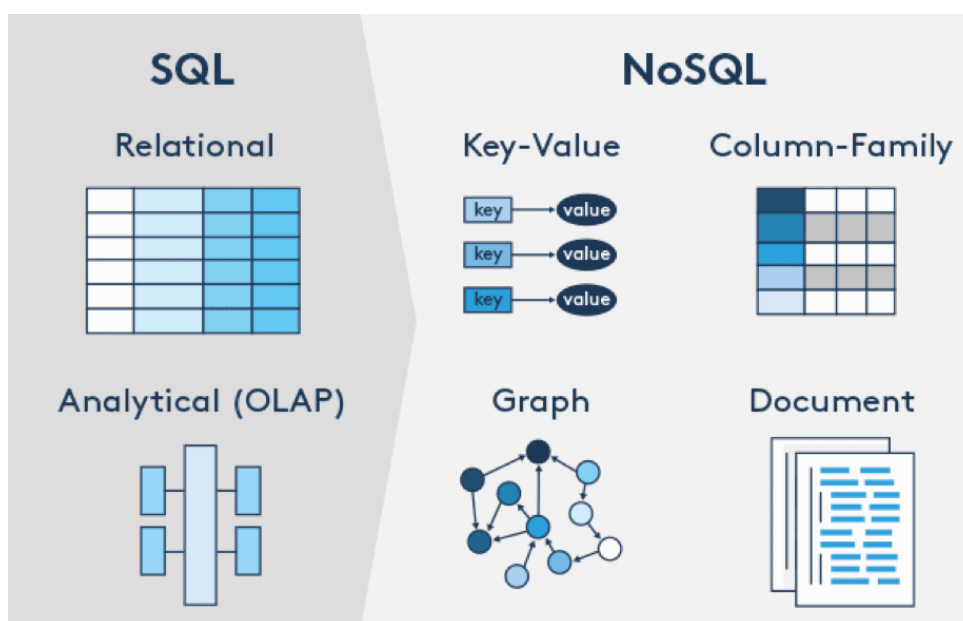


Рисунок 2.1 – Основные виды SQL и NoSQL баз данных

Далее рассмотрим более подробно основные отличия реляционных и нереляционных баз данных.

2.2.1 Реляционные базы данных

Реляционные базы данных – это базы данных, которые используют табличную структуру для хранения и обработки данных. В них информация организована в виде таблиц, состоящих из строк и столбцов, где каждая строка представляет собой запись с уникальным ключом. Столбцы содержат атрибуты, описывающие характеристики записей. Связи между таблицами устанавливаются с помощью ключей, что поддерживает целостность и согласованность данных [5]. Пример реляционной базы данных показан на рисунке 2.2.

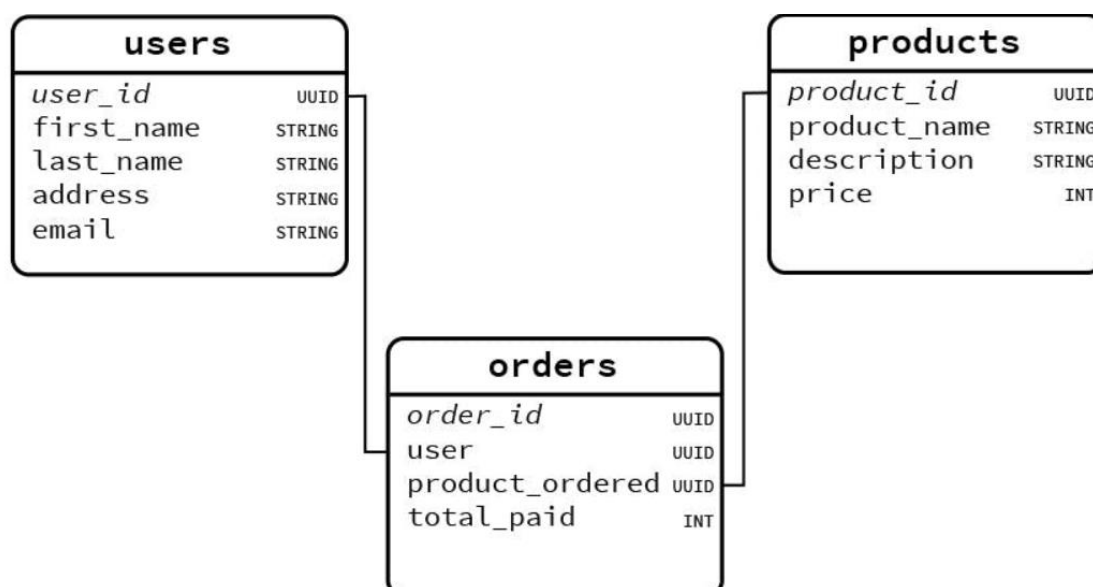


Рисунок 2.2 – Пример организации данных в реляционной теории

Для управления реляционными базами данных применяется стандартный язык SQL, позволяющий выполнять различные операции с данными: выборку, добавление, обновление, удаление, объединение, агрегацию и другие. SQL также поддерживает транзакции – неделимые блоки операций, которые подчиняются принципам ACID:

1 Атомарность (Atomicity). Транзакция выполняется как единое целое, «всё или ничего». Это значит, что либо все операции в транзакции выполняются успешно, либо ни одна из них не выполняется. Если какая-то операция в транзакции не удастся, происходит откат, и база данных

возвращается к состоянию до начала транзакции. Например, при переводе денег между счетами атомарность гарантирует, что деньги спишутся с одного счета только в том случае, если они зачислятся на другой.

2 Согласованность (Consistency). После завершения транзакции база данных должна перейти из одного согласованного состояния в другое. Согласованность подразумевает соблюдение всех установленных правил и ограничений. Например, если для определённого столбца указано ограничение на уникальные значения, транзакция, нарушающая это правило, не будет завершена.

3 Изолированность (Isolation). Параллельные транзакции не должны влиять друг на друга, обеспечивая независимость их выполнения. Изолированность гарантирует, что одна транзакция не сможет «увидеть» изменения, вносимые другой, пока та не завершится. Существуют разные уровни изолированности, такие как «чтение с повторной фиксацией» и «серийная изолированность», которые регулируют степень изолированности транзакций.

4 Долговечность (Durability). После успешного завершения транзакции ее результаты сохраняются в базе данных даже в случае сбоя системы. Это означает, что изменения в данных, внесенные транзакцией, сохраняются в долгосрочной перспективе и остаются доступными после завершения транзакции.

Примерная структура команд и их классификация по группам представлена на рисунке 2.3.

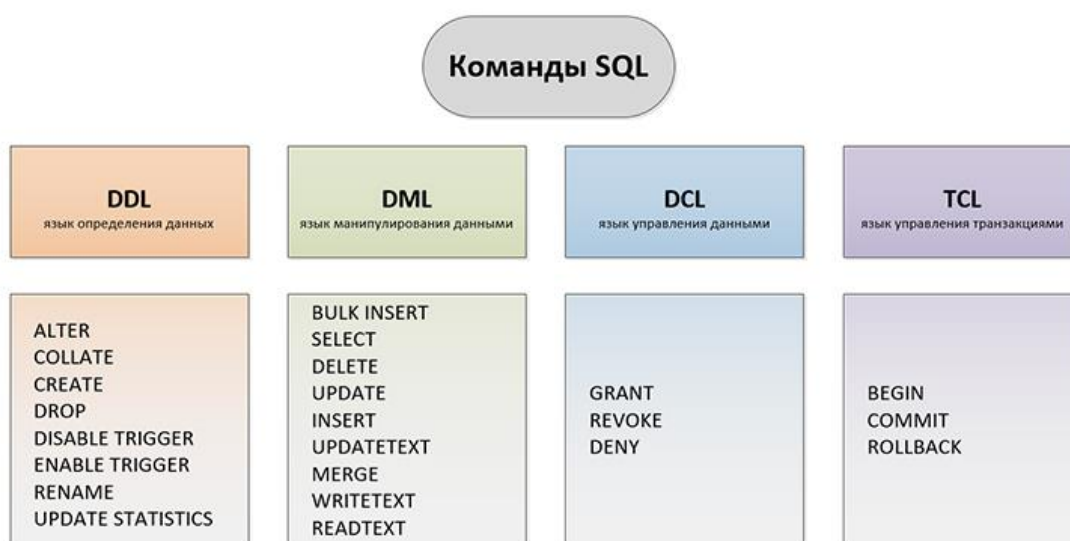


Рисунок 2.3 – Структура команд SQL

Логическое и физическое представления баз данных – это два подхода к описанию и организации данных. Логическое представление показывает, как данные воспринимаются и используются пользователями и приложениями, в то время как физическое представление описывает способ хранения и обработки данных системой управления базой данных. [6, 7]

Логическое представление включает логические объекты, такие как таблицы, представления, индексы и т.д., которые определяют структуру данных, их свойства и взаимосвязи, а также правила и ограничения, обеспечивающие целостность данных. Оно описывается с помощью концептуальной схемы, которая отображает все элементы данных и их связи в виде графической диаграммы. Логическое представление можно разделить на несколько уровней абстракции: внешний, логический и внутренний:

1 Внешний уровень: отображает данные в удобном для восприятия формате для конкретного пользователя или приложения.

2 Логический уровень: представляет данные в общем формате, доступном для всех пользователей и приложений.

3 Внутренний уровень: описывает данные в формате, понятном системе управления базой данных.

Пример логического представления базы данных представлен на рисунке 2.4:

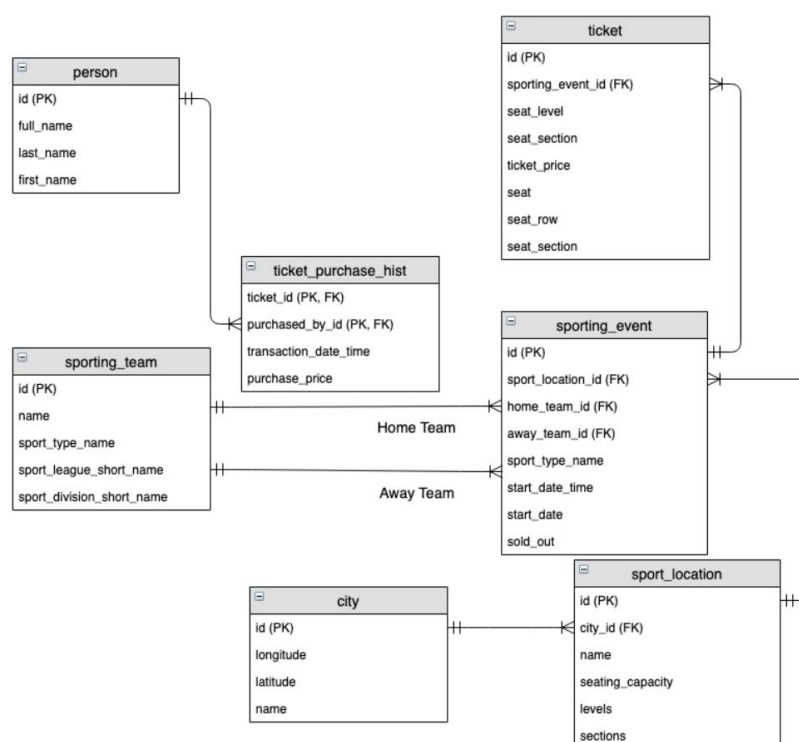


Рисунок 2.4 – Логическое представление базы данных

Физическое представление базы данных включает физические объекты, такие как файлы, блоки, страницы, сегменты и другие элементы. Эти объекты определяют, как данные хранятся, обрабатываются и доступны на физическом носителе, будь то диск, память или сеть. [8, 9]

Физическое представление описывается с помощью физической схемы, в которой указано расположение и размер физических объектов, а также настройки и параметры, влияющие на производительность и эффективность базы данных. Для оптимизации физического представления могут применяться различные методы и техники, такие как сжатие данных, шардирование, кэширование, индексирование и другие. Схема организации базы данных на физическом уровне представлена на рисунке 2.5.

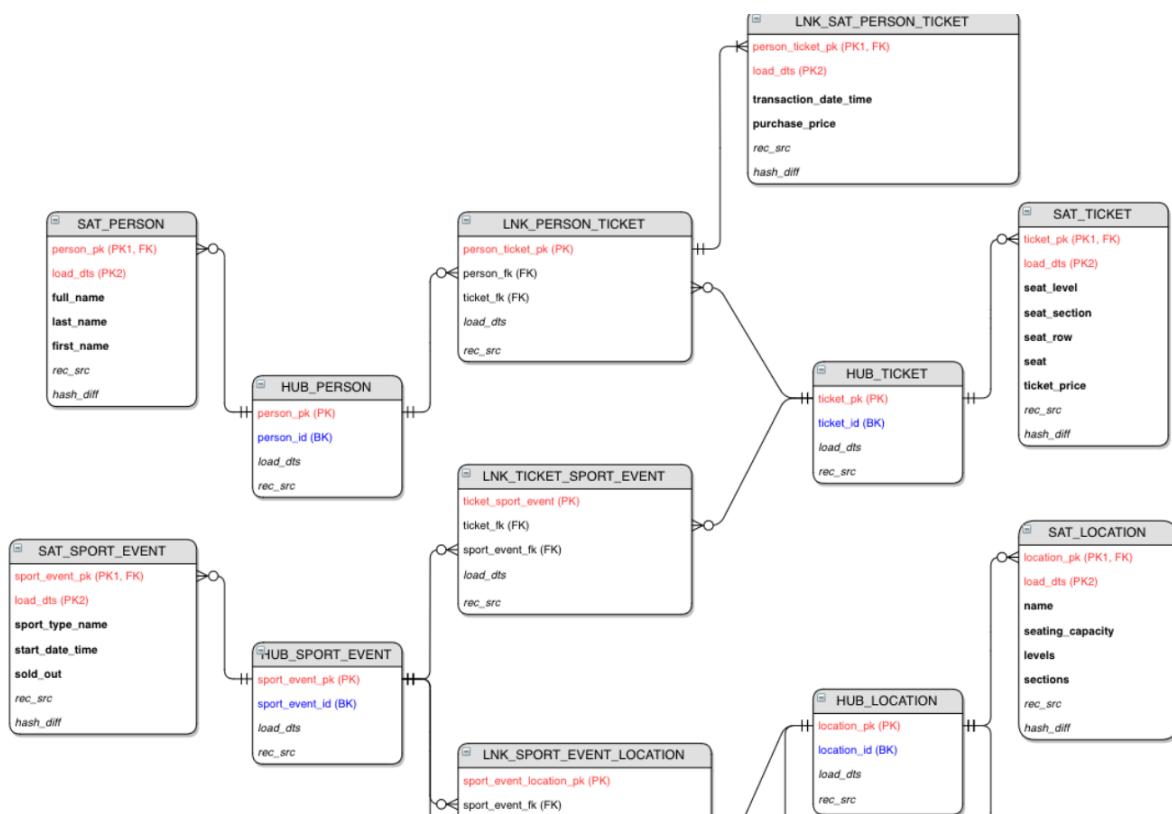


Рисунок 2.5 – Схема организации базы данных с физическим уровнем

Разделение логического и физического представлений базы данных обеспечивает логическую и физическую независимость данных. Логическая независимость подразумевает, что изменения в структуре данных на логическом уровне не затрагивают прикладные программы и пользователей. Физическая независимость означает, что изменения в физическом хранении данных не оказывают влияния на их логическую структуру и прикладные программы. [10, 11]

2.2.2 Нормализация реляционных баз данных

Избыточность данных в отношениях – одна из основных сложностей, с которой сталкиваются разработчики при проектировании баз данных. Она возникает, когда одни и те же данные повторяются в разных отношениях или когда поля не соответствуют сущностям, которые они описывают (что часто бывает взаимосвязано).

Процесс нормализации помогает устранить избыточность данных, основываясь на правилах проектирования баз данных, таких как правило, что «каждый факт должен быть представлен только один раз». Нормализация преобразует базу данных к определённой нормальной форме, улучшая её структуру пошагово. Этот процесс обратим, то есть можно вернуться к предыдущим формам, например, из третьей нормальной формы (3НФ) в исходную вторую (2НФ), что является важным преимуществом, так как это сохраняет исходные данные без потерь. В настоящее время существует шесть нумерованных нормальных форм, включая нормальную форму Бойса-Кодда и доменно-ключевую нормальную форму. На практике, для рабочих проектов чаще всего достаточно нормализовать данные до третьей нормальной формы (3НФ), тогда как остальные формы применяются в научных исследованиях.

Первая нормальная форма – это базовое требование для всех реляционных баз данных. Сущность находится в 1НФ, если каждый кортеж в допустимом значении переменной содержит только одно значение для каждого атрибута. Это требование подразумевает, что отношения по умолчанию должны находиться в первой нормальной форме. Однако, отношение, находящееся только в 1НФ, обычно плохо подходит для внедрения в информационные системы, так как оно не устраняет избыточность данных, что приводит к возникновению так называемых аномалий обновления. [12, 13]

Аномалии обновления – это группа проблем, возникающих при операциях вставки, обновления и удаления данных (INSERT, UPDATE, DELETE) в отношениях с избыточностью. Эти проблемы могут вызывать некорректное поведение системы, поскольку для обновления информации о какой-то сущности приходится редактировать данные в нескольких местах базы.

Вторая нормальная форма дополняет 1НФ и добавляет требование, чтобы каждое поле зависело от всего первичного ключа. Сущность находится во второй нормальной форме, если она соответствует требованиям 1НФ и при этом каждый неключевой атрибут полностью зависит от первичного ключа.

Цель нормализации до 2НФ – устранить ситуации, в которых отношение имеет составной первичный ключ, и значение одного из неключевых атрибутов зависит только от части этого ключа. [14]

Для нормализации до 2НФ сущность заменяется несколькими проекциями, которые в совокупности представляют ту же самую информацию. Например, если в переменной отношений А есть поля В, С, D, E, где В и С составляют первичный ключ, и поле D зависит только от В, а не от С, то А можно разделить на две проекции А1 и А2. В А1 будут поля В и D, а в А2 – В, С и E, где В будет как частью первичного ключа, так и внешним ключом, ссылающимся на А1. Этот подход позволяет избежать зависимости D от С, сохраняя при этом исходные данные.

Третья нормальная форма расширяет 2НФ и добавляет условие устранения транзитивных зависимостей. Сущность находится в 3НФ, если она соответствует требованиям 2НФ и ни один неключевой атрибут не зависит транзитивно от первичного ключа. Нормализация до 3НФ устраняет случаи, когда один неключевой атрибут зависит от другого неключевого атрибута.

Метод нормализации схож с переходом ко второй нормальной форме: необходимо разбить сущность на несколько проекций. Например, если сущность А имеет поля В, С и D, где В – первичный ключ, а D зависит от С, то можно разделить А на А1 и А2. В А1 будут поля С и D с С в качестве первичного ключа, а в А2 – поля В и С, где В – первичный ключ, а С – внешний ключ, ссылающийся на А1. [15]

2.2.3 Нереляционные базы данных

NoSQL – это обширная категория систем управления базами данных, которые не соответствуют традиционной реляционной модели. Основные типы NoSQL баз данных включают:

- документо-ориентированные базы данных;
- базы данных ключ-значение;
- графовые базы данных;
- колонно-ориентированные базы данных.

Далее данные типы будут рассмотрены более подробно.

Документо-ориентированные базы данных хранят информацию в виде документов, которые имеют свою структуру и могут включать различные типы данных. Документы группируются в коллекции, которые могут иметь разные схемы. Для работы с такими базами данных используются специализированные языки запросов, позволяющие обращаться к данным по

их атрибутам. Рассмотрим несколько примеров документо-ориентированных хранилищ данных:

1 MongoDB – одна из самых известных и мощных документо-ориентированных баз данных, поддерживающая различные форматы документов, такие как JSON, BSON и XML. MongoDB характеризуется высокой производительностью, масштабируемостью и гибкостью, а также предоставляет множество функций для работы с данными, таких как агрегация, индексация, шардирование и репликация.

2 Firebase – облачная платформа, предлагающая документо-ориентированную базу данных в реальном времени, известную как Cloud Firestore. Firebase позволяет хранить и синхронизировать данные между разными клиентами и серверами, а также предлагает различные сервисы для разработки мобильных и веб-приложений, такие как аутентификация, хостинг и аналитика.

Далее перейдем к базам данных ключ-значение. Эти базы данных хранят данные в виде пар ключ-значение, где ключ служит уникальным идентификатором, а значение может быть любым типом данных. Базы данных ключ-значение обеспечивают быстрый доступ к данным по ключу, однако не поддерживают сложные запросы и связи между данными. Основные особенности таких баз данных включают:

1 Горизонтальная масштабируемость. Данные могут быть распределены по различным узлам или серверам без необходимости объединения таблиц или синхронизации схем.

2 Хранение неструктурированных данных. Идеально подходят для работы с текстами, изображениями и видео, которые могут иметь различные размеры и форматы.

3 Гибкость в структуре данных. Не требуют жесткой схемы или типизации данных, что позволяет легко изменять свойства и структуру.

4 Высокая производительность. Обеспечивают низкую задержку, так как данные обрабатываются в оперативной памяти или на быстрых носителях.

Примеры баз данных ключ-значение включают:

1 Redis – популярная база данных, которая хранит данные в оперативной памяти и поддерживает различные типы значений, такие как строки, списки и множества. Redis также предлагает функции, такие как транзакции, репликация и кэширование.

2 DynamoDB – облачная база данных от Amazon Web Services, которая хранит данные на твердотельных накопителях, обеспечивая высокую доступность и надежность. DynamoDB поддерживает различные типы

значений и позволяет выполнять условные запросы и обновления.

Графовые базы данных хранят данные в виде узлов и ребер, представляющих сущности и связи между ними. Они подходят для моделирования сложных сетей и отношений, таких как социальные сети и рекомендательные системы. Для работы с графовыми базами данных используются специализированные языки запросов, которые позволяют искать пути и паттерны в графах.

Графовые базы данных используют математическую теорию графов для отображения и обработки связей между данными. Они предлагают ряд преимуществ по сравнению с реляционными и другими типами нереляционных баз данных:

1 Эффективность в запросах. Позволяют быстро выполнять запросы, требующие анализа связей, такие как поиск кратчайшего пути или выявление аномалий.

2 Гибкость и масштабируемость. Не требуют жесткой схемы данных, что позволяет добавлять, удалять и изменять узлы и ребра без нарушения целостности данных.

Графовые базы данных могут быть разделены на два основных типа: базы данных свойственных графов, где узлы и ребра имеют атрибуты, и базы данных знаний, где узлы и ребра имеют семантические метки. Примеры баз данных свойственных графов: Neo4j и ArangoDB; примеры баз данных знаний: AllegroGraph и Datomic.

Для работы с графовыми базами данных применяются специальные языки запросов, такие как Cypher, Gremlin и SPARQL.

Колонно-ориентированные базы данных представляют собой один из типов NoSQL систем управления базами данных, которые хранят данные в виде колонок, а не строк. Это подход позволяет эффективно организовывать и обрабатывать большие объемы данных, особенно в сценариях, требующих высокой производительности при выполнении аналитических запросов. Колонно-ориентированные базы данных оптимизированы для работы с большими наборами данных и часто используются в аналитических приложениях, бизнес-аналитике и хранилищах данных.

Основные характеристики колонно-ориентированных баз данных включают в себя:

1 Хранение по колонкам. Данные хранятся в виде отдельных колонок, что позволяет эффективно считывать только необходимые данные без необходимости загружать всю строку. Это значительно повышает производительность при выполнении агрегирующих запросов и выборок по

определённым колонкам.

2 Оптимизация для чтения. Колонно-ориентированные базы данных идеально подходят для сценариев, где требуется высокая скорость обработки запросов и анализ данных. Это достигается благодаря тому, что данные одной колонки хранятся вместе, что позволяет лучше использовать кэш и сократить время доступа к данным.

3 Сжатие данных. Поскольку данные в колонках часто имеют одинаковый тип, это позволяет применять более эффективные алгоритмы сжатия, что приводит к уменьшению объема хранимых данных и экономии дискового пространства.

4 Гибкость схемы. Колонно-ориентированные базы данных могут поддерживать динамические схемы, что позволяет добавлять новые колонки без необходимости изменения всей структуры базы данных.

Примеры колонно-ориентированных баз данных:

1 Apache Cassandra. Это распределенная колонно-ориентированная база данных, предназначенная для обработки больших объемов данных. Cassandra обеспечивает высокую доступность и отказоустойчивость благодаря дублированию данных на нескольких узлах. Она идеально подходит для работы с веб-приложениями и IoT-решениями.

2 HBase. Это колонно-ориентированная база данных, построенная на базе Hadoop. HBase позволяет обрабатывать большие объемы данных в реальном времени и подходит для аналитических и отчетных систем, работающих с данными в распределенной среде.

Таким образом, NoSQL базы данных предлагают гибкость, масштабируемость и производительность, что делает их подходящими для работы с большими объемами данных и динамически меняющимися схемами.

2.3 Выбор базы данных для разработки предметной области

Исходя из анализа подходов, представленных в предыдущем разделе, можно заключить, что предметная область библиотеки фильмов и сериалов будет реализована с использованием реляционных баз данных. Это объясняется тем, что в данной области четко определены атрибуты, сущности и связи, которые удобно отображать в виде таблиц и ключей в рамках реляционной модели.

Для обеспечения корректной работы системы необходимо гарантировать целостность и согласованность данных, что особенно важно для избежания ошибок при бронировании фильмов, управлении коллекциями и

размещении информации о сериалах. Также для аналитики и создания рекомендательной системы потребуется выполнять сложные SQL-запросы, что хорошо поддерживается реляционными базами данных.

В качестве базы данных для разработки была выбрана PostgreSQL по нескольким причинам. Эта система управления базами данных (СУБД) обладает высокой производительностью и может эффективно работать с большими объемами данных, обслуживая множество пользователей одновременно. [16, 17]

Преимущества PostgreSQL заключаются в следующем:

1 Эффективный механизм обработки запросов. PostgreSQL использует продвинутые алгоритмы, что позволяет оптимально работать с данными.

2 Поддержка параллелизма. Это означает, что несколько запросов могут выполняться одновременно, что существенно увеличивает общую производительность.

3 Масштабируемость. PostgreSQL может масштабироваться как вертикально, увеличивая ресурсы на одном сервере, так и горизонтально, распределяя нагрузку между несколькими серверами.

Система также подходит для различных типов приложений, включая:

1 Корпоративные решения. PostgreSQL идеальна для крупных организаций, которым необходимы высокая производительность и надежность.

2 Веб-приложения. Эта СУБД часто используется для разработки масштабируемых и надежных веб-приложений.

3 Интернет-магазины. PostgreSQL может эффективно обрабатывать транзакции и управлять товарными запасами и клиентами.

4 Социальные сети. Она подходит для создания платформ, где необходимо хранение пользовательских данных и обеспечение безопасности.

PostgreSQL обеспечивает высокий уровень безопасности данных, предлагая разнообразные методы аутентификации, такие как пароли, сертификаты и двухфакторная аутентификация. Она также поддерживает ролевые и объектно-ориентированные схемы авторизации, что позволяет администраторам контролировать доступ пользователей к данным и функциям. База данных включает методы шифрования, как для хранения, так и для передачи данных, защищая информацию от несанкционированного доступа. [18, 19]

Таким образом, PostgreSQL была выбрана благодаря своей высокой производительности, масштабируемости и богатым возможностям безопасности.

3 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

3.1 Описание сущностей

Стартовой точкой в проектировании базы данных является описание основных сущностей, которые взаимосвязаны между собой.

Всего описываемая база данных включает в себя 10 сущностей. Постепенно опишем каждую из них:

1 FilmsSerials (фильмы и сериалы):

- film_id – идентификатор фильма или сериала (первичный ключ);
- title – название;
- release_year – год выпуска;
- duration – продолжительность (в минутах);
- plot_summary – краткое содержание;
- genre_id – идентификатор жанра (внешний ключ к сущности Genres);
- director_id – идентификатор режиссера (внешний ключ к сущности Directors);

– average_user_rating – средняя пользовательская оценка (десятичное, 2 знака после запятой).

2 Genres (жанры):

- genre_id – идентификатор жанра (первичный ключ);
- genre_name – название жанра.

3 Actors (актеры):

- actor_id – идентификатор актера (первичный ключ);
- first_name – имя актера;
- last_name – фамилия актера;
- date_of_birth – дата рождения;
- nationality – национальность.

4 Directors (режиссеры):

- director_id – идентификатор режиссера (первичный ключ);
- first_name – имя режиссера;
- last_name – фамилия режиссера;
- date_of_birth – дата рождения;
- nationality – национальность.

5 Ratings (оценки):

- rating_id – идентификатор оценки (первичный ключ);
- film_or_serial_id – идентификатор фильма или сериала (внешний ключ к сущности FilmsSerials);

- user_id – идентификатор пользователя (внешний ключ к сущности UserAccounts);

- user_rating – оценка пользователя (десятичное, от 1.0 до 10.0);

- review_text – текст отзыва.

6 Awards (награды):

- award_id – идентификатор награды (первичный ключ);

- film_or_serial_id – идентификатор фильма или сериала (внешний ключ к сущности FilmsSerials);

- award_name – название награды;

- award_category – категория награды;

- year_received – год получения.

7 Languages (языки):

- language_id – идентификатор языка (первичный ключ);

- language_name – название языка;

- film_id – идентификатор фильма (внешний ключ к сущности FilmsSerials).

8 Roles (роли):

- role_id – идентификатор роли (первичный ключ);

- role_name – название роли.

9 UserAccounts (пользователи):

- user_id – идентификатор пользователя (первичный ключ);

- role_id – идентификатор роли (внешний ключ к сущности Roles);

- username – имя пользователя;

- email – почта;

- password – пароль (зашифрованный);

- registration_date – дата регистрации.

10 WatchHistory (история просмотра):

- history_id – идентификатор истории (первичный ключ);

- user_id – идентификатор пользователя (внешний ключ к сущности UserAccounts);

- film_or_serial_id – идентификатор фильма или сериала (внешний ключ к сущности FilmsSerials);

- date_watched – дата просмотра.

Такое описание подчеркивает структуру базы данных и ее связь с функциональными требованиями, включая хранение информации о фильмах, пользователях, ролях, жанрах и историях просмотров.

3.2 Разработка даталогической модели базы данных

По результатам разработки описания сущностей базы данных построим даталогическую модель базы данных, в которой будут определены типы данных для каждого поля, участвующего в описании разрабатываемой предметной области. ER-диаграмма базы данных представлена на рисунке 3.1:

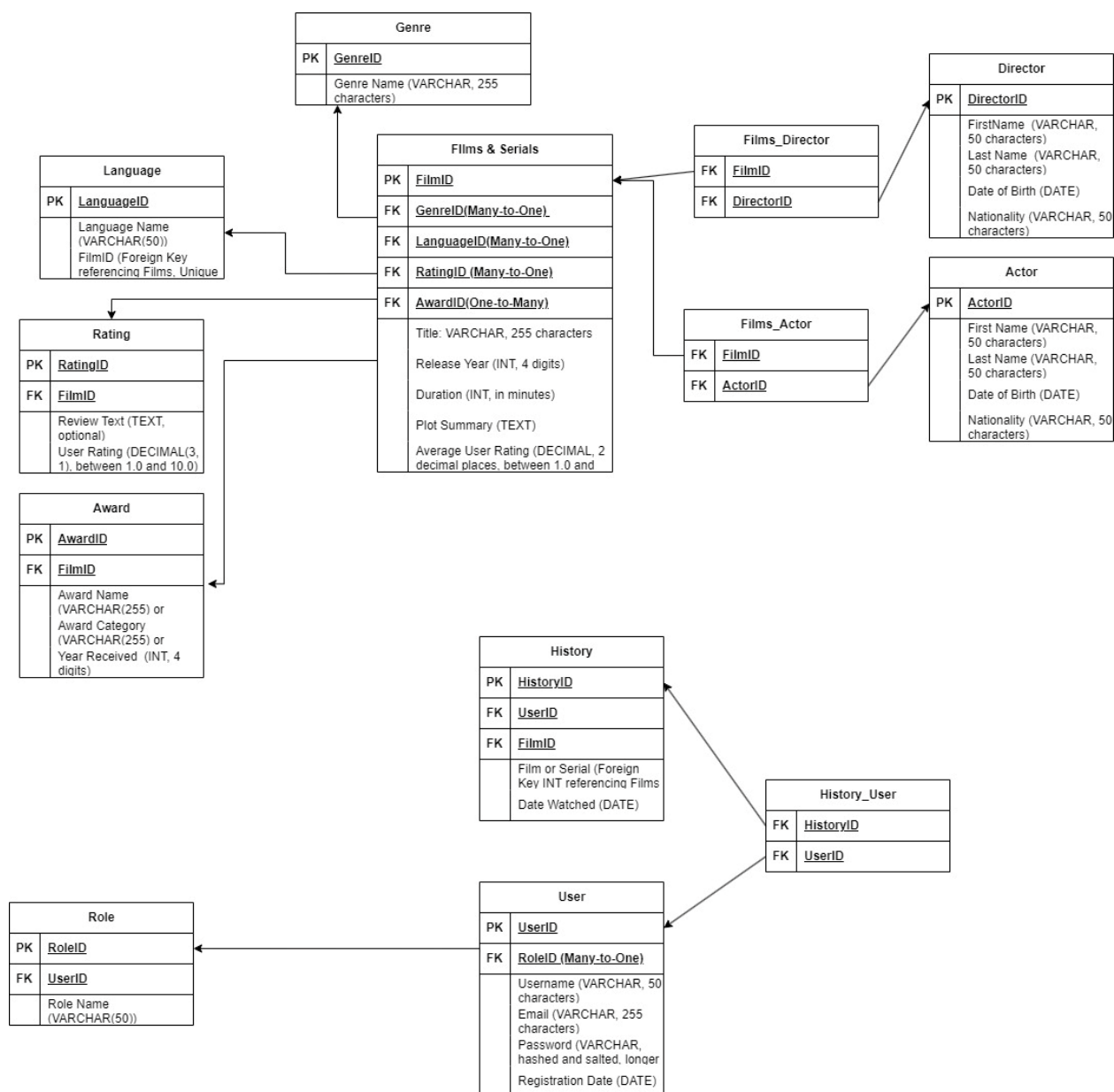


Рисунок 3.1 – ER-диаграмма базы данных

Таким образом мы получаем готовый образец для переноса сущностей диаграммы на физический носитель информации средствами PostgreSQL.

4 РАЗРАБОТКА БАЗЫ ДАННЫХ

4.1 Создание исходных таблиц, индексов и ограничений

Для начала работы с описанными в диаграмме сущностями, необходимо написать SQL-скрипты для создания указанных таблиц, добавить к ним ограничения на внешние ключи и использовать индексы по мере необходимости. Весь исходный код для данных скриптов расположен в Приложении А, а здесь разместим небольшое представление на рисунке 4.1, которое продемонстрирует набор индексов, используемых в базе данных.

```
-- Индексы для таблицы FilmsSerials
CREATE INDEX index_films_title ON FilmsSerials(title);
CREATE INDEX index_films_genre_id ON FilmsSerials(genre_id);
CREATE INDEX index_films_director_id ON FilmsSerials(director_id);

-- Индексы для таблицы Genres
CREATE INDEX index_genres_name ON Genres(genre_name);

-- Индексы для таблицы Actors
CREATE INDEX index_actors_last_name ON Actors(last_name);
CREATE INDEX index_actors_first_name ON Actors(first_name);

-- Индексы для таблицы Directors
CREATE INDEX index_directors_last_name ON Directors(last_name);
CREATE INDEX index_directors_first_name ON Directors(first_name);

-- Индексы для таблицы Ratings
CREATE INDEX index_ratings_film_id ON Ratings(film_or_serial_id);
CREATE INDEX index_ratings_user_id ON Ratings(user_id);

-- Индексы для таблицы Awards
CREATE INDEX index_awards_film_id ON Awards(film_or_serial_id);
CREATE INDEX index_awards_year_received ON Awards(year_received);

-- Индексы для таблицы Languages
CREATE INDEX index_languages_name ON Languages(language_name);

-- Индексы для таблицы UserAccounts
CREATE INDEX index_useraccounts_username ON UserAccounts(username);
CREATE INDEX index_useraccounts_email ON UserAccounts(email);

-- Индексы для таблицы WatchHistory
CREATE INDEX index_watchhistory_user_id ON WatchHistory(user_id);
CREATE INDEX index_watchhistory_film_id ON WatchHistory(film_or_serial_id);
CREATE INDEX index_watchhistory_date_watched ON WatchHistory(date_watched);
```

Рисунок 4.1 – Набор индексов, используемых в проектируемой базе данных

4.2 Создание хранимых процедур

К хранимым процедурам отнесем следующие процедуры:

1 Процедура FindFilmByTitle служит для поиска фильма по названию и вывода информации по найденному фильму. Сама процедура представлена на рисунке 4.2:

```
CREATE OR REPLACE PROCEDURE FindFilmByTitle(IN film_title VARCHAR(255))
AS $$
DECLARE
    film_info RECORD;
    language_info RECORD;
    award_info RECORD;
    director_info RECORD;
    user_info RECORD;
    actor_info RECORD;
    genre_info RECORD;
BEGIN
    -- Select the movie based on the provided title
    SELECT * INTO film_info FROM "Films" WHERE "Title" = film_title;

    -- Check if a movie with the provided title exists
    IF FOUND THEN
        -- Fetch language details based on LanguageID_FK from the Languages table
        SELECT * INTO language_info FROM "Languages" WHERE "LanguageID" = film_info."LanguageID_FK";
        SELECT * INTO award_info FROM "Awards" WHERE "AwardID" = film_info."AwardID_FK";
        SELECT * INTO director_info FROM "Directors" WHERE "DirectorID" = film_info."DirectorID_FK";

        RAISE NOTICE 'Film Found - MovieID: %, Title: %,
            Release Year: %, Duration: %,
            Plot Summary: %, Average rating: %,
            Language: %, Award: %,
            Director: %,
            film_info."FilmID", film_info."Title",
            film_info."ReleaseYear", film_info."Duration",
            film_info."PlotSummary", film_info."AverageRating",
            language_info."Language", award_info."Award",
            director_info."LastName";

        -- Fetch all actors for the film based on FilmsActors
        FOR actor_info IN
            SELECT "Actors"."LastName"
            FROM "Actors"
            INNER JOIN "FilmsActors" ON "Actors"."ActorID" = "FilmsActors"."ActorID_FK"
            WHERE "FilmsActors"."FilmID_FK" = film_info."FilmID"
        LOOP
            RAISE NOTICE 'Actor: %', actor_info."LastName";
        END LOOP;

        -- Fetch all genres for the film based on FilmsGenres
        FOR genre_info IN
            SELECT "Genres"."Genre"
            FROM "Genres"
            INNER JOIN "FilmsGenres" ON "Genres"."GenreID" = "FilmsGenres"."GenreID_FK"
            WHERE "FilmsGenres"."FilmID_FK" = film_info."FilmID"
        LOOP
            RAISE NOTICE 'Genre: %', genre_info."Genre";
        END LOOP;
    ELSE
        RAISE NOTICE 'Film Not Found';
    END IF;
END;
$$ LANGUAGE plpgsql;
```

Рисунок 4.2 – Процедура для поиска информации о фильме

2 Аналогичным образом выглядят процедура для поиска пользователя и вывода основной информации о нем get_user_by_name, реализация которой

представлена на рисунке 4.3:

```
CREATE OR REPLACE PROCEDURE get_user_by_name(  
    IN user_name VARCHAR  
)  
AS $$  
DECLARE  
    user_record "Users"%ROWTYPE;  
    role_name VARCHAR;  
BEGIN  
    SELECT * INTO user_record FROM "Users" WHERE "Username" = user_name;  
  
    -- Fetch the role name based on RoleID_FK from the Roles table  
    SELECT "Role" INTO role_name FROM "Roles" WHERE "RoleID" = user_record."RoleID_FK";  
  
    -- Print or do something with the user details including the role name  
    RAISE NOTICE 'User ID: %, Role: %, U  
        Username: %, Email: %, P  
        Password: %, RegistrationDate: %',  
        user_record."UserID", role_name,  
        user_record."Username", user_record."Email",  
        user_record."Password", user_record."RegistrationDate";  
END;  
$$ LANGUAGE plpgsql;
```

Рисунок 4.3 – Процедура для поиска информации о пользователе

3 Также упомянем реализацию процедуры для добавления пользователя create_user, которая представлена на рисунке 4.4:

```
CREATE OR REPLACE PROCEDURE create_user(  
    IN roleid_fk INT,  
    IN new_username VARCHAR,  
    IN new_email VARCHAR,  
    IN new_password VARCHAR  
)  
AS $$  
BEGIN  
    INSERT INTO "Users" ("RoleID_FK", "Username", "Email", "Password", "RegistrationDate")  
    VALUES (roleid_fk, new_username, new_email, new_password, CURRENT_DATE);  
END;  
$$ LANGUAGE plpgsql;
```

Рисунок 4.4 – Процедура для добавления нового пользователя

4 Аналогичным образом выглядят процедура для добавления отзыва на фильм create_review, реализация которой представлена на рисунке 4.5:

```

CREATE OR REPLACE PROCEDURE create_review(
    IN username VARCHAR,
    IN film_title VARCHAR,
    IN rating INT,
    IN review_text TEXT
)
AS $$
DECLARE
    user_id INT;
    film_id INT;
BEGIN
    -- Get UserID based on the provided username
    SELECT "UserID" INTO user_id FROM "Users" WHERE "Username" = username;

    -- Get FilmID based on the provided film_title
    SELECT "FilmID" INTO film_id FROM "Films" WHERE "Title" = film_title;

    -- Check if the user and film exist
    IF user_id IS NOT NULL AND film_id IS NOT NULL THEN
        -- Insert the review
        INSERT INTO "Reviews" ("UserID_FK", "FilmID_FK", "Rating", "ReviewText")
        VALUES (user_id, film_id, rating, review_text);
    ELSE
        RAISE EXCEPTION 'User or film not found';
    END IF;
END;
$$ LANGUAGE plpgsql;

```

Рисунок 4.5 – Процедура для добавления отзыва

5 Процедура `update_user` служит для обновления данных пользователя. Сама процедура представлена на рисунке 4.6:

```

CREATE OR REPLACE PROCEDURE update_user(
    IN user_id INT,
    IN updated_username VARCHAR,
    IN updated_email VARCHAR,
    IN updated_password VARCHAR
)
AS $$
BEGIN
    UPDATE "Users"
    SET "Username" = updated_username,
        "Email" = updated_email,
        "Password" = updated_password
    WHERE "UserID" = user_id;
END;
$$ LANGUAGE plpgsql;

```

Рисунок 4.6 – Процедура для обновления данных пользователя

6 Процедура `delete_user` служит для удаления пользователя из базы данных. Сама процедура представлена на рисунке 4.7:


```

CREATE OR REPLACE PROCEDURE delete_user(
    IN user_name VARCHAR
)
AS $$
BEGIN
    DELETE FROM "Users" WHERE "Username" = user_name;
END;
$$ LANGUAGE plpgsql;

```

Рисунок 4.7 – Процедура для удаления пользователя

Таким образом, были реализованы хранимые процедуры, которые служат для упрощения и стандартизации работы с базой данных, обеспечивая автоматизацию, целостность данных и защиту от ошибок или несанкционированного доступа.

4.3 Реализация триггеров

В разрабатываемой базе данных в первую очередь необходимо добавить триггер для расчета средней оценки фильма, в зависимости от того поставили или удалили оценку. Вариант реализации такого триггера представлен на рисунке 4.8:

```

CREATE TRIGGER update_film_average_rating_trigger
AFTER INSERT OR UPDATE OR DELETE ON "Reviews"
FOR EACH ROW
EXECUTE FUNCTION update_film_average_rating();

```

Рисунок 4.8 – Реализация триггера для расчета средней оценки фильма

Функция, которая вызывается в данном триггере представлена на рисунке 4.9:

```

CREATE OR REPLACE FUNCTION update_film_average_rating()
RETURNS TRIGGER AS $$
BEGIN
    -- Update average rating in Films table when a new review is inserted, updated, or deleted
    UPDATE "Films"
    SET "AverageRating" = (
        SELECT COALESCE(AVG("Rating"), 0) FROM "Reviews" WHERE "FilmID_FK" = NEW."FilmID_FK"
    )
    WHERE "FilmID" = NEW."FilmID_FK";

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

```

Рисунок 4.9 – Реализация функции для расчета средней оценки фильма

Также в разрабатываемой базе данных необходимо добавить триггер для добавления записи о просмотре фильма в историю просмотров после создания отзыва на фильм. Вариант реализации такого триггера представлен на рисунке 4.10:

```
CREATE TRIGGER review_created_trigger
AFTER INSERT ON "Reviews"
FOR EACH ROW
EXECUTE FUNCTION update_history_insert_review();
```

Рисунок 4.10 – Реализация триггера для добавления просмотров в историю

Функция, которая вызывается в данном триггере представлена на рисунке 4.11:

```
CREATE OR REPLACE FUNCTION update_history_insert_review()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO "Histories" ("FilmID_FK", "DateWatched", "UserID_FK")
    VALUES (NEW."FilmID_FK", CURRENT_DATE, NEW."UserID_FK");
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Рисунок 4.11 – Реализация функции для добавления просмотров в историю

Таким образом, данные триггеры автоматизируют обновление данных в связанных таблицах, поддерживая их согласованность. Они сокращают ручную работу и минимизируют вероятность ошибок, обеспечивая актуальность информации о взаимодействии пользователей с фильмами.

5 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

5.1 Описание разработанного программного средства

Программное средство представляет собой цифровую библиотеку фильмов, построенную с использованием Tkinter для интерфейса и PostgreSQL для хранения данных.

При запуске приложения появляется главное окно, предлагающее пользователю либо авторизоваться, либо зарегистрироваться, либо покинуть приложение. Данное окно продемонстрировано на рисунке 5.1:

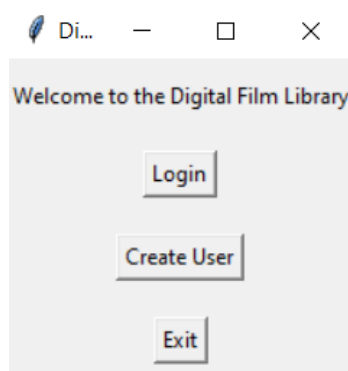


Рисунок 5.1 – Окно, появляющееся при входе в приложение

Основные функции приложения:

1 Авторизация. Пользователь вводит имя и пароль для входа в систему. Если данные верны, пользователю предоставляется доступ в зависимости от его роли (обычный пользователь или администратор). В случае ошибки отображается соответствующее сообщение. Пример авторизации показан на рисунке 5.2:

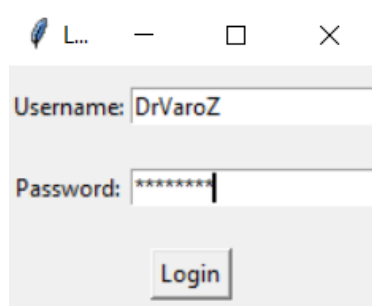


Рисунок 5.2 – Окно для авторизации

2 Регистрация нового пользователя. Пользователь может зарегистрироваться, указав имя, пароль и роль. После регистрации новый пользователь добавляется в базу данных. Окно для регистрации показано на рисунке 5.3:

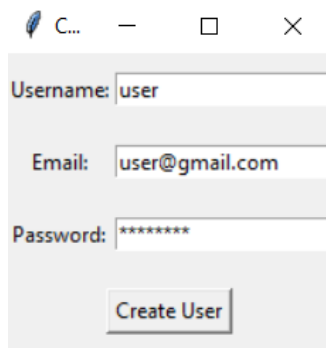
A screenshot of a web application window titled 'C...' with standard minimize, maximize, and close buttons. The window contains a registration form with three input fields: 'Username:' with the value 'user', 'Email:' with the value 'user@gmail.com', and 'Password:' with the value '*****'. Below the fields is a button labeled 'Create User'.

Рисунок 5.3 – Окно для регистрации

3 Администратор имеет доступ к дополнительным функциям, таким как управление пользователями и фильмами. Он может искать, редактировать или удалять пользователей. Также администратор может искать фильмы по названию, добавлять новые фильмы и управлять отзывами пользователей. На рисунке 5.4 продемонстрирован интерфейс приложения для администратора:

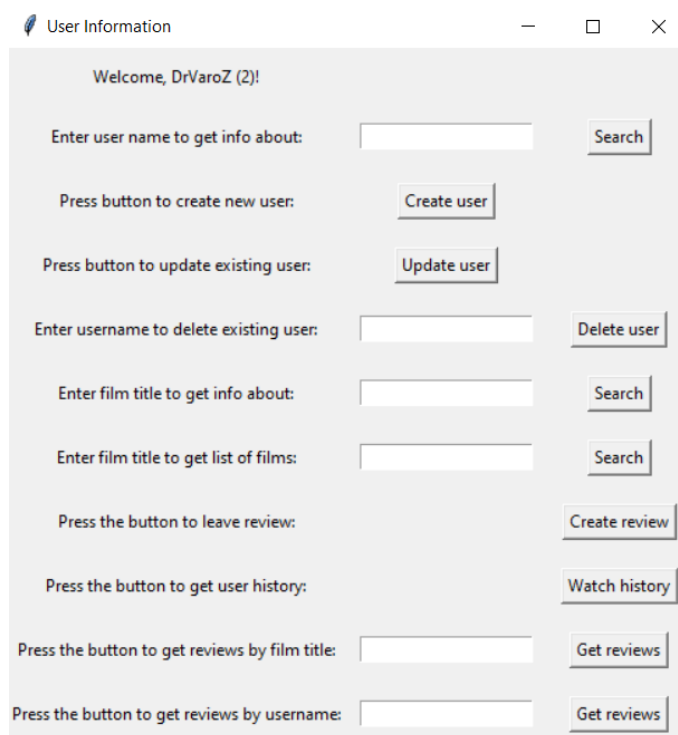
A screenshot of a web application window titled 'User Information' with standard minimize, maximize, and close buttons. The window displays a 'Welcome, DrVaroZ (2)!' message. Below the message are several interactive elements: a search bar for user information with a 'Search' button; buttons for 'Create user', 'Update user', and 'Delete user'; a search bar for film information with a 'Search' button; a search bar for film titles with a 'Search' button; a 'Create review' button; a 'Watch history' button; a search bar for reviews by film title with a 'Get reviews' button; and a search bar for reviews by username with a 'Get reviews' button.

Рисунок 5.4 – Интерфейс приложения у администратора

4 Обычный пользователь имеет доступ к таким функциям, как поиск информации о фильмах, работа с отзывами и просмотр истории. Он может искать фильмы по частичному совпадению названия, и результаты отображаются в виде списка. Пользователь может оставлять отзывы на фильмы. Отзывы сохраняются в базе данных, и пользователи могут просматривать их. Также пользователь может видеть историю своих взаимодействий с приложением, включая оставленные отзывы. На рисунке 5.5 продемонстрирован интерфейс приложения для обычного пользователя:

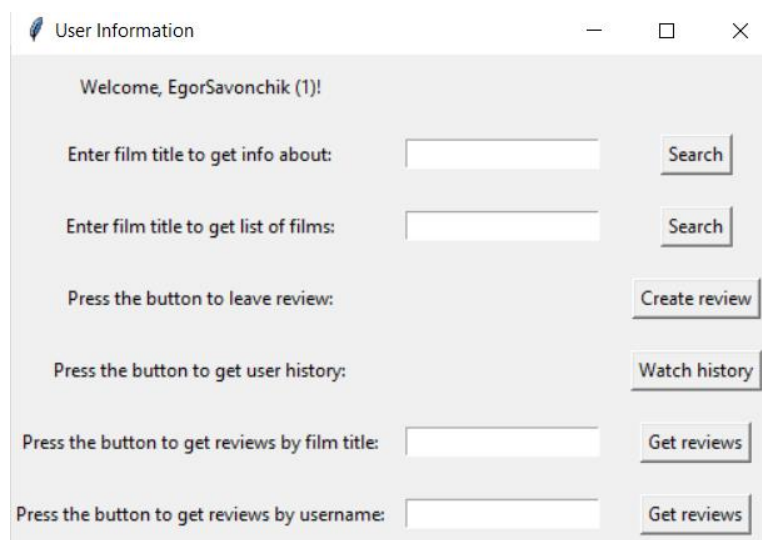


Рисунок 5.5 – Интерфейс приложения у обычного пользователя

Приложение использует библиотеку `psycopg2` для подключения и взаимодействия с базой данных PostgreSQL. Все SQL-запросы параметризованы для обеспечения безопасности от SQL-инъекций.

В целом, приложение проделало значительный путь в реализации основных функций и предоставляет основу для дальнейшего улучшения и добавления новых возможностей.

ЗАКЛЮЧЕНИЕ

Результатом работы данного курсового проекта является создание базы данных для цифровой библиотеки фильмов и сериалов, которая включает в себя все ключевые аспекты, необходимые для организации и хранения информации о фильмах и сериалах. Проект обеспечил хранение данных о кинокартинах, актерах, режиссерах, жанрах, рейтингах, а также информации о пользователях, их действиях, таких как просмотр.

В качестве системы управления базами данных была выбрана реляционная СУБД PostgreSQL, которая предоставила надежную и масштабируемую платформу для реализации проекта. PostgreSQL позволил эффективно организовать данные с возможностью гибкого поиска и анализа, обеспечивая при этом высокую производительность и безопасность данных. Использование этой СУБД также обеспечило поддержку сложных запросов, что важно для работы с большим объемом информации в цифровой библиотеке. [20]

В ходе реализации проекта были достигнуты поставленные цели: разработан функционал для организации и управления коллекцией фильмов и сериалов, а также обеспечена возможность эффективного взаимодействия пользователей с системой. Интерактивные функции, такие как хранение истории просмотров, обеспечивают удобство использования для конечного пользователя.

Таким образом, цели курсовой работы были успешно достигнуты, и проект продемонстрировал возможность создания цифровой библиотеки фильмов и сериалов с использованием современных технологий и подходов к организации баз данных.

СПИСОК ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

- [1] IMDb.com [Электронный ресурс]:IMDb. Режим доступа: <https://www.imdb.com>. Дата доступа: 01.12.2024.
- [2] Letterboxd.com [Электронный ресурс]:Letterboxd. Режим доступа: <https://letterboxd.com>. Дата доступа: 01.12.2024.
- [3] Netflix.com [Электронный ресурс]:Netflix. Режим доступа: <https://netflix.com>. Дата доступа: 01.12.2024.
- [4] Иванов А.А. Проектирование баз данных: учебник для вузов / А.А. Иванов. – М.: Издательство «Академия», 2023. – 512 с.
- [5] Петров В.В. Основы работы с СУБД MySQL / В.В. Петров. – СПб.: Питер, 2023. – 1024 с.
- [6] Корнелл, Р. PostgreSQL: Искусство и практика / Р. Корнелл. – М.: Вильямс, 2018. – 480 с.
- [7] Бен-Ари, М. Основы проектирования и администрирования баз данных / М. Бен-Ари. – М.: Диалектика, 2017. – 416 с.
- [8] Майер, Г. Проектирование реляционных баз данных: Основы и практика / Г. Майер. – М.: Издательский дом «ДиаСофт», 2019. – 352 с.
- [9] Чесноков, В. А. Реляционные базы данных: Теория и практика / В. А. Чесноков. – СПб.: Питер, 2017. – 344 с.
- [10] Резников, М. В. Цифровая библиотека: Проектирование и разработка / М. В. Резников. – М.: Издательство МГУ, 2018. – 256 с.
- [11] Шилдт, Х. SQL: Полное руководство / Х. Шилдт. – М.: Вильямс, 2020. – 832 с.
- [12] Коэн, И. Разработка приложений для цифровых библиотек / И. Коэн. – М.: Наука, 2021. – 400 с.
- [13] Гимпель, И. М. Современные информационные системы: Проектирование и управление проектами / И. М. Гимпель. – М.: БХВ-Петербург, 2016. – 448 с.
- [14] Базы данных в киноиндустрии [Электронный ресурс]: DataArt. Режим доступа: <https://dataart.ru/database-in-cinema-industry/>. Дата доступа: 01.12.2024.
- [15] Системы управления базами данных. Основные характеристики. [Электронный ресурс]: DBstudy. – Режим доступа: https://dbstudy.net/814892/informatika/osnovnye_harakteristiki_sistem_upravleniya_bazami_dannyh. Дата доступа: 01.12.2024.
- [16] Базы данных и их роль в современном мире. [Электронный ресурс]: ITtoday. – Режим доступа: <https://ittoday.ru/databases-and-their-role-in-modern->

world/. Дата доступа: 01.12.2024.

[17] Смирнов И.И. Базы данных и Delphi: учебное пособие / И.И. Смирнов. – М.: Издательство «Академия», 2023. – 350 с.

[18] Базы данных в киноиндустрии: новые тенденции [Электронный ресурс]: CinemaTrends. Режим доступа: <https://cinematrends.ru/database-trends-in-cinema-industry/>. Дата доступа: 01.12.2024.

[19] Применение баз данных в кинотеатрах: кейс-стади [Электронный ресурс]: CinemaCase. – Режим доступа: <https://cinemacase.ru/>. Дата доступа: 01.12.2024.

[20] Базы данных и их роль в цифровой экономике [Электронный ресурс]: ITtoday. –Режим доступа: <https://ittoday.ru/databases-and-their-role-in-digital-economy/>. Дата доступа: 01.12.2024.

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг кода программы

```
CREATE TABLE public."Actors"
(
    ActorID integer NOT NULL PRIMARY KEY,
    FirstName character varying(50) NOT NULL,
    LastName character varying(50) NOT NULL,
    DateBirthdate date NOT NULL,
);

CREATE TABLE public."Awards"
(
    AwardID integer NOT NULL PRIMARY KEY,
    Award character varying(255) NOT NULL,
    YearRecieved integer NOT NULL,
);

CREATE TABLE public."Genres"
(
    GenreID integer NOT NULL PRIMARY KEY,
    Genre character varying(50) NOT NULL,
);

CREATE TABLE public."Directors"
(
    DirectorID integer NOT NULL PRIMARY KEY,
    FirstName character varying(50) NOT NULL,
    LastName character varying(50) NOT NULL,
    DateBirthdate date NOT NULL,
);

CREATE TABLE public."Films"
(
    FilmID integer NOT NULL PRIMARY KEY,
    Title character varying(255) NOT NULL,
    ReleaseYear integer NOT NULL,
    Duration integer NOT NULL,
    AverageRating numeric(3, 1) NOT NULL,
    PlotSummary text NOT NULL,
    CONSTRAINT "LanguageID_FK" FOREIGN KEY (LanguageID_FK) REFERENCES
public."Languages" ("LanguageID") MATCH SIMPLE
    CONSTRAINT "AwardID_FK" FOREIGN KEY (AwardID_FK) REFERENCES
public."Awards" ("AwardID") MATCH SIMPLE
    CONSTRAINT "DirectorID_FK" FOREIGN KEY (DirectorID_FK) REFERENCES
public."Directors" ("DirectorID") MATCH SIMPLE
    CONSTRAINT "ReviewID_FK" FOREIGN KEY (ReviewID_FK) REFERENCES
public."Reviews" ("ReviewID") MATCH SIMPLE
);
-- Индексы для таблицы FilmsSerials
CREATE INDEX index_films_title ON FilmsSerials(title);
```

```

CREATE INDEX index_films_genre_id ON FilmsSerials(genre_id);
CREATE INDEX index_films_director_id ON FilmsSerials(director_id);

-- Индексы для таблицы Genres
CREATE INDEX index_genres_name ON Genres(genre_name);

-- Индексы для таблицы Actors
CREATE INDEX index_actors_last_name ON Actors(last_name);
CREATE INDEX index_actors_first_name ON Actors(first_name);

-- Индексы для таблицы Directors
CREATE INDEX index_directors_last_name ON Directors(last_name);
CREATE INDEX index_directors_first_name ON Directors(first_name);

-- Индексы для таблицы Ratings
CREATE INDEX index_ratings_film_id ON Ratings(film_or_serial_id);
CREATE INDEX index_ratings_user_id ON Ratings(user_id);

-- Индексы для таблицы Awards
CREATE INDEX index_awards_film_id ON Awards(film_or_serial_id);
CREATE INDEX index_awards_year_received ON Awards(year_received);

-- Индексы для таблицы Languages
CREATE INDEX index_languages_name ON Languages(language_name);

-- Индексы для таблицы UserAccounts
CREATE INDEX index_useraccounts_username ON UserAccounts(username);
CREATE INDEX index_useraccounts_email ON UserAccounts(email);

-- Индексы для таблицы WatchHistory
CREATE INDEX index_watchhistory_user_id ON WatchHistory(user_id);
CREATE INDEX index_watchhistory_film_id ON
WatchHistory(film_or_serial_id);
CREATE INDEX index_watchhistory_date_watched ON
WatchHistory(date_watched);

CREATE OR REPLACE PROCEDURE FindFilmByTitle(IN film_title VARCHAR(255))
AS $$
DECLARE
    film_info RECORD;
    language_info RECORD;
    award_info RECORD;
    director_info RECORD;
    user_info RECORD;
    actor_info RECORD;
    genre_info RECORD;
BEGIN
    -- Select the movie based on the provided title

```

```

SELECT * INTO film_info FROM "Films" WHERE "Title" = film_title;

-- Check if a movie with the provided title exists
IF FOUND THEN

    -- Fetch language details based on LanguageID_FK from the Languages
table
    SELECT * INTO language_info FROM "Languages" WHERE "LanguageID" =
film_info."LanguageID_FK";

    SELECT * INTO award_info FROM "Awards" WHERE "AwardID" =
film_info."AwardID_FK";

    SELECT * INTO director_info FROM "Directors" WHERE "DirectorID" =
film_info."DirectorID_FK";

    RAISE NOTICE 'Film Found - MovieID: %, Title: %,
        Release Year: %, Duration: %,
        Plot Summary: %, Average rating: %,
        Language: %, Award: %,
        Director: %',
        film_info."FilmID", film_info."Title",
        film_info."ReleaseYear", film_info."Duration",
        film_info."PlotSummary", film_info."AverageRating",
        language_info."Language", award_info."Award",
        director_info."LastName";

-- Fetch all actors for the film based on FilmsActors
FOR actor_info IN

    SELECT "Actors"."LastName"

    FROM "Actors"

    INNER JOIN "FilmsActors" ON "Actors"."ActorID" = "FilmsAc-
tors"."ActorID_FK"

    WHERE "FilmsActors"."FilmID_FK" = film_info."FilmID"

LOOP

    RAISE NOTICE 'Actor: %', actor_info."LastName";

END LOOP;

-- Fetch all genres for the film based on FilmsGenres
FOR genre_info IN

```

```

        SELECT "Genres"."Genre"
        FROM "Genres"
        INNER JOIN "FilmsGenres" ON "Genres"."GenreID" = "FilmsGen-
res"."GenreID_FK"
        WHERE "FilmsGenres"."FilmID_FK" = film_info."FilmID"
    LOOP
        RAISE NOTICE 'Genre: %', genre_info."Genre";
    END LOOP;
ELSE
    RAISE NOTICE 'Film Not Found';
END IF;
END;
$$ LANGUAGE plpgsql;

```

```
CALL FindFilmByTitle('Prestige');
```

```

CREATE OR REPLACE PROCEDURE FindUserByName(IN user_name VARCHAR(255))
AS $$
DECLARE
    user_info RECORD;
    role_info RECORD;
BEGIN
    -- Select the user based on the provided username
    SELECT * INTO user_info FROM "Users" WHERE "Username" = user_name;

    -- Check if a user with the provided username exists
    IF FOUND THEN
        -- Fetch role details based on RoleID_FK from the Roles table
        SELECT * INTO role_info FROM "Roles" WHERE "RoleID" =
user_info."RoleID_FK";

        RAISE NOTICE 'User Found - UserID: %, Role: %,
            Username: %, Email: %,
            Password: %, Registration Date: %',
            user_info."UserID", role_info."Role",
            user_info."Username", user_info."Email",

```

```

        user_info."Password", user_info."RegistrationDate";

    ELSE

        RAISE NOTICE 'User Not Found';

    END IF;

END;

$$ LANGUAGE plpgsql;

CALL FindUserByName('DrVaroZ');

CREATE OR REPLACE PROCEDURE create_user(
    IN roleid_fk INT,
    IN new_username VARCHAR,
    IN new_email VARCHAR,
    IN new_password VARCHAR
)
AS $$
BEGIN
    INSERT INTO "Users" ("RoleID_FK", "Username", "Email", "Password", "RegistrationDate")
    VALUES (roleid_fk, new_username, new_email, new_password, CURRENT_DATE);
END;

$$ LANGUAGE plpgsql;

CALL create_user(1, 'EgorSavonchik', 'egorsavonchik@gmail.com', '123123123')

CREATE OR REPLACE PROCEDURE get_user_by_id(
    IN user_id INT
)
AS $$
DECLARE
    user_record "Users"%ROWTYPE;
    role_name VARCHAR;
BEGIN
    SELECT * INTO user_record FROM "Users" WHERE "UserID" = user_id;

```

```

-- Fetch the role name based on RoleID_FK from the Roles table
SELECT "Role" INTO role_name FROM "Roles" WHERE "RoleID" = user_rec-
ord."RoleID_FK";

-- Print or do something with the user details including the role name
RAISE NOTICE 'User ID: %, Role: %,
              Username: %, Email: %,
              Password: %, RegistrationDate: %',
              user_record."UserID", role_name,
              user_record."Username", user_record."Email",
              user_record."Password", user_record."RegistrationDate";

END;

$$ LANGUAGE plpgsql;

CALL get_user_by_id(3);

CREATE OR REPLACE PROCEDURE get_user_by_name(
    IN user_name VARCHAR
)
AS $$
DECLARE
    user_record "Users"%ROWTYPE;
    role_name VARCHAR;
BEGIN
    SELECT * INTO user_record FROM "Users" WHERE "Username" = user_name;

    -- Fetch the role name based on RoleID_FK from the Roles table
    SELECT "Role" INTO role_name FROM "Roles" WHERE "RoleID" = user_rec-
ord."RoleID_FK";

    -- Print or do something with the user details including the role name
    RAISE NOTICE 'User ID: %, Role: %,
                  Username: %, Email: %,
                  Password: %, RegistrationDate: %',
                  user_record."UserID", role_name,
                  user_record."Username", user_record."Email",

```



```

        user_record."Password", user_record."RegistrationDate";

END;

$$ LANGUAGE plpgsql;

CALL get_user_by_name('DrVaroZ');

CREATE OR REPLACE PROCEDURE update_user(
    IN user_id INT,
    IN updated_username VARCHAR,
    IN updated_email VARCHAR,
    IN updated_password VARCHAR
)
AS $$
BEGIN
    UPDATE "Users"
    SET "Username" = updated_username,
        "Email" = updated_email,
        "Password" = updated_password
    WHERE "UserID" = user_id;
END;

$$ LANGUAGE plpgsql;

CALL update_user(1, 'DrVaroZ', 'drvaroz1@gmail.com', '12345678')

CREATE OR REPLACE PROCEDURE delete_user(
    IN user_name VARCHAR
)
AS $$
BEGIN
    DELETE FROM "Users" WHERE "Username" = user_name;
END;

$$ LANGUAGE plpgsql;

CALL delete_user('xxx')

```

```

CREATE OR REPLACE PROCEDURE create_review(
    IN username VARCHAR,
    IN film_title VARCHAR,
    IN rating INT,
    IN review_text TEXT
)
AS $$
DECLARE
    user_id INT;
    film_id INT;
BEGIN
    -- Get UserID based on the provided username
    SELECT "UserID" INTO user_id FROM "Users" WHERE "Username" = username;

    -- Get FilmID based on the provided film_title
    SELECT "FilmID" INTO film_id FROM "Films" WHERE "Title" = film_title;

    -- Check if the user and film exist
    IF user_id IS NOT NULL AND film_id IS NOT NULL THEN
        -- Insert the review
        INSERT INTO "Reviews" ("UserID_FK", "FilmID_FK", "Rating", "Review-
Text")
        VALUES (user_id, film_id, rating, review_text);
    ELSE
        RAISE EXCEPTION 'User or film not found';
    END IF;
END;
$$ LANGUAGE plpgsql;

CALL create_review('Eugene009', 'Prestige', 10, 'I am impressed by this
film')

```

```

CREATE OR REPLACE FUNCTION update_history_insert_review()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO "Histories" ("FilmID_FK", "DateWatched", "UserID_FK")

```

```

VALUES (NEW."FilmID_FK", CURRENT_DATE, NEW."UserID_FK");

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION update_film_average_rating()
RETURNS TRIGGER AS $$
BEGIN
    -- Update average rating in Films table when a new review is inserted,
    updated, or deleted
    UPDATE "Films"
    SET "AverageRating" = (
        SELECT COALESCE(AVG("Rating"), 0) FROM "Reviews" WHERE "FilmID_FK" =
NEW."FilmID_FK"
    )
    WHERE "FilmID" = NEW."FilmID_FK";

    RETURN NEW;
END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER review_created_trigger
AFTER INSERT ON "Reviews"
FOR EACH ROW
EXECUTE FUNCTION update_history_insert_review();

CREATE TRIGGER update_film_average_rating_trigger
AFTER INSERT OR UPDATE OR DELETE ON "Reviews"
FOR EACH ROW
EXECUTE FUNCTION update_film_average_rating();

```

ПРИЛОЖЕНИЕ Б
(обязательное)
Схема базы данных