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» |
| --- | --- |
|  | И.О. Заведующего кафедрой  информатики  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_С. И. Сиротко |
|  | \_\_\_.\_\_\_\_.2025 |

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

на тему

**ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА БАЗЫ-ДАННЫХ ДЛЯ КИНОТЕАТРА**

БГУИР КП 1-40 04 01 ПЗ 002

Студент С. С. Бекарев

Руководитель А. В. Давыдчик

Нормоконтролер А. В. Давыдчик

Минск 2025

**СОДЕРЖАНИЕ**

[1 Анализ предметной области 5](#_heading=h.6t5y2nqw5sil)

[1.1 Обзор онлайн-кинотеатра Кинопоиск HD 5](#_heading=h.yey665cy2orr)

[1.2 Обзор онлайн ресурса для покупки билетов Bycard.by 7](#_heading=h.2aqv26o05i6e)

[1.3 Обзор онлайн ресурса для бронирования билетов Relax.by 8](#_heading=h.k5ippfip6e1d)

[2 Формирование функциональных требований 11](#_heading=h.fik6hd38h53h)

[2.1 Функциональные требования к разрабатываемой базе данных 11](#_heading=h.sx5w6p3rjplt)

[2.2 Анализ существующих подходов к разработке баз данных 11](#_heading=h.83wkjv41ic82)

[2.3 Выбор базы данных для разработки предметной области 20](#_heading=h.ymx5v23lo3u4)

[2.4 Выбор архитектуры взаимодействия 22](#_heading=h.9f3clct56z55)

[3 Проектирование базы данных 23](#_heading=h.m1vtfwvfprf2)

[3.1 Инфологическая модель 23](#_heading=h.s01doqxwdq1m)

[3.2 Разработка даталогической модели базы данных 29](#_heading=h.lc1fn8p5ie5v)

[4 Разработка базы данных 31](#_heading=h.vdjcum3rtud9)

[4.1 Создание исходных таблиц, индексов и ограничений 31](#_heading=h.w52ittqg3wz7)

[4.2 Создание хранимых процедур 31](#_heading=h.pzjlxmyzgaxl)

[5 Тестирование базы данных 35](#_heading=h.vdci1a6rxe9)

[5.1 Руководство по развертыванию системы 35](#_heading=h.x0vvrs4ggoy)

[5.2 Руководство пользователя 35](#_heading=h.yj8lrbqre4yf)

[Заключение 41](#_heading=h.wm3qvzvmuknt)

[Список литературных источников 42](#_heading=h.m59ha7wwhvgs)

**ВВЕДЕНИЕ**

Киноискусство – это одна из самых массовых и влиятельных форм культуры, которая затрагивает все социальные слои населения. Кино способно не только развлекать, но и образовывать, вдохновлять, критиковать, объединять людей. Кино отражает реальность, создает фантазию, предлагает альтернативу. Кино – это искусство, которое требует внимания, уважения и поддержки.

Традиционным местом просмотра кино являются физические кинотеатры, которые предлагают зрителям атмосферу, качество звука и изображения, а также социальное взаимодействие с другими любителями кино. Однако, с развитием интернет-технологий и компьютерных устройств, появилась возможность смотреть кино онлайн, без необходимости посещать кинотеатры. Онлайн-кинотеатры предоставляют пользователям доступ к широкому ассортименту контента, который можно смотреть в любое время и в любом месте, при этом экономя время и деньги. Однако, несмотря на все преимущества онлайн-кинотеатров, многие люди по-прежнему предпочитают посещать физические кинотеатры, чтобы насладиться уникальным опытом просмотра кино на большом экране. Физические кинотеатры также имеют свои особенности и проблемы, такие как расписание сеансов, цены на билеты, доступность мест, качество обслуживания, конкуренция с другими развлекательными учреждениями и т.д.

В связи с этим, появилась идея создания онлайн-кинотеатра с возможностью бронирования в физических. Такой онлайн-кинотеатр позволил бы пользователям не только выбирать и смотреть контент онлайн, но и бронировать места в ближайших кинотеатрах, которые показывают тот же контент.

Целью данной курсовой работы является исследование потребностей и предпочтений пользователей, связанных с онлайн-кинотеатром с возможностью бронирования в физических, а также разработка концепции и прототипа такого онлайн-кинотеатра. Для достижения этой цели были поставлены следующие задачи:

Подведем основные цели на данный курсовой проект:

– проанализировать предметную область и существующие аналоги;

– описать сущности проектируемой базы данных и их связи;

– в необходимых случаях использовать индексы;

– создать необходимые триггеры и хранимые процедуры для работы с существующей базой данных.

# 1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Как таковых систем, имеющих в своем арсенале одновременно возможность просмотра кинофильмов и бронирования билетов в физических кинотеатрах в ходе анализа выявлено не было, в таком случае была произведена декомпозиция составных частей рассматриваемой предметной области результатом которой является обзор таких сервисов как:

– сервис приобретения билетов на мероприятия Bycard.by;

– сервис бронирования билетов Relax.by;

– онлайн-кинотеатр Кинопоиск HD.

## Обзор онлайн-кинотеатра Кинопоиск HD

Меню с выбранным фильмом в данном кинотеатре выглядит следующим образом на рисунке 1.1:

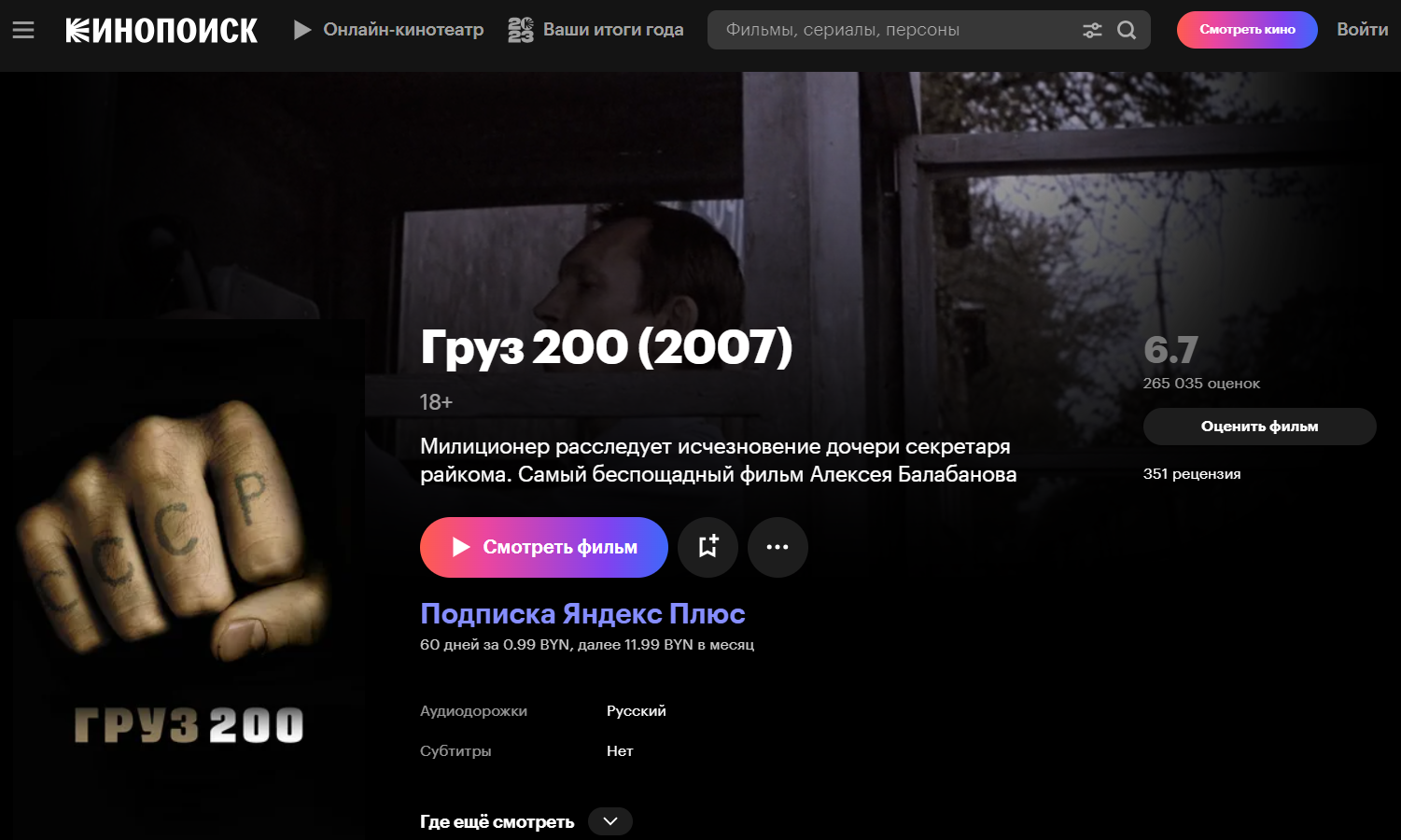


Рисунок 1.1 – Меню с информацией о выбранном фильме

Из данного рисунка видно, что отличительной чертой от других сервисов, является наличие информации о среднем рейтинге фильма, о количестве оценок и рецензий. Разница между оценкой и рецензией заключается в наличии текстового мнения о фильме, а в свою очередь оценки могут в себе текст не содержать.

Так как информация о структуре базы данных любого из рассматриваемых ресурсов является собственностью компаний, разрабатывающих данные ресурсы, попробуем предположить, что использует Кинопоиск HD в качестве базы данных.

Возможно, что Кинопоиск HD использует нереляционную базу данных MongoDB, которая хранит данные в виде документов, имеющих ключ-значение структуру и могущих содержать разные типы данных. MongoDB подходит для работы с данными, которые имеют неоднородную и изменчивую структуру, такие как информация о фильмах, сериалах, актерах, режиссерах и т.д. MongoDB также обладает высокой масштабируемостью, производительностью и гибкостью, что важно для онлайн-кинотеатра, который должен обслуживать большое количество пользователей и предоставлять им разнообразный и актуальный контент.

Основными сущностями базы данных в таком случае могут быть:

Фильм – это сущность, представляющая собой кинематографическое произведение, доступное для просмотра на ресурсе. Атрибуты фильма могут быть название, жанр, страна, год, режиссер, актеры, описание, рейтинг, длительность, формат и т.д. Ключом фильма может быть его уникальный идентификатор или название.

Сериал – это сущность, представляющая эпизодическое произведение кинематографа, где количество эпизодов ограничено фантазией режиссера и как правило доступное для просмотра на ресурсе. Атрибуты сериала могут быть название, жанр, страна, год, режиссер, актеры, описание, рейтинг, длительность, формат, сезоны, эпизоды и т.д. Ключом сериала может быть его уникальный идентификатор или название.

Рецензия – это сущность, которая представляет текстовый отзыв на просмотренное произведение, которое скорее всего помимо текстового описания требует и числовую оценку. В таком случае данная сущность должна хранить в себе текст рецензии, уникальный идентификатор пользователя, который это написал и числовой рейтинг, который он поставил данному фильму. Ключом будет уникальный идентификатор рецензии.

Пользователь – это сущность, которая представляет собой человека, который зарегистрирован на ресурсе и использует его услуги. Атрибуты пользователя могут быть логин, пароль, имя, фамилия, электронная почта, телефон, дата рождения, пол, роль, подписка, история, избранное, настройки и т.д. Ключом пользователя может быть его уникальный идентификатор или логин.

Просмотр – это сущность, которая дает возможность анализировать поведение пользователя во время просмотра, а именно длительность просмотра в сравнении с длительностью фильма, следить за тем, когда необходимо вставить рекламную интеграцию и успешной её применения в заданный момент времени.

## 

## Обзор онлайн ресурса для покупки билетов Bycard.by

Bycard.by является белорусским сервисом для бронирования билетов на мероприятия, которые в себя включают и бронирование билетов на киносеансы в кинотеатрах областных центров разной величины Республики Беларусь. На рисунке 1.2 представлен графический интерфейс приложения:

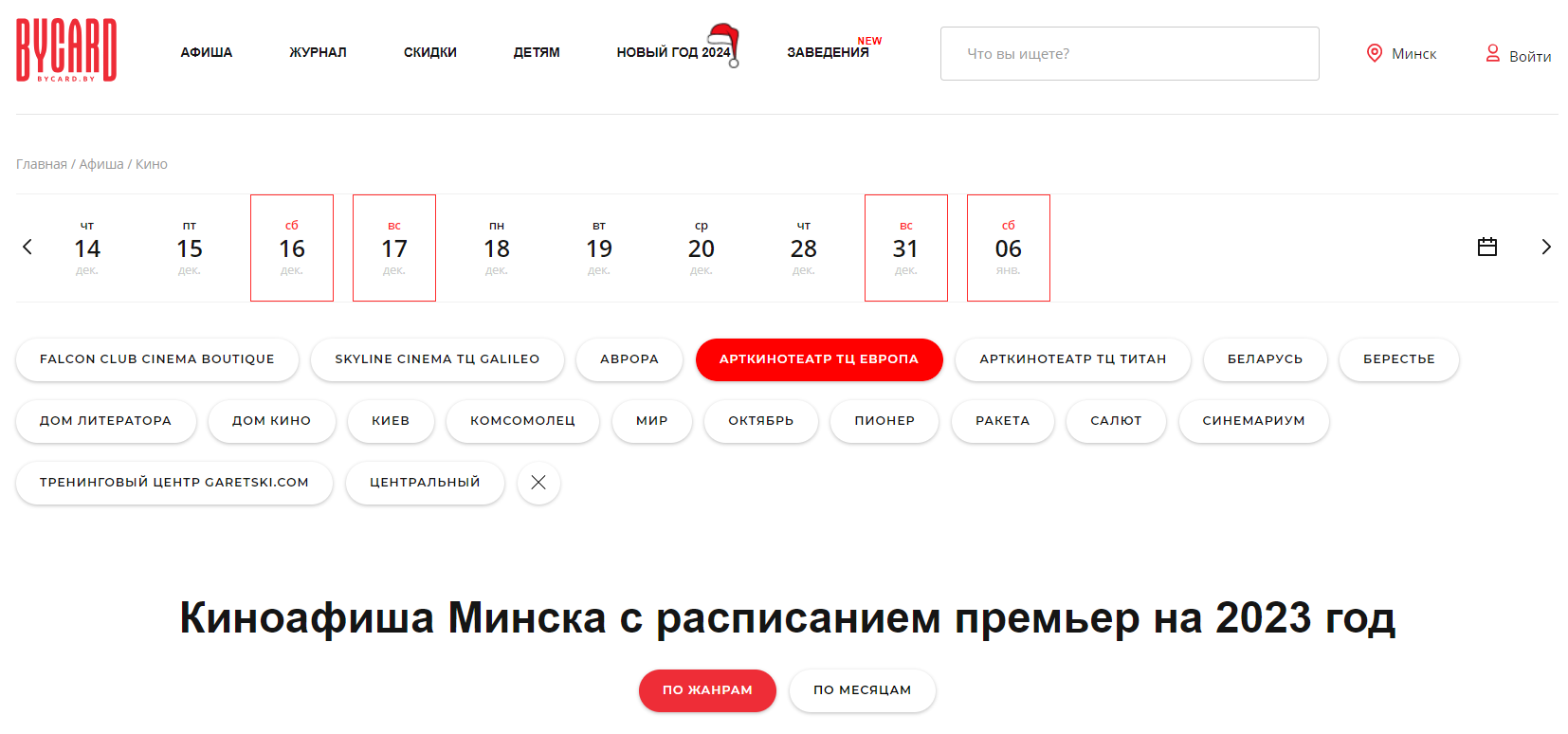


Рисунок 1.2 – Графический интерфейс Bycard.by

Из рисунка видно, что у ресурса имеются даты, на которые можно бронировать билеты на мероприятия, сами места проведения киносеансов, расположенные регионально, в данном случае регионом выступает город Минск и на рисунке представлены большинство известных кинотеатров города. Из этого можно сделать вывод, что данный ресурс в той или иной форме имеет сущности мероприятий, который содержат в себе дату его проведения, а также место проведения и другую необходимую информацию.

В разрабатываемой базе данных предметной областью является кино, со всеми вытекающими из неё информационными данными, в таком случае сделаем вывод, что как минимум необходимо иметь сущность названия фильма, стоимость билетов к нему.

При переходе на конкретную страницу с фильмом и открытии описания фильма, получим представление, изображенное на рисунке 1.2:

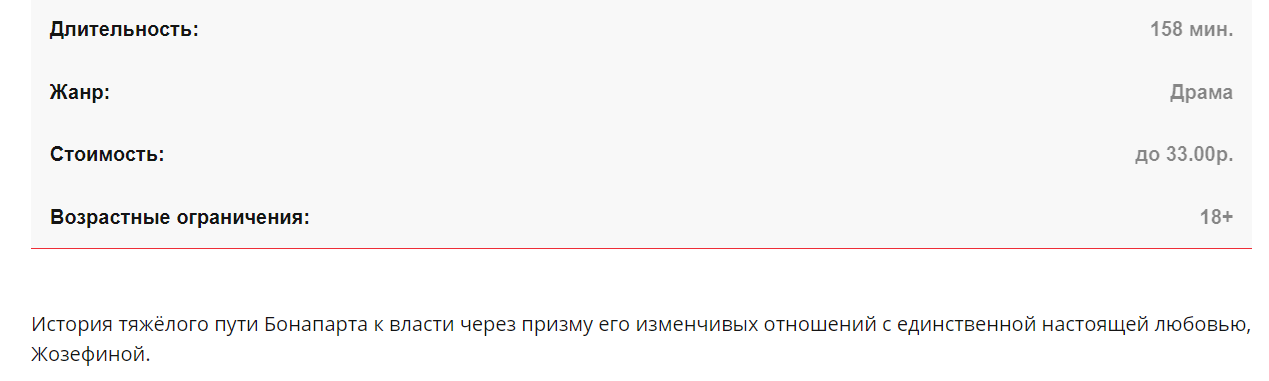


Рисунок 1.2 – Описание выбранного фильма

Исходя из изображенной на рисунке 1.2 информации предположим, что в контексте разрабатываемой базы данных будет необходимо добавить информацию о длительности фильма, о его жанре, а также возрастных ограничениях.

В рассматриваемом ресурсе, который является сайтом для покупки билетов на мероприятия, есть специальная вкладка, где пользователи могут оставлять свои отзывы о просмотренных фильмах. Отзывы могут быть полезны для других пользователей, которые хотят выбрать фильм для просмотра, а также для анализа предпочтений и впечатлений аудитории. Однако в данном случае ни один из предложенных фильмов не получил ни одного отзыва от пользователей, поэтому невозможно проанализировать, как отзывы влияют на популярность и оценку фильмов на этом ресурсе.

## Обзор онлайн ресурса для бронирования билетов Relax.by

Информационный ресурс Relax.by по своим размерам является большим агрегатором мероприятий, нежели его предыдущий конкурент Bycard.by. Так как он вдобавок может предоставлять пользователям туры и экскурсии в городе Минске, но видимо в сложившейся обстановке это не является преимуществом, и вкладка с турами, представленная на рисунке 1.3 представлена следующим образом:

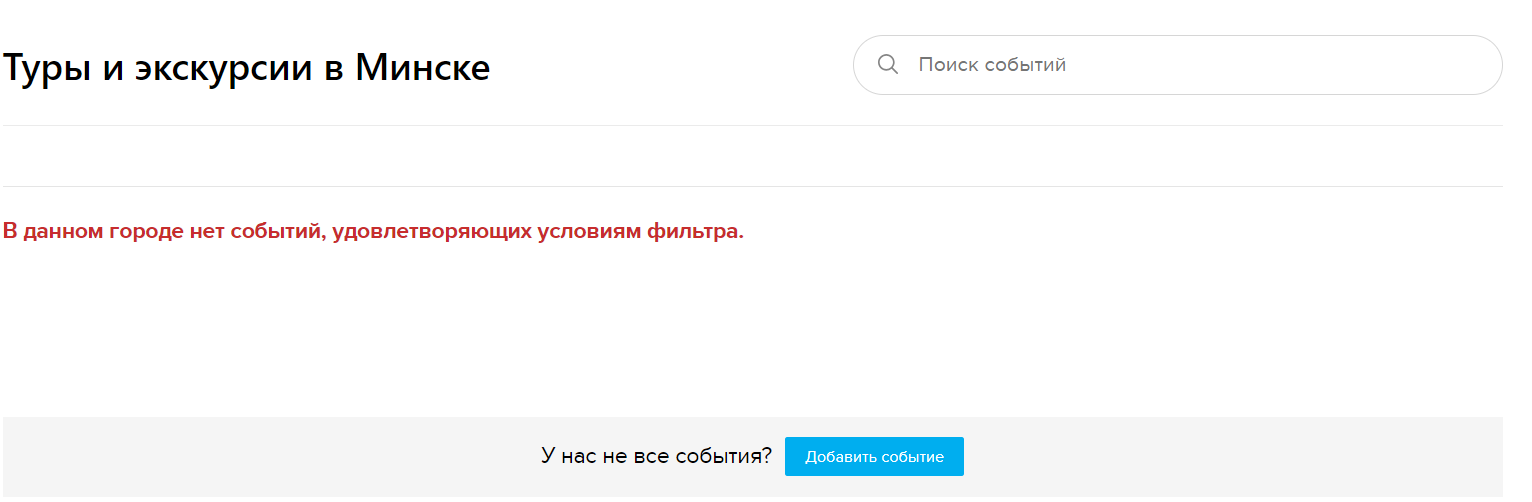


Рисунок 1.3 – Вкладка туров и экскурсий ресурса Relax.by

Касательно мероприятий с кинофильмами, информационное наполнение рассматриваемого ресурса схоже с предыдущим аналогом, страница с текущими премьерами и фильмами в кинотеатрах представлена на рисунке 1.4:

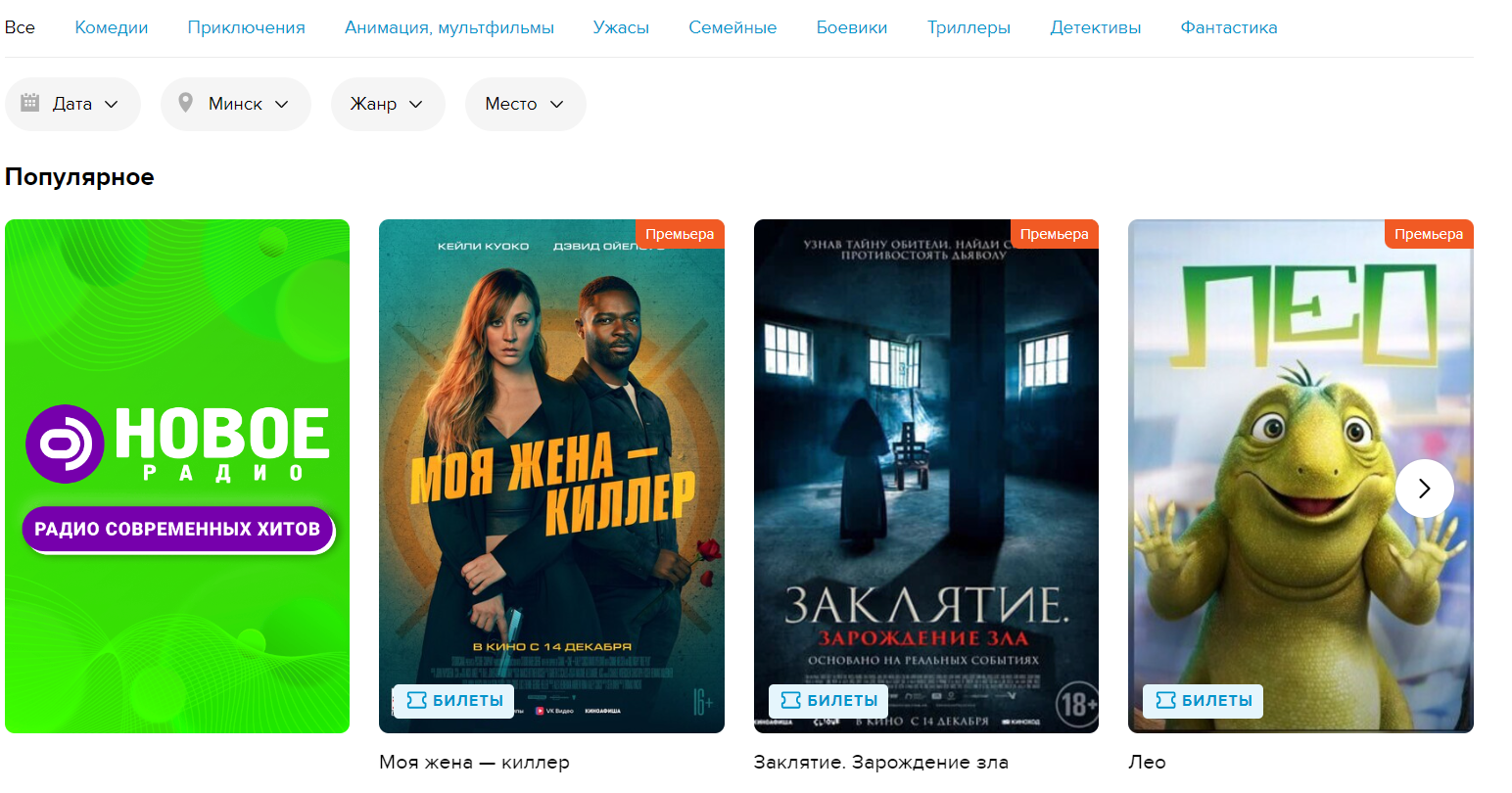


Рисунок 1.4 – Страница с информацией о фильмах

Условно данную страницу можно разделить на 3 информационных блока:

1 Блок фильтрации жанра фильма.

2 Блок фильтрации фильмов по дате, по месту проведения киносеанса в масштабе городов и областных центров, фильтрация по жанру фильмов и непосредственно фильтрация по месту проведения в масштабе кинотеатра.

3 Блок карточек с информацией о фильмах.

Исходя из рассмотренных информационных блоков сделаем вывод, что дополнительной сущностью или полем уже существующей сущности необходимо сохранять информацию, отвечающую за жанр фильма.

В данном ресурсе также имеется блок для того, чтобы оставить отзыв о фильме, но существующих отзывов в текущих фильмах не было найдено, поэтому вопрос о работе механизма рейтинга фильма и отзывов остается открытым.

Но также нельзя не заметить обильное количество рекламы, размещением которой, судя по всему Relax.by зарабатывает дополнительные денежные средства.

Таким образом, при разработке базы данных размещение рекламы на сайте для онлайн-кинотеатра может позволить покрыть некоторую часть расходов на разработку продукта и протестировать разрабатываемые способы оплаты, таких как банковские карты, электронные кошельки, смс-сервисы и т.д., так как первоначально вопрос об окупаемости проекта за счет приобретения пользовательских подписок на данный сервис, которые будут давать доступ к расширенному каталогу контента и дополнительным функциям, отложен в долгий ящик, то есть не является приоритетным в данный момент.

# 2 ФОРМИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

## 2.1 Функциональные требования к разрабатываемой базе данных

На основе анализа существующих аналогов выдвинем функциональные требования, к разрабатываемой системе онлайн кинотеатра, в котором предусмотрена возможность бронирования билетов в физических кинотеатрах, а также реализованы сущности для добавления и сбора статистики по рекламным интеграциям [5]:

1 Возможность к добавлению и удалению различного рода информации, такой как: информация о пользователях, режиссерах и их фильмах, а также актерах.

2 Возможность оставить удалить или исправить пользовательский комментарий и поставить оценку фильму, с последующим пересчетом средней оценки по всем записям.

3 Возможность изменять данные по купленным билетам на киносеанс, в случае покупки или возврата пересчитать количество свободных билетов и на основе этого делать пересчет стоимости билетов.

4 Возможность проверять покупателей билетов на фильмы, с высоким ограничением по возрасту.

5 Возможность дать рекомендацию пользователю на основе фильмов, которые он посмотрел, с учетом продолжительности просмотра.

6 Возможность дать рекомендацию пользователю на основе фильмов, которые находятся у него в плейлистах.

7 Возможность получить фильмы по списку актеров.

8 Возможность получить фильмы по определенному жанру и стране.

## 2.2 Анализ существующих подходов к разработке баз данных

Всего в современном проектировании и разработке баз данных выделяют 2 основных вида: SQL-, NoSQL- базы данных. Но это ни в коем случае не говорит о том, что для первого вида используются SQL-запросы, а для второй нет, здесь скорее это относится к тому, что SQL базы данных используют реляционную теорию, а NoSQL – нет. Для более явного визуального представления рассмотрим рисунок 2.1, на котором изображены основные подвиды баз данных:



Рисунок 2.1 – Основные виды SQL и NoSQL баз данных

Теперь более явным образом рассмотрим основные отличия реляционных и нереляционных баз данных.

**2.2.1 Реляционные базы данных**

Реляционные базы данных — это базы данных, которые используют табличный способ хранения и обработки данных [6]. В реляционных базах данных данные организованы в таблицы, состоящие из строк и столбцов. Каждая строка представляет собой запись с уникальным идентификатором, называемым ключом. Каждый столбец имеет атрибут данных, который характеризует записи. Таблицы связаны между собой посредством ключей, которые обеспечивают целостность и согласованность данных. Пример такой базы данных представлен на рисунке 2.2

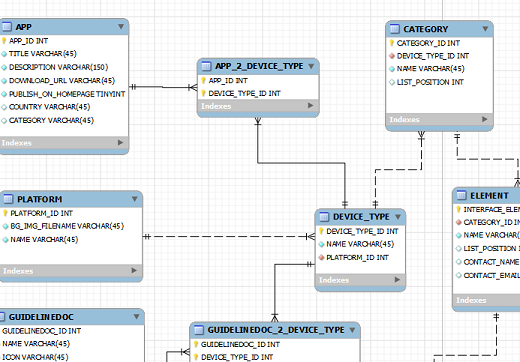


Рисунок 2.2 – Пример организации данных в реляционной теории

Для работы с реляционными базами данных используется стандартный язык SQL, который позволяет выполнять различные операции над данными, такие как выборка, вставка, обновление, удаление, объединение, агрегация и т.д. SQL также поддерживает транзакции, которые являются неделимыми блоками операций, обеспечивающими атомарность, согласованность, изолированность и устойчивость данных [7]. Примерную структуру команд и их разделение на группы представлены на рисунке 2.3

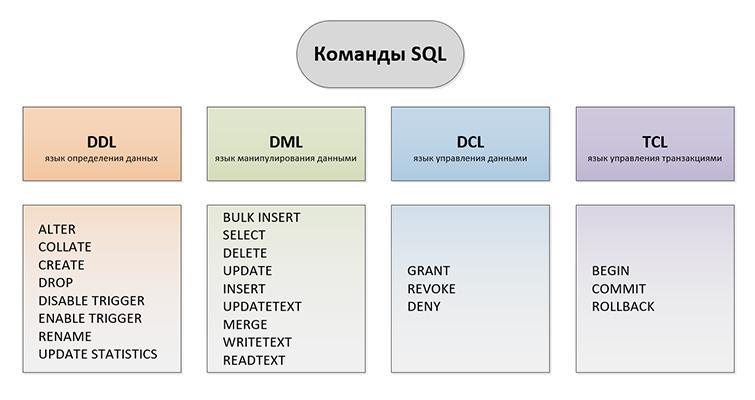


Рисунок 2.3 – Структура команд SQL

Логическое и физическое представление баз данных — это два разных способа описания и организации данных в базе данных. Логическое представление отражает, как данные понимаются и используются пользователями и приложениями. Физическое представление отражает, как данные хранятся и обрабатываются системой управления базой данных.

Логическое представление базы данных состоит из логических объектов, таких как таблицы, представления, индексы и т.д. Эти объекты определяют структуру, свойства и отношения данных, а также правила и ограничения, которые обеспечивают целостность и согласованность данных. Логическое представление базы данных может быть описано с помощью концептуальной схемы, которая показывает все элементы данных и их связи в виде графического диаграммы. Логическое представление базы данных может быть разделено на несколько уровней абстракции, таких как внешний, логический и внутренний.

Внешний уровень представляет данные в формате, понятном конкретному пользователю или приложению. Логический уровень представляет данные в формате, понятном всем пользователям и приложениям. Внутренний уровень представляет данные в формате, понятном системе управления базой данных. Пример логического представления базы данных представлен на рисунке 2.4:

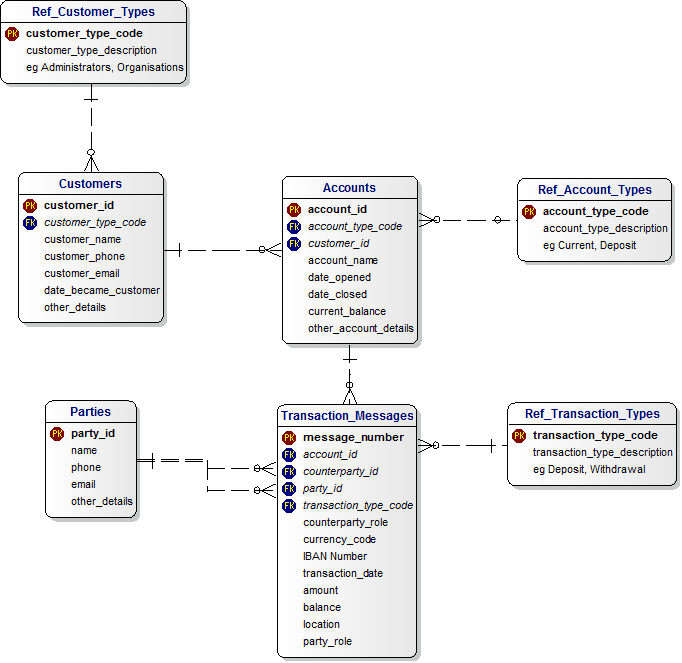


Рисунок 2.4 – Логическое представление базы данных

Физическое представление базы данных состоит из физических объектов, таких как файлы, блоки, страницы, сегменты и т.д. Эти объекты определяют способы хранения, доступа и обработки данных на физическом носителе, таком как диск, память или сеть.

Физическое представление базы данных может быть описано с помощью физической схемы, которая показывает расположение и размер физических объектов, а также параметры и настройки, которые влияют на производительность и эффективность базы данных.

Физическое представление базы данных может быть оптимизировано с помощью различных методов и техник, таких как сжатие, шардирование, кэширование, индексирование и т.д. Схема организации базы данных с физическим уровнем представлена на рисунке 2.4:

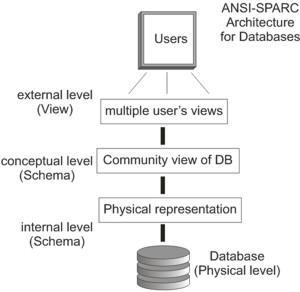


Рисунок 2.4 – Схема организации базы данных с физическим уровнем

Разделение логического и физического представления базы данных позволяет достичь логической и физической независимости данных. Логическая независимость означает, что изменения в логической структуре данных не влияют на прикладные программы и пользователей. Физическая независимость означает, что изменения в физической организации данных не влияют на логическую структуру данных и прикладные программы.

**2.2.2 Нормализация реляционных баз данных**

Избыточность данных в отношениях — это одна из главных проблем, с которой могут столкнуться разработчики при проектировании баз данных. Эту проблему можно представить, как ситуацию, когда в различных отношениях повторяются одни и те же данные или когда поля не соответствуют сущностям, которые они должны характеризовать (обычно такие проблемы связаны между собой).

Нормализация, по своей сути заключается в создании формальных правил для реализации некоторых основных принципов проектирования баз данных: один факт в одном месте и так далее.

Для того, чтобы привести базу данных к определенной нормальной форме, используется процесс нормализации. Его можно описать как пошаговое преобразование заданного набора переменных отношения к более предпочтительной форме. Стоит отметить, что этот процесс можно обратить, то есть всегда можно использовать его результат таким примером может служить набор переменных отношения, находящихся в 3НФ, возвращенный в переменные отношения 2НФ.

Возможность выполнения обратного преобразования — это очень важная особенность, потому что это означает, что в процессе нормализации информация не теряется. Сейчас существуют шесть нумерованных нормальных форм, нормальная форма Бойса-Кодда и доменно-ключевая нормальная форма. Их нужно рассматривать по порядку, потому что каждая следующая нормальная форма требует, чтобы база данных была в предыдущей нормальной форме. В целом, для правильного логического разложения базы данных и её использования в рабочих проектах достаточно нормализовать базу данных до 3НФ, а остальные формы носят скорее научный характер.

Первая нормальная форма (1НФ). Первая нормальная форма — это базовая нормальная форма для всех баз данных, основанных на реляционной модели данных. Ее суть можно описать следующим образом.

Переменная отношения находится в 1НФ тогда и только тогда, когда в любом допустимом значении этой переменной отношения каждый кортеж имеет только одно значение для каждого атрибута. Другими словами, здесь говорится, что любое отношение по умолчанию находится в первой нормальной форме, что верно. Но также верно, что отношение, которое находится только в первой нормальной форме (без перехода ко второй и третьей) плохо подходит для внедрения в какую-либо информационную систему, потому что оно не решает проблему избыточности данных и может приводить к так называемым аномалиям обновления.

Аномалии обновления — это общий термин для группы проблем, которые появляются при выполнении операций обновления данных (INSERT, UPDATE, DELETE) в отношениях, содержащих избыточные данные. Такие проблемы часто приводят к непредсказуемому поведению системы после обновления некоторых данных, так как для обновления одного аспекта описываемой сущности нужно выполнить операцию обновления в нескольких независимых друг от друга местах базы данных.

Вторая нормальная форма (2НФ). Вторая нормальная форма является продолжением первой нормальной формы, где добавляется требование, чтобы элементы кортежа зависели от всего первичного ключа целиком.

Вот ее полное определение: Переменная отношения находится во второй нормальной форме тогда и только тогда, когда она находится в первой нормальной форме и каждый неключевой атрибут несводимо зависит от ее первичного ключа. Нормализация до второй формы направлена на устранение ситуаций, когда отношение имеет составной первичный ключ, а значение некоторого неключевого поля зависит от одной части первичного ключа, но не зависит от другой.

Сам процесс нормализации заключается в замене текущей ненормализованной переменной отношений на несколько эквивалентных проекций, которые в совокупности дают ту же переменную отношений (нужно помнить, что нормализация должна проходить без потери данных).

Тогда в ситуации, когда мы имеем переменную отношений A с полями B, C, D, E, где B и C образуют составной первичный ключ, а D функционально зависит от B, но не зависит от C - правильно будет разбить A на две проекции: A1 и A2, где A1 содержит поля B и D, а A2 - B, C, E. При этом поле B в проекции A2 должно быть не только частью первичного ключа, но и внешним ключом, ссылкой на таблицу A1.

Этот способ нормализации позволяет избежать непосредственной зависимости поля D от поля C, при этом исходную переменную отношений можно легко восстановить, то есть данные не потеряны. Третья нормальная форма (3НФ). Третья нормальная форма в классическом виде является развитием второй нормальной формы, но добавляет требование устранения транзитивных зависимостей. Ее можно определить так:

Переменная отношения находится в третьей нормальной форме тогда и только тогда, когда она находится во второй нормальной форме и никакой неключевой атрибут не является транзитивно зависимым от ее первичного ключа.

Цель нормализации до третьей нормальной формы — это устранить ситуации, когда одно неключевое поле определяется другим неключевым полем. Метод нормализации похож на метод нормализации до второй нормальной формы. Для этого также нужно разделить переменную отношения на несколько проекций. Например, есть переменная отношения A с полями B, C и D, где поле B — это первичный ключ.

Также есть функциональная зависимость поля D от поля C. Один из возможных способов решения этой проблемы — это разделить переменную отношения A на проекции A1 и A2, где проекция A1 включает поля C и D, при этом поле C — это первичный ключ; а проекция A2 включает поля B и C, где B — это первичный ключ, а C — внешний ключ, ссылкой на проекцию A1.

**2.2.3 Нереляционные базы данных**

Документо-ориентированные базы данных хранят данные в виде документов, которые имеют свою структуру и могут содержать разные типы данных. Документы группируются в коллекции, которые могут иметь разные схемы. Для работы с документо-ориентированными базами данных используются специальные языки запросов, которые позволяют обращаться к данным по их атрибутам. Рассмотрим подробнее примеры документо-ориентированных хранилищ данных:

1 MongoDB — одна из самых популярных и мощных документо-ориентированных баз данных, которая поддерживает разные форматы документов, такие как JSON, BSON и XML. MongoDB обладает высокой производительностью, масштабируемостью и гибкостью, а также предоставляет различные функции и инструменты для работы с данными, такие как агрегация, индексация, шардирование, репликация и т.д.

2 Firebase — облачная платформа, которая предоставляет документо-ориентированную базу данных в реальном времени, называемую 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, аналитика, машинное обучение и т.д.

Далее рассмотрим базы данных ключ-значение. Базы данных ключ-значение хранят данные в виде пар ключ-значение, где ключ является уникальным идентификатором, а значение может быть любым типом данных. Базы данных ключ-значение обеспечивают быстрый доступ к данным по ключу, но не поддерживают сложные запросы и связи между данными. Обладают они следующими особенностями:

1 Легко масштабируются по горизонтали. Достигается это благодаря тому, что данные могут быть распределены по разным узлам или серверам без необходимости объединения таблиц или синхронизации схем.

2 Подходят для хранения и обработки неструктурированных или полуструктурированных данных. Такими данными как правило выступают текст, изображения или видео, которые могут иметь разные размеры или формат.

3 Они позволяют гибко изменять структуру и свойства данных, так как не требуют жесткой схемы или типизации данных.

4 Они обладают высокой производительностью и низкой задержкой, так как обрабатывают данные в оперативной памяти или на быстрых носителях.

Отличными примерами базы данных ключ-значения являются:

Redis — одна из самых популярных и мощных баз данных ключ-значение, которая хранит данные в оперативной памяти и поддерживает разные типы значений, такие как строки, списки, множества, хеши, битовые массивы и т.д. Redis также предоставляет различные функции и инструменты для работы с данными, такие как транзакции, репликация, шардирование, кэширование, публикация-подписка и т.д.

DynamoDB — облачная база данных ключ-значение, предоставляемая Amazon Web Services. DynamoDB хранит данные на твердотельных накопителях и обеспечивает высокую доступность, надежность и масштабируемость данных. DynamoDB поддерживает разные типы значений, такие как строки, числа, бинарные данные, списки и карты, а также позволяет выполнять условные запросы и обновления данных.

Графовые базы данных хранят данные в виде узлов и ребер, которые представляют собой сущности и связи между ними. Графовые базы данных подходят для моделирования сложных сетей и отношений, таких как социальные сети, рекомендательные системы, маршрутизация и т.д. Для работы с графовыми базами данных используются специальные языки запросов, которые позволяют искать пути и паттерны в графе. Примеры графовых баз данных: Neo4j, OrientDB, ArangoDB и др.

Они лучше отражают реальную структуру и семантику данных, которые часто имеют сложные и динамические взаимосвязи, такие как социальные сети, рекомендательные системы, биоинформатика и т.д.

Графовые базы данных — это базы данных, которые используют математическую теорию графов для отображения и обработки связей между данными. В графовых базах данных данные представлены в виде узлов и ребер, которые обозначают сущности и отношения между ними. Графовые базы данных имеют ряд преимуществ перед реляционными и другими видами нереляционных баз данных, таких как:

Они позволяют быстро и эффективно выполнять запросы, которые требуют обхода и анализа связей в графе, такие как поиск кратчайшего пути, обнаружение сообществ, выявление аномалий и т.д.

Они обеспечивают высокую гибкость и масштабируемость, так как не требуют жесткой схемы данных и позволяют 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 и базы данных знаний. Базы данных свойственных графов хранят данные в виде графов со свойствами, то есть узлы и ребра имеют атрибуты, которые описывают их характеристики.

Базы данных знаний хранят данные в виде графов онтологий, то есть узлы и ребра имеют семантические метки, которые определяют их типы и смысл. Примеры баз данных свойственных графов: Neo4j, ArangoDB, JanusGraphи др. Примеры баз данных знаний: AllegroGraph, Datomic, TerminusDB и др.

Для работы с графовыми базами данных используются специальные языки запросов, которые позволяют обращаться к данным по их узлам, ребрам и свойствам. Некоторые известные языки запросов для графовых баз данных: Cypher, Gremlin, SPARQL и др.

## 2.3 Выбор базы данных для разработки предметной области

Исходя из анализа подходов в предыдущем пункте, следует сделать вывод, что предметная область онлайн-кинотеатра будет реализована с помощью реляционных баз данных [8]. Потому что выбранная предметная область имеет четкие атрибуты, сущности и связи, которые достаточно просто можно представить в виде таблиц и ключей в реляционной теории разработке баз данных.

Но все это будет корректно работать только при условии обеспечения целостности и согласованности данных, чтобы избежать ошибок и конфликтов при бронировании билетов, просмотре фильмов и размещении рекламы.

Для проведения аналитики и создания рекомендательной системы на стороне базы данных необходимо будет выполнять сложные SQL-запросы, с чем отлично справляются базы данных реляционной теории.

В качестве базы данных для разработки была выбрана PostgreSQL по ряду причин. PostgreSQL имеет много достоинств, которые делают ее желанным решением для разнообразных приложений и компаний. PostgreSQL — это СУБД с высокой производительностью, которая может работать с большими данными и обслуживать много пользователей одновременно [9].

Это достигается благодаря нескольким причинам. В PostgreSQL применяется усовершенствованный механизм запросов, который помогает эффективно работать с данными в базе. Поддерживает параллельность, что дает возможность выполнять несколько запросов одновременно. PostgreSQL может масштабироваться вертикально, добавляя дополнительные ресурсы, или горизонтально, распределяя нагрузку между несколькими серверами. PostgreSQL может применяться для разных приложений, в том числе:

1 Большие корпоративные приложения. PostgreSQL подходит для больших компаний, которым нужна высокая производительность, масштабируемость и безопасность. Она может применяться для таких приложений, как системы управления ресурсами предприятия (ERP), системы управления цепочками поставок (SCM) и системы управления клиентами (CRM).

2 Веб-приложения. Популярным решением при разработке веб-приложений в качестве базы данных использовать PostgreSQL. Она поддерживает функции, необходимые для разработки масштабируемых и надежных веб-приложений.

3 Интернет-магазины. Также отличным вариантом для использования PostgreSQL является создания и управление интернет-магазинами. Она поддерживает функции, необходимые для обработки транзакций, хранения товаров и управления клиентами.

4 Социальные сети. PostgreSQL может использоваться для создания и управления социальными сетями. Она поддерживает функции, необходимые для хранения данных пользователей, обработки сообщений и обеспечения безопасности.

PostgreSQL имеет много возможностей безопасности, которые помогают обезопасить данные от неправомерного доступа и использования. Существуют разные способы аутентификации, такие как пароли, сертификаты и двухэтапная аутентификация. PostgreSQL поддерживает разные схемы авторизации, такие как основанная на ролях и основанная на объектах. Такие схемы дают возможности администраторам базы данных управлять, какие пользователи имеют доступ к каким данным и функциям [10]. Эта база данных поддерживает разные способы шифрования, такие как шифрование данных на диске и шифрование данных в процессе передачи. Это помогает защитить данные от неправомерного доступа во время хранения и трансфера.

## 2.4  Выбор архитектуры взаимодействия

Для организации взаимодействия между клиентской и серверной частями системы выбрана архитектура RESTful API с использованием HTTP протокола и формата обмена данными JSON. Данное решение основывается на необходимости обеспечения гибкого, масштабируемого и безопасного взаимодействия между компонентами распределенной системы при сохранении простоты разработки и поддержки.

REST представляет собой архитектурный стиль проектирования сетевых приложений, основанный на принципе клиент-серверного взаимодействия без сохранения состояния между запросами. Каждый HTTP-запрос от клиента к серверу содержит всю необходимую информацию для его обработки, включая параметры операции и токен аутентификации, что позволяет серверу не хранить контекст сессий пользователей.

REST API использует стандартные HTTP-методы для выражения семантики операций над ресурсами: GET для получения данных без изменения состояния сервера, POST для создания новых ресурсов или выполнения операций с побочными эффектами, PUT для обновления существующих ресурсов путем полной замены их представления, DELETE для удаления ресурсов, PATCH для частичного обновления атрибутов ресурса. Такое использование семантики HTTP-методов делает API самодокументируемым и интуитивно понятным для разработчиков клиентских приложений.

Взаимодействие с базой данных PostgreSQL на серверной стороне организовано через прямое подключение к базе данных, получение данных посредством запросов на языке SQL.

Выбранная архитектура взаимодействия обеспечивает разделение ответственности между клиентской частью, отвечающей за представление данных и пользовательский интерфейс, и серверной частью, реализующей бизнес-логику и доступ к данным. Такое разделение упрощает параллельную разработку компонентов различными командами, облегчает тестирование каждого уровня изолированно, позволяет независимое масштабирование клиентской и серверной частей в соответствии с нагрузкой. Стандартизация на HTTP протоколе и JSON формате обеспечивает совместимость с широким спектром инструментов разработки, тестирования, мониторинга и упрощает интеграцию системы с внешними сервисами при необходимости.

# 3 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

## 3.1 Инфологическая модель

Стартовой точкой в проектировании базы данных является создание инфологической модели [11]. Сама инфологическая модель представляет собой только описание сущностей и связанный с нею атрибутов, без конкретизации типов данных и подробностей реализации на серверной части базы данных.

Графический пример такой схемы представлен на рисунке 3.1:



Рисунок 3.1 – Графический пример инфологической модели

Всего описываемая база данных включает в себя 30 сущностей. Постепенно опишем каждую из них:

**3.1.1** Таблица User

Таблица User хранит информацию о пользователях системы. Поле id является первичным ключом. Поле first\_name содержит имя пользователя. Поле last\_name содержит фамилию пользователя. Поле email хранит адрес электронной почты для авторизации и связи. Поле password содержит зашифрованный пароль для входа в систему. Поле contact\_num хранит контактный телефон пользователя. Поле role\_id является внешним ключом, ссылающимся на таблицу User\_roles и определяет роль пользователя в системе. Поле age содержит возраст пользователя.

**3.1.2** Таблица User\_roles

Таблица User\_roles содержит справочник ролей пользователей в системе. Поле id является первичным ключом. Поле name содержит название роли (например, администратор, пользователь, модератор).

**3.1.3** Таблица Movies

Таблица Movies хранит информацию о фильмах в системе. Поле id является первичным ключом. Поле title содержит название фильма. Поле description содержит развернутое описание сюжета и особенностей фильма. Поле release\_date хранит дату выхода фильма в прокат. Поле rating содержит средний рейтинг фильма. Поле duration хранит продолжительность фильма в минутах. Поле genre\_id является внешним ключом к таблице Genres и определяет жанр фильма. Поле country\_id является внешним ключом к таблице Countries и указывает страну производства. Поле age\_restriction\_id является внешним ключом к таблице Age\_restrictions и определяет возрастное ограничение для просмотра.

**3.1.4** Таблица Movie\_director

Таблица Movie\_director реализует связь многие-ко-многим между фильмами и режиссерами. Поле id является первичным ключом. Поле movie\_id является внешним ключом к таблице Movies и указывает на конкретный фильм. Поле director\_id является внешним ключом к таблице Directors и указывает на режиссера фильма.

**3.1.5** Таблица Genres

Таблица Genres содержит справочник жанров кино. Поле id является первичным ключом. Поле name содержит название жанра (например, комедия, драма, боевик).

**3.1.6** Таблица Directors

Таблица Directors хранит информацию о режиссерах. Поле id является первичным ключом. Поле first\_name содержит имя режиссера. Поле last\_name содержит фамилию режиссера.

**3.1.7** Таблица Actors

Таблица Actors хранит информацию об актерах. Поле id является первичным ключом. Поле first\_name содержит имя актера. Поле last\_name содержит фамилию актера.

**3.1.8** Таблица Cinemas

Таблица Cinemas содержит информацию о кинотеатрах. Поле id является первичным ключом. Поле name содержит название кинотеатра. Поле address хранит физический адрес расположения кинотеатра.

**3.1.9** Таблица Halls

Таблица Halls хранит информацию о кинозалах в кинотеатрах. Поле id является первичным ключом. Поле cinema\_id является внешним ключом к таблице Cinemas и указывает на кинотеатр, в котором расположен зал. Поле hall\_num содержит номер зала для идентификации конкретного помещения. Поле capacity хранит максимальную вместимость кинозала в количестве мест.

**3.1.10** Таблица Seats

Таблица Seats содержит информацию о посадочных местах в кинозалах. Поле id является первичным ключом. Поле hall\_id является внешним ключом к таблице Halls и указывает на кинозал, в котором находится место. Поле row\_num содержит номер ряда. Поле seat\_num содержит номер сиденья в ряду. Поле seat\_type\_id является внешним ключом к таблице Seat\_types и определяет тип сиденья (стандартное, VIP, для инвалидов).

**3.1.11** Таблица Showtimes

Таблица Showtimes хранит расписание киносеансов. Поле id является первичным ключом. Поле movie\_id является внешним ключом к таблице Movies и указывает на демонстрируемый фильм. Поле hall\_id является внешним ключом к таблице Halls и определяет кинозал проведения сеанса. Поле startime содержит дату и время начала сеанса.

**3.1.12** Таблица Tickets

Таблица Tickets содержит информацию о билетах на киносеансы. Поле id является первичным ключом. Поле showtime\_id является внешним ключом к таблице Showtimes и указывает на конкретный сеанс. Поле seat\_id является внешним ключом к таблице Seats и определяет забронированное место. Поле price содержит стоимость билета в денежных единицах.

**3.1.13** Таблица Reservations

Таблица Reservations хранит информацию о бронированиях билетов пользователями. Поле id является первичным ключом. Поле reservation\_time содержит дату и время создания бронирования. Поле user\_id является внешним ключом к таблице Users и указывает на пользователя, создавшего бронь.

**3.1.14** Таблица Online\_Movies

Таблица Online\_Movies содержит информацию о фильмах, доступных для онлайн-просмотра. Поле id является первичным ключом. Поле movie\_id является внешним ключом к таблице Movies и указывает на конкретный фильм. Поле language\_id является внешним ключом к таблице Languages и определяет язык озвучки или субтитров. Поле url содержит уникальную ссылку для доступа к онлайн-версии фильма. Поле created\_at фиксирует дату и время создания записи. Поле updated\_at содержит дату и время последнего обновления записи.

**3.1.15** Таблица Views

Таблица Views ведет учет просмотров онлайн-фильмов пользователями. Поле id является первичным ключом. Поле online\_movie\_id является внешним ключом к таблице Online\_Movies и указывает на просматриваемый онлайн-фильм. Поле view\_time содержит дату и время начала просмотра. Поле view\_dur\_time хранит продолжительность просмотра в минутах или секундах. Поле user\_id является внешним ключом к таблице Users и указывает на пользователя, осуществившего просмотр.

## 

## 3.1.16 Таблица Reviews

Таблица Reviews хранит отзывы пользователей о фильмах. Поле id является первичным ключом. Поле movie\_id является внешним ключом к таблице Movies и указывает на оцениваемый фильм. Поле rating содержит числовую оценку фильма, выставленную пользователем. Поле comment содержит текстовое описание впечатлений и мнения о фильме. Поле created\_at фиксирует дату и время создания отзыва. Поле updated\_at содержит дату и время последнего редактирования отзыва.

## 

## 3.1.17 Таблица Payments

Таблица Payments хранит информацию о платежных операциях в системе. Поле id является первичным ключом. Поле reservation\_id является внешним ключом к таблице Reservations и связывает платеж с конкретным бронированием. Поле payment\_method\_id является внешним ключом к таблице Payment\_Methods и определяет способ оплаты. Поле created\_at фиксирует дату и время проведения платежа. Поле amount содержит сумму платежа в денежных единицах.

## 

## 3.1.18 Таблица Payment\_Methods

Таблица Payment\_Methods содержит справочник доступных способов оплаты. Поле id является первичным ключом. Поле name содержит название способа оплаты (например, банковская карта, электронный кошелек, наличные).

## 

## 3.1.19 Таблица Advertisements

Таблица Advertisements хранит информацию о рекламных материалах в системе. Поле id является первичным ключом. Поле title содержит название рекламного объявления. Поле description содержит развернутое описание рекламируемого продукта или услуги. Поле url хранит ссылку на внешний источник или целевую страницу рекламы. Поле type\_id является внешним ключом к таблице Adv\_Type и определяет тип рекламного материала.

## 

## 3.1.20 Таблица Adv\_Type

Таблица Adv\_Type содержит справочник типов рекламы. Поле id является первичным ключом. Поле name содержит название типа рекламы (например, баннер, видеореклама, всплывающее окно). Поле priority определяет приоритет показа данного типа рекламы относительно других.

## 

## 3.1.21 Таблица Ad\_Views

Таблица Ad\_Views ведет учет показов рекламы во время просмотров онлайн-фильмов. Поле id является первичным ключом. Поле ad\_id является внешним ключом к таблице Advertisements и указывает на показанное рекламное объявление. Поле view\_id является внешним ключом к таблице Views и связывает показ рекламы с конкретным просмотром фильма. Поле viewed\_at содержит дату и время показа рекламы пользователю.

## 

## 3.1.22 Таблица Countries

Таблица Countries содержит справочник стран. Поле id является первичным ключом. Поле name содержит название страны.

## 

## 3.1.23 Таблица Languages

Таблица Languages содержит справочник языков. Поле id является первичным ключом. Поле name содержит название языка.

## 

## 3.1.24 Таблица Age\_restrictions

Таблица Age\_restrictions содержит справочник возрастных ограничений для фильмов. Поле id является первичным ключом. Поле age\_restriction содержит обозначение возрастного ограничения (например, 0+, 6+, 12+, 16+, 18+).

## 3.1.25 Таблица Seat\_types

Таблица Seat\_types содержит справочник типов сидений в кинозалах. Поле id является первичным ключом. Поле name содержит название типа сиденья (например, стандартное, VIP, диван, места для инвалидов).

## 

## 3.1.26 Таблица Movie\_actor

Таблица Movie\_actor реализует связь многие-ко-многим между фильмами и актерами. Поле id является первичным ключом. Поле movie\_id является внешним ключом к таблице Movies и указывает на конкретный фильм. Поле actor\_id является внешним ключом к таблице Actors и указывает на актера, снявшегося в данном фильме.

## 

## 3.1.27 Таблица Movie\_genre

Таблица Movie\_genre реализует связь многие-ко-многим между фильмами и жанрами. Поле id является первичным ключом. Поле movie\_id является внешним ключом к таблице Movies и указывает на конкретный фильм. Поле genre\_id является внешним ключом к таблице Genres и определяет один из жанров фильма.

## 

## 3.1.28 Таблица Playlist

Таблица Playlist хранит пользовательские плейлисты фильмов. Поле id является первичным ключом. Поле user\_id является внешним ключом к таблице Users и указывает на владельца плейлиста. Поле name содержит название плейлиста, заданное пользователем.

## 

## 3.1.29 Таблица Playlist\_movies

Таблица Playlist\_movies реализует связь многие-ко-многим между плейлистами и фильмами. Поле id является первичным ключом. Поле movie\_id является внешним ключом к таблице Movies и указывает на фильм, добавленный в плейлист. Поле playlist\_id является внешним ключом к таблице Playlist и определяет плейлист, в который добавлен фильм. Поле created\_at фиксирует дату и время добавления фильма в плейлист.

## 

## 3.1.30 Таблица Recommendation

Таблица Recommendation хранит персонализированные рекомендации фильмов для пользователей. Поле id является первичным ключом. Поле user\_id является внешним ключом к таблице Users и указывает на пользователя, для которого сформирована рекомендация. Поле movie\_id является внешним ключом к таблице Movies и указывает на рекомендуемый фильм. Поле created\_at фиксирует дату и время создания рекомендации.

Таким образом выглядит инфологическая модель базы данных, которая описывает структуру и связи между различными данными, используемыми в проекте. Получилась она достаточно громоздкой, но данная величина обусловлена попыткой объединить в себе сразу три составные части: сервис просмотра кинофильмов, сервис для продажи билетов в кинотеатр и рекламный агреггатор для онлайн-кинотеатра.

Сервис для продажи билетов в кинотеатр должен учитывать расписание сеансов, количество и стоимость мест.

Рекламный агреггатор для онлайн-кинотеатра должен анализировать поведение и предпочтения пользователей, чтобы показывать им наиболее подходящие и эффективные рекламные объявления.

## 3.2 Разработка даталогической модели базы данных

По результатам разработки инфологической модели базы данных построим даталогическую модель базы данных, в которой будут определены типы данных для каждого поля, участвующего в описании разрабатываемой предметной области.

Схема базы данных проектируемой платформы организована в несколько логических групп таблиц, каждая из которых отвечает за хранение определенного аспекта информации предметной области. Структура разработана с учетом требований обеспечения целостности данных через механизмы ограничений и внешних ключей, оптимизации производительности запросов через продуманную систему индексов.

ER-диаграмма базы данных [12] представлена на рисунке 3.2:

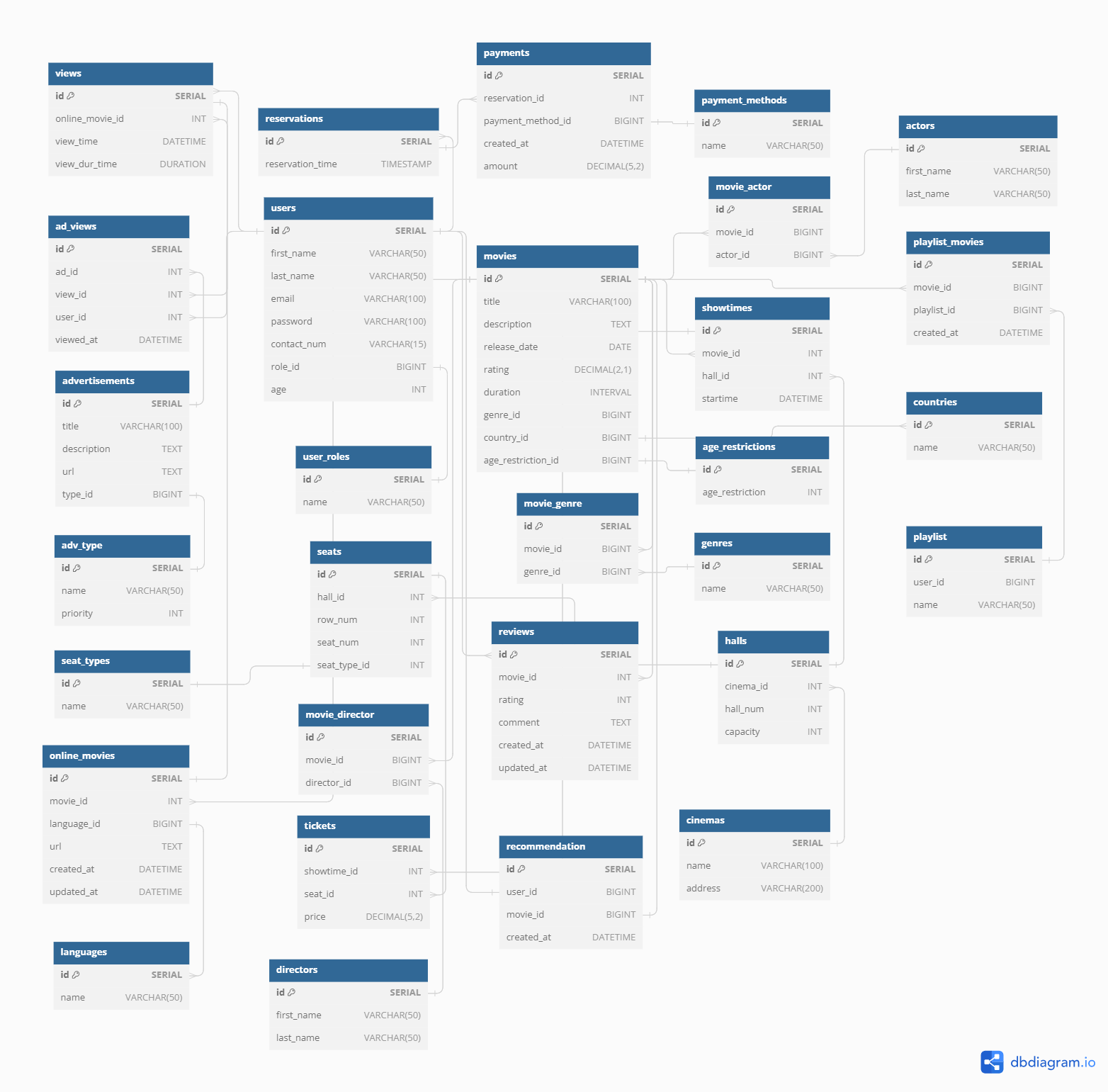


Рисунок 3.2 – ER-диаграмма базы данных

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 [13] 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.

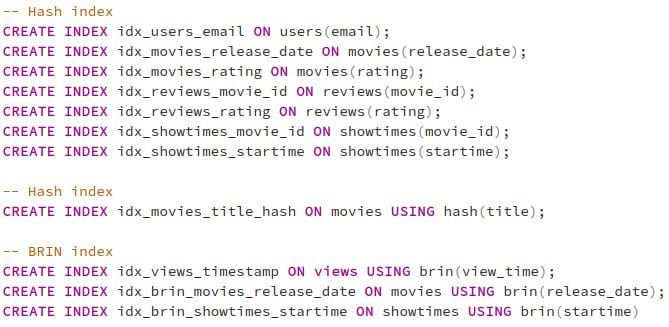


Рисунок 4.1 – Набор индексов, используемых в проектируемой базе данных

## 4.2 Создание хранимых процедур

К хранимым процедурам отнесем следующие процедуры [14]:

1 Процедура add\_movie, принимающая все необходимые параметры для создания записи с фильмом. Сама процедура представлена на рисунке 4.2:



Рисунок 4.2 – Процедура для добавления нового фильма

Аналогичным образом выглядят процедура для добавления пользователя add\_user, реализация которой представлена на рисунке 4.3:

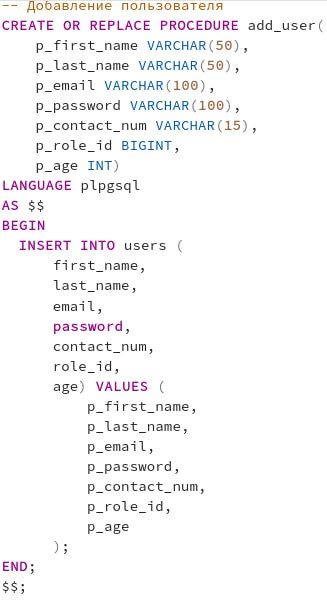


Рисунок 4.3 – Процедура для добавления нового пользователя

Также упомянем реализацию процедуры для добавления режиссера add\_director, один из вариантов которой представлен на рисунке 4.4:



Рисунок 4.4 – Процедура для добавления нового директора

Но в плане реализации эти процедуры являются примером простейших SQL-запросов, а наибольший интерес представляют рекомендательные запросы, а именно рекомендация фильма для пользователя на основе того, наибольшего количества фильмов, просмотренных в определенном жанре [15].

Реализация процедуры рекомендации представлена на рисунке 4.5:

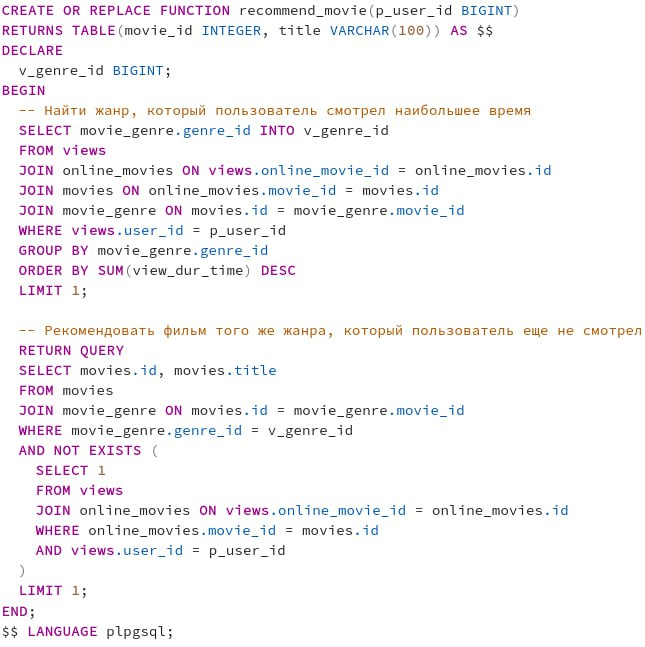


Рисунок 4.5 – Процедура рекомендации фильма по просмотрам

Далее сгенерируем тестовые данные и проверим работоспособность данной процедуры. На рисунке 4.6 изображены данные по фильмам, имеющих genre\_id=1:



Рисунок 4.6 – Фильмы с жанром id=1

В свою очередь пользователь посмотрел фильмы, изображенные на рисунке 4.7:

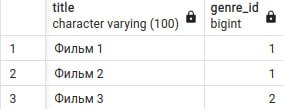


Рисунок 4.7 – Фильмы, которые посмотрел тестовый пользователь

В таком случае получается, что наибольшее количество фильмов, которые посмотрел пользователем это фильмы, с жанром id которого 1. Тогда рекомендательная система должна порекомендовать Фильм 5. В результате выполнения процедуры получаем результат на рисунке 4.8:

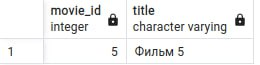


Рисунок 4.8 – Результат работы процедуры

Помимо этой рекомендательной процедуры имеется процедура, которая рекомендует пользователю фильм по жанру, который наберет наибольшее количество среди всех пользовательских плейлистов.

Таким образом была реализована физическая модель базы данных.

# 5 ТЕСТИРОВАНИЕ БАЗЫ ДАННЫХ

## 5.1 Руководство по развертыванию системы

Развертывание платформы по сбору статистики цен полностью автоматизировано с использованием технологии контейнеризации Docker и инструмента оркестрации Docker Compose. Все компоненты системы упакованы в Docker-контейнеры с предварительно настроенными конфигурациями, что обеспечивает быстрое и воспроизводимое развертывание на любом сервере, удовлетворяющем минимальным техническим требованиям.

Процесс развертывания системы сводится к выполнению нескольких команд в терминале сервера. После получения архива проекта в корневую директорию проекта, где расположен файл docker-compose.yml. Развертывание всех компонентов системы выполняется единственной командой: “docker compose up”.

Эта команда автоматически выполняет следующую последовательность действий: загрузку базовых образов Docker из публичных репозиториев для PostgreSQL, сборку пользовательских образов для серверного приложения на FastAPI и клиентского приложения на React на основе Dockerfile, создание Docker-сети для обеспечения взаимодействия между контейнерами, создание томов для постоянного хранения данных базы данных и конфигураций, запуск всех контейнеров в правильном порядке с учетом зависимостей между сервисами.

## 5.2 Руководство пользователя

## 5.2.1 Вход в систему и регистрация

Для доступа к системе пользователь открывает в веб-браузере URL-адрес платформы онлайн-кинотеатра. Отображается главная страница с формой аутентификации, содержащей поле для ввода адреса электронной почты, поле для ввода пароля и кнопку "Войти".

Новые пользователи могут нажать на ссылку "Регистрация", что открывает форму с полями: имя, фамилия, email, пароль, подтверждение пароля, контактный телефон и возраст.

После заполнения всех обязательных полей и нажатия кнопки "Зарегистрироваться" создается учетная запись с ролью обычного пользователя. При корректных учетных данных происходит автоматическое перенаправление на главную страницу системы, содержимое которой определяется ролью пользователя.

## 

## 5.2.2 Главная страница для обычного пользователя

Обычный пользователь с базовой ролью видит главную страницу с каталогом фильмов.

Центральную часть занимает карусель с новинками кино, отображающая постеры, названия, рейтинги и краткие описания последних поступлений.

Ниже расположены тематические подборки: "Популярное сейчас", "Рекомендации для вас" (формируются на основе истории просмотров и оценок пользователя), "Фильмы по жанрам". В верхней части страницы находится панель поиска с возможностью фильтрации по жанрам, странам, году выпуска, рейтингу и возрастному ограничению. Боковое меню содержит разделы: "Мои плейлисты", "История просмотров", "Афиша кинотеатров", "Мои билеты" и "Профиль".

## 

## 5.2.3 Просмотр информации о фильме и онлайн-просмотр

При клике на постер или название фильма пользователь переходит на детальную страницу фильма. Отображается расширенная информация: полное название, постер, описание сюжета, рейтинг, продолжительность, дата выхода, жанры (может быть несколько), страна производства, возрастное ограничение.

Ниже представлены списки режиссеров и актеров с возможностью перехода к их фильмографии. Раздел "Отзывы" содержит комментарии других пользователей с оценками и датами публикации. Если фильм доступен для онлайн-просмотра, отображается кнопка "Смотреть онлайн" с выбором языка озвучки или субтитров из списка доступных языков.

При нажатии открывается видеоплеер с функциями воспроизведения, паузы, перемотки, регулировки громкости и качества изображения. Система автоматически фиксирует время начала просмотра и его продолжительность в таблице Views. В процессе просмотра могут демонстрироваться рекламные ролики согласно типу и приоритету рекламы из таблицы Advertisements, информация о показах сохраняется в таблице Ad\_Views.

## 

## 5.2.4 Оценка фильмов и написание отзывов

После просмотра или в любой момент на странице фильма пользователь может оставить свою оценку и отзыв. Блок "Моя оценка" содержит интерактивную шкалу рейтинга (обычно от 1 до 10 звезд), текстовое поле для развернутого комментария и кнопку "Опубликовать отзыв".

После отправки отзыв сохраняется в таблице Reviews с фиксацией времени создания. Пользователь может вернуться к своему отзыву позже и отредактировать его, при этом обновляется поле updated\_at. Все отзывы пользователя доступны в разделе "Мои отзывы" личного кабинета с возможностью редактирования или удаления.

## 

## 5.2.5 Управление плейлистами

В разделе "Мои плейлисты" пользователь видит список созданных им подборок фильмов. Кнопка "Создать плейлист" открывает модальное окно с полем для ввода названия плейлиста. После создания плейлист отображается в списке, при клике на него открывается страница с добавленными фильмами.

На странице любого фильма присутствует кнопка "Добавить в плейлист" с выпадающим списком доступных плейлистов пользователя. При выборе плейлиста создается запись в таблице Playlist\_movies с фиксацией времени добавления.

Внутри плейлиста пользователь может изменять порядок фильмов перетаскиванием, удалять отдельные позиции или полностью очистить плейлист.

## 

## 5.2.6 Бронирование билетов в кинотеатр

Раздел "Афиша кинотеатров" отображает список доступных кинотеатров с их названиями и адресами из таблицы Cinemas.

При выборе кинотеатра показывается расписание сеансов с информацией о фильме, времени начала, зале и доступности мест. Клик на интересующий сеанс открывает интерактивную схему зала с цветовой индикацией: зеленые места - свободные, красные - занятые, синие - VIP, желтые - для инвалидов (типы из таблицы Seat\_types).

Пользователь выбирает желаемое количество мест кликом по ним, система отображает номера рядов и сидений, типы мест и рассчитывает общую стоимость на основе цен из таблицы Tickets. Кнопка "Забронировать" создает запись в таблице Reservations с фиксацией времени бронирования и привязкой к пользователю. Пользователь перенаправляется на страницу оплаты.

## 

## 5.2.7 Оплата билетов

Страница оплаты отображает детали бронирования: фильм, кинотеатр, зал, дату и время сеанса, выбранные места и общую сумму к оплате. Форма оплаты содержит выбор способа оплаты из доступных вариантов (банковская карта, электронный кошелек, оплата на месте) из справочника Payment\_Methods.

При выборе онлайн-оплаты пользователь вводит необходимые платежные данные и нажимает "Оплатить". После успешной транзакции создается запись в таблице Payments с привязкой к бронированию, указанием способа оплаты, суммы и времени платежа.

Пользователь получает подтверждение на email и может просмотреть электронный билет в разделе "Мои билеты" с QR-кодом для предъявления в кинотеатре.

## 5.2.8 Личный кабинет пользователя

Раздел "Профиль" содержит персональную информацию пользователя с возможностью редактирования. Отображаются поля: имя, фамилия, email (недоступен для изменения), контактный телефон и возраст. Кнопка "Изменить пароль" открывает форму с полями для текущего пароля, нового пароля и его подтверждения.

Вкладка "История просмотров" показывает хронологический список просмотренных онлайн-фильмов с датой, временем и продолжительностью просмотра из таблицы Views. Вкладка "Мои билеты" содержит все забронированные и оплаченные билеты с фильтрацией по статусу (предстоящие, прошедшие, отмененные).

Раздел "Рекомендации" отображает персонализированные подборки фильмов на основе истории просмотров, оценок и предпочтений пользователя, сформированные системой и сохраненные в таблице Recommendation.

## 

## 5.2.9 Функционал для модератора

Модератор с соответствующей ролью имеет доступ к расширенному функционалу через дополнительный пункт меню "Модерация". Раздел "Управление отзывами" отображает таблицу всех отзывов пользователей с колонками: автор, фильм, рейтинг, текст комментария, дата создания, статус.

Модератор может фильтровать отзывы по фильмам, датам, рейтингам и статусу модерации. Каждый отзыв имеет кнопки "Одобрить", "Скрыть" и "Удалить" для управления видимостью контента.

Раздел "Управление контентом" позволяет модератору редактировать информацию о фильмах: обновлять описания, корректировать жанры, добавлять или удалять связи с актерами и режиссерами через таблицы Movie\_actor и Movie\_director, изменять возрастные ограничения и страны производства.

## 

## 5.2.10 Административная панель

Администратор с ролью admin имеет доступ к разделу "Администрирование" с несколькими подразделами. Подраздел "Управление пользователями" отображает таблицу всех зарегистрированных пользователей с колонками: имя, фамилия, email, роль, возраст, дата регистрации, статус активности.

Администратор может изменять роли пользователей, блокировать или разблокировать учетные записи, просматривать детальную активность конкретного пользователя. Подраздел "Управление кинотеатрами" содержит список всех кинотеатров с возможностью добавления новых, редактирования названий и адресов, управления залами.

При выборе кинотеатра отображаются его залы из таблицы Halls с информацией о номере зала и вместимости. Для каждого зала доступно управление схемой расположения мест через редактор, позволяющий добавлять ряды и сиденья с указанием их типов из справочника Seat\_types.

Подраздел "Управление сеансами" предоставляет календарный интерфейс для планирования расписания киносеансов: администратор выбирает фильм из каталога Movies, кинотеатр и зал, указывает дату и время начала, система автоматически создает запись в таблице Showtimes и генерирует билеты в таблице Tickets для всех мест зала с установкой базовых цен, которые затем можно скорректировать индивидуально.

## 

## 5.2.11 Управление онлайн-контентом и рекламой

Подраздел "Онлайн-фильмы" административной панели отображает список всех фильмов с индикацией доступности для онлайн-просмотра. Администратор может добавить фильм к онлайн-каталогу, нажав кнопку "Добавить версию", что открывает форму с полями: выбор языка из справочника Languages, загрузка видеофайла или указание URL-адреса потокового вещания.

После сохранения создается запись в таблице Online\_Movies с автоматической фиксацией времени создания. Для каждой онлайн-версии отображается статистика просмотров с количеством уникальных пользователей и общей продолжительностью из таблицы Views. Подраздел "Управление рекламой" содержит таблицу всех рекламных материалов из Advertisements с колонками: название, описание, тип рекламы, приоритет, статус активности.

Администратор может создавать новые рекламные объявления, указывая заголовок, описание, URL-ссылку и выбирая тип из справочника Adv\_Type с установкой приоритета показа. Статистический дашборд отображает эффективность каждого рекламного объявления: количество показов из таблицы Ad\_Views, распределение по фильмам и временным периодам.

## 

## 5.2.12 Справочники и аналитика

Подраздел "Справочники" предоставляет доступ к управлению вспомогательными таблицами системы.

Администратор может редактировать списки жанров (Genres), стран (Countries), языков (Languages), возрастных ограничений (Age\_restrictions), ролей пользователей (User\_roles), способов оплаты (Payment\_Methods) и типов сидений (Seat\_types). Для каждого справочника доступны операции добавления новых записей, редактирования названий существующих и деактивации неиспользуемых позиций. Подраздел "Аналитика и отчеты" содержит дашборды с ключевыми показателями эффективности платформы.

График "Динамика просмотров" отображает количество онлайн-просмотров по дням, неделям или месяцам на основе данных из Views.

Диаграмма "Популярные фильмы" показывает топ-10 фильмов по количеству просмотров и средним оценкам из Reviews. График "Продажи билетов" визуализирует выручку от бронирований по кинотеатрам и периодам на основе таблицы Payments. Сводная таблица "Активность пользователей" отображает количество зарегистрированных пользователей, активных за период, среднюю продолжительность сеансов и вовлеченность по функциям платформы.

## 

## 5.2.13 Типовые сценарии работы

Типичный сценарий обычного пользователя включает: вход в систему, просмотр рекомендаций на главной странице, выбор интересующего фильма, изучение информации и отзывов, принятие решения о просмотре онлайн или посещении кинотеатра, при выборе онлайн - выбор языка и начало просмотра с возможным показом рекламы, при выборе кинотеатра - просмотр афиши, выбор удобного сеанса и мест, бронирование и оплата билетов, получение подтверждения, после просмотра - оставление отзыва и оценки, добавление понравившегося фильма в личный плейлист.

Модератор ежедневно проверяет новые отзывы в разделе модерации, читает комментарии на предмет нарушений правил сообщества, одобряет корректные отзывы, скрывает или удаляет неприемлемый контент, при необходимости актуализирует информацию о фильмах.

Администратор планирует расписание киносеансов на предстоящую неделю, выбирая популярные новинки и проверенные хиты, распределяя их по залам и времени с учетом вместимости и прогнозируемого спроса, загружает новые фильмы в онлайн-каталог с версиями на разных языках, настраивает рекламные кампании с приоритизацией по типам, анализирует статистику для принятия решений об оптимизации контентной стратегии и ценообразования.

Таким образом, платформа онлайн-кинотеатра обеспечивает удобный и интуитивно понятный интерфейс для всех категорий пользователей. Обычные пользователи получают беспрепятственный доступ к просмотру фильмов онлайн и бронированию билетов в кинотеатры с персонализированными рекомендациями. Модераторы эффективно управляют пользовательским контентом, поддерживая качество сообщества. Администраторы имеют полный контроль над всеми аспектами платформы - от управления каталогом и расписанием до мониторинга бизнес-показателей и настройки рекламы. Система готова к масштабированию и адаптации под меняющиеся потребности киноиндустрии и зрительской аудитории.

# 

# ЗАКЛЮЧЕНИЕ

Результатом работы данного курсового проекта является база данных для такой предметной области, как онлайн-кинотеатр, с возможностью бронирования в физических с элементами вставки и обработки рекламы.

Для разработки была выбрана реляционная СУБД PostgreSQL, обладающая своими преимуществами перед известными конкурентами. Как итог можно сказать, что цели данной курсовой работы достигнуты.

# СПИСОК ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

[1] Bycard.by [Электронный ресурс]:Bycard. Режим доступа: https://bycard.by. Дата доступа: 15.12.2024.

[2] Relax.by [Электронный ресурс]:Relax. Режим доступа: https://relax.by. Дата доступа: 15.12.2024.

[3] Okko.tv [Электронный ресурс]:Okko. Режим доступа: https://okko.tv. Дата доступа: 15.12.2024.

[4] Kinopoisk.ru[Электронный ресурс]:Kinopoisk. Режим доступа: https://kinopoisk.ru. Дата доступа: 15.12.2024.

[5] Иванов А.А. Проектирование баз данных: учебник для вузов / А.А. Иванов. – М.: Издательство “Академия”, 2023. – 512 с.

[6] Петров В.В. Основы работы с СУБД MySQL / В.В. Петров. – СПб.: Питер, 2023. – 1024 с.

[7] Реализация онлайн-кинотеатра с использованием баз данных [Электронный ресурс]: GeekBrains. – Режим доступа: https://geekbrains.ru/posts/online\_cinema\_database. Дата доступа: 15.12.2024.

[8] Как создать базу данных для бронирования мест [Электронный ресурс]: SQLPro. – Режим доступа: https://sqlpro.ru/create-database-for-booking. Дата доступа: 15.12.2024.

[9] Базы данных в киноиндустрии [Электронный ресурс]: DataArt. Режим доступа: https://dataart.ru/database-in-cinema-industry/. Дата доступа: 15.12.2024.

[10] Применение баз данных в кинотеатрах [Электронный ресурс]: CinemaDB. – Режим доступа: https://cinemadb.ru/. Дата доступа: 15.12.2024.

[11] Системы управления базами данных. Основные характеристики. [Электронный ресурс]: DBstudy. – Режим доступа: https://dbstudy.net/814892/informatika/osnovnye\_harakteristiki\_sistem\_upravleniya\_bazami\_dannyh. Дата доступа: 15.12.2024.

[12] 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: 15.12.2024

[13] Смирнов И.И. Базы данных и Delphi: учебное пособие / И.И. Смирнов. – М.: Издательство “Академия”, 2023. – 350 с.

[14] Петрова Е.А. Применение SQL в проектах на Java / Е.А. Петрова. – СПб.: Питер, 2023. – 800 с.

[15] Создание онлайн-кинотеатра с использованием баз данных [Электронный ресурс]: ProgLib. – Режим доступа: https://proglib.ru/create-online-cinema-database. Дата доступа: 15.12.2024.

[16] Базы данных для бронирования мест в кинотеатрах [Электронный ресурс]: DBPro. – Режим доступа: https://dbpro.ru/booking-database-for-cinemas. Дата доступа: 15.12.2024.

[17] Базы данных в киноиндустрии: новые тенденции [Электронный ресурс]: CinemaTrends. Режим доступа: https://cinematrends.ru/database-trends-in-cinema-industry/. Дата доступа: 15.12.2024.

[18] Применение баз данных в кинотеатрах: кейс-стади [Электронный ресурс]: CinemaCase. – Режим доступа: https://cinemacase.ru/. Дата доступа: 15.12.2024.

[19] 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: 15.12.2024

**ПРИЛОЖЕНИЕ А  
(обязательное)  
Листинг кода программы**

CREATE TABLE "users" (

"id" SERIAL PRIMARY KEY,

"first\_name" VARCHAR(50),

"last\_name" VARCHAR(50),

"email" VARCHAR(100) UNIQUE,

"password" VARCHAR(100),

"contact\_num" VARCHAR(15),

"role\_id" BIGINT,

"age" INT

);

CREATE TABLE "user\_roles" (

"id" SERIAL PRIMARY KEY,

"name" VARCHAR(50)

);

CREATE TABLE "movies" (

"id" SERIAL PRIMARY KEY,

"title" VARCHAR(100),

"description" TEXT,

"release\_date" DATE,

"rating" DECIMAL(2,1),

"duration" INTERVAL,

"genre\_id" BIGINT,

"country\_id" BIGINT,

"age\_restriction\_id" BIGINT,

);

CREATE TABLE "movie\_director" (

"id" SERIAL PRIMARY KEY,

"movie\_id" BIGINT,

"director\_id" BIGINT

);

CREATE TABLE "genres" (

"id" SERIAL PRIMARY KEY,

"name" VARCHAR(50)

);

CREATE TABLE "directors" (

"id" SERIAL PRIMARY KEY,

"first\_name" VARCHAR(50),

"last\_name" VARCHAR(50)

);

CREATE TABLE "actors" (

"id" SERIAL PRIMARY KEY,

"first\_name" VARCHAR(50),

"last\_name" VARCHAR(50)

);

CREATE TABLE "cinemas" (

"id" SERIAL PRIMARY KEY,

"name" VARCHAR(100),

"address" VARCHAR(200)

);

CREATE TABLE "halls" (

"id" SERIAL PRIMARY KEY,

"cinema\_id" INT,

"hall\_num" INT,

"capacity" INT

);

CREATE TABLE "seats" (

"id" SERIAL PRIMARY KEY,

"hall\_id" INT,

"row\_num" INT,

"seat\_num" INT,

"seat\_type\_id" INT

);

CREATE TABLE "showtimes" (

"id" SERIAL PRIMARY KEY,

"movie\_id" INT,

"hall\_id" INT,

"startime" DATETIME

);

CREATE TABLE "tickets" (

"id" SERIAL PRIMARY KEY,

"showtime\_id" INT,

"seat\_id" INT,

"price" DECIMAL(5,2)

);

CREATE TABLE "reservations" (

"id" SERIAL PRIMARY KEY,

"user\_id" BIGINT,

"reservation\_time" TIMESTAMP

);

CREATE TABLE "online\_movies" (

"id" SERIAL PRIMARY KEY,

"movie\_id" INT,

"language\_id" BIGINT,

"url" TEXT,

"created\_at" DATETIME,

"updated\_at" DATETIME

);

CREATE TABLE "views" (

"id" SERIAL PRIMARY KEY,

"online\_movie\_id" INT,

"view\_time" DATETIME,

"view\_dur\_time" DURATION,

"user\_id" INT

);

CREATE TABLE "reviews" (

"id" SERIAL PRIMARY KEY,

"movie\_id" INT,

"rating" INT,

"comment" TEXT,

"created\_at" DATETIME,

"updated\_at" DATETIME

);

CREATE TABLE "payments" (

"id" SERIAL PRIMARY KEY,

"reservation\_id" INT,

"payment\_method\_id" BIGINT,

"created\_at" DATETIME,

"amount" DECIMAL(5,2)

);

CREATE TABLE "payment\_methods" (

"id" SERIAL PRIMARY KEY,

"name" VARCHAR(50) UNIQUE

);

CREATE TABLE "advertisements" (

"id" SERIAL PRIMARY KEY,

"title" VARCHAR(100),

"description" TEXT,

"url" TEXT,

"type\_id" BIGINT

);

CREATE TABLE "adv\_type" (

"id" SERIAL PRIMARY KEY,

"name" VARCHAR(50) UNIQUE,

"priority" INT

);

CREATE TABLE "ad\_views" (

"id" SERIAL PRIMARY KEY,

"ad\_id" INT,

"view\_id" INT,

"user\_id" INT,

"viewed\_at" DATETIME

);

CREATE TABLE "countries" (

"id" SERIAL PRIMARY KEY,

"name" VARCHAR(50) UNIQUE

);

CREATE TABLE "languages" (

"id" SERIAL PRIMARY KEY,

"name" VARCHAR(50) UNIQUE

);

CREATE TABLE "age\_restrictions" (

"id" SERIAL PRIMARY KEY,

"age\_restriction" INT UNIQUE

);

CREATE TABLE "seat\_types" (

"id" SERIAL PRIMARY KEY,

"name" VARCHAR(50) UNIQUE

);

CREATE TABLE "movie\_actor" (

"id" SERIAL PRIMARY KEY,

"movie\_id" BIGINT,

"actor\_id" BIGINT

);

CREATE TABLE "movie\_genre" (

"id" SERIAL PRIMARY KEY,

"movie\_id" BIGINT,

"genre\_id" BIGINT

);

CREATE TABLE "playlist" (

"id" SERIAL PRIMARY KEY,

"user\_id" BIGINT,

"name" VARCHAR(50)

);

CREATE TABLE "playlist\_movies" (

"id" SERIAL PRIMARY KEY,

"movie\_id" BIGINT,

"playlist\_id" BIGINT,

"created\_at" DATETIME

);

CREATE TABLE "recommendation" (

"id" SERIAL PRIMARY KEY,

"user\_id" BIGINT,

"movie\_id" BIGINT,

"review\_id" BIGINT,

"created\_at" TIMESTAMP

);

ALTER TABLE "users" ADD FOREIGN KEY ("role\_id") REFERENCES "user\_roles" ("id");

ALTER TABLE "online\_movies" ADD FOREIGN KEY ("language\_id") REFERENCES "languages" ("id");

ALTER TABLE "movies" ADD FOREIGN KEY ("age\_restriction\_id") REFERENCES "age\_restrictions" ("id");

ALTER TABLE "payments" ADD FOREIGN KEY ("payment\_method\_id") REFERENCES "payments" ("id");

ALTER TABLE "seats" ADD FOREIGN KEY ("seat\_type\_id") REFERENCES "seat\_types" ("id");

ALTER TABLE "recommendation" ADD FOREIGN KEY ("review\_id") REFERENCES "reviews" ("id");

ALTER TABLE "advertisements" ADD FOREIGN KEY ("type\_id") REFERENCES "adv\_type" ("id");

ALTER TABLE "halls" ADD FOREIGN KEY ("cinema\_id") REFERENCES "cinemas" ("id");

ALTER TABLE "seats" ADD FOREIGN KEY ("hall\_id") REFERENCES "halls" ("id");

ALTER TABLE "showtimes" ADD FOREIGN KEY ("movie\_id") REFERENCES "movies" ("id");

ALTER TABLE "showtimes" ADD FOREIGN KEY ("hall\_id") REFERENCES "halls" ("id");

ALTER TABLE "tickets" ADD FOREIGN KEY ("showtime\_id") REFERENCES "showtimes" ("id");

ALTER TABLE "tickets" ADD FOREIGN KEY ("seat\_id") REFERENCES "seats" ("id");

ALTER TABLE "reservations" ADD FOREIGN KEY ("user\_id") REFERENCES "users" ("id");

ALTER TABLE "online\_movies" ADD FOREIGN KEY ("movie\_id") REFERENCES "movies" ("id");

ALTER TABLE "views" ADD FOREIGN KEY ("user\_id") REFERENCES "users" ("id");

ALTER TABLE "views" ADD FOREIGN KEY ("online\_movie\_id") REFERENCES "online\_movies" ("id");

ALTER TABLE "reviews" ADD FOREIGN KEY ("id") REFERENCES "users" ("id");

ALTER TABLE "reviews" ADD FOREIGN KEY ("movie\_id") REFERENCES "movies" ("id");

ALTER TABLE "payments" ADD FOREIGN KEY ("reservation\_id") REFERENCES "reservations" ("id");

ALTER TABLE "ad\_views" ADD FOREIGN KEY ("ad\_id") REFERENCES "advertisements" ("id");

ALTER TABLE "ad\_views" ADD FOREIGN KEY ("user\_id") REFERENCES "users" ("id");

ALTER TABLE "ad\_views" ADD FOREIGN KEY ("view\_id") REFERENCES "views" ("id");

ALTER TABLE "languages" ADD FOREIGN KEY ("id") REFERENCES "online\_movies" ("language\_id");

ALTER TABLE "movie\_actor" ADD FOREIGN KEY ("movie\_id") REFERENCES "movies" ("id");

ALTER TABLE "movie\_actor" ADD FOREIGN KEY ("actor\_id") REFERENCES "actors" ("id");

ALTER TABLE "movie\_genre" ADD FOREIGN KEY ("movie\_id") REFERENCES "movies" ("id");

ALTER TABLE "movie\_genre" ADD FOREIGN KEY ("genre\_id") REFERENCES "genres" ("id");

ALTER TABLE "age\_restrictions" ADD FOREIGN KEY ("id") REFERENCES "movies" ("age\_restriction\_id");

ALTER TABLE "payment\_methods" ADD FOREIGN KEY ("id") REFERENCES "payments" ("payment\_method\_id");

ALTER TABLE "seat\_types" ADD FOREIGN KEY ("id") REFERENCES "seats" ("seat\_type\_id");

ALTER TABLE "playlist\_movies" ADD FOREIGN KEY ("playlist\_id") REFERENCES "playlist" ("id");

ALTER TABLE "playlist\_movies" ADD FOREIGN KEY ("movie\_id") REFERENCES "movies" ("id");

ALTER TABLE "users" ADD FOREIGN KEY ("id") REFERENCES "recommendation" ("user\_id");

ALTER TABLE "movies" ADD FOREIGN KEY ("id") REFERENCES "recommendation" ("movie\_id");

ALTER TABLE "adv\_type" ADD FOREIGN KEY ("id") REFERENCES "advertisements" ("type\_id");

ALTER TABLE "movie\_director" ADD FOREIGN KEY ("movie\_id") REFERENCES "movies" ("id");

ALTER TABLE "movie\_director" ADD FOREIGN KEY ("director\_id") REFERENCES "directors" ("id");

-- Hash index

CREATE INDEX idx\_users\_email ON users(email);

CREATE INDEX idx\_movies\_title ON movies(title);

CREATE INDEX idx\_movies\_release\_date ON movies(release\_date);

CREATE INDEX idx\_movies\_rating ON movies(rating);

CREATE INDEX idx\_reviews\_movie\_id ON reviews(movie\_id);

CREATE INDEX idx\_reviews\_rating ON reviews(rating);

CREATE INDEX idx\_showtimes\_movie\_id ON showtimes(movie\_id);

CREATE INDEX idx\_showtimes\_startime ON showtimes(startime);

-- Hash index

CREATE INDEX idx\_movies\_title\_hash ON movies USING hash(title);

-- BRIN index

-- Assuming you have a column for timestamp in views table

CREATE INDEX idx\_views\_timestamp ON views USING brin(view\_time);

CREATE INDEX idx\_brin\_movies\_release\_date ON movies USING brin(release\_date);

CREATE INDEX idx\_brin\_showtimes\_startime ON showtimes USING brin(startime);

CREATE OR REPLACE FUNCTION update\_movie\_rating()

RETURNS TRIGGER AS $$

BEGIN

UPDATE movies

SET rating = (SELECT AVG(rating) FROM reviews WHERE movie\_id = COALESCE(NEW.movie\_id, OLD.movie\_id))

WHERE id = COALESCE(NEW.movie\_id, OLD.movie\_id);

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER update\_movie\_rating\_trigger

AFTER INSERT OR DELETE ON reviews

FOR EACH ROW EXECUTE PROCEDURE update\_movie\_rating();

CREATE OR REPLACE FUNCTION recommend\_movie(p\_user\_id BIGINT)

RETURNS TABLE(movie\_id INTEGER, title VARCHAR(100)) AS $$

DECLARE

v\_genre\_id BIGINT;

BEGIN

-- Найти жанр, который пользователь смотрел наибольшее время

SELECT movie\_genre.genre\_id INTO v\_genre\_id

FROM views

JOIN online\_movies ON views.online\_movie\_id = online\_movies.id

JOIN movies ON online\_movies.movie\_id = movies.id

JOIN movie\_genre ON movies.id = movie\_genre.movie\_id

WHERE views.user\_id = p\_user\_id

GROUP BY movie\_genre.genre\_id

ORDER BY SUM(view\_dur\_time) DESC

LIMIT 1;

-- Рекомендовать фильм того же жанра, который пользователь еще не смотрел

RETURN QUERY

SELECT movies.id, movies.title

FROM movies

JOIN movie\_genre ON movies.id = movie\_genre.movie\_id

WHERE movie\_genre.genre\_id = v\_genre\_id

AND NOT EXISTS (

SELECT 1

FROM views

JOIN online\_movies ON views.online\_movie\_id = online\_movies.id

WHERE online\_movies.movie\_id = movies.id

AND views.user\_id = p\_user\_id

)

LIMIT 1;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION recommend\_movie\_based\_on\_playlists(p\_user\_id BIGINT)

RETURNS TABLE(movie\_id INTEGER, title VARCHAR(100)) AS $$

DECLARE

v\_genre\_id BIGINT;

BEGIN

-- Найти жанр, который часто встречается в плейлистах пользователя

SELECT movie\_genre.genre\_id INTO v\_genre\_id

FROM playlist\_movies

JOIN playlist ON playlist\_movies.playlist\_id = playlist.id

JOIN movies ON playlist\_movies.movie\_id = movies.id

JOIN movie\_genre ON movies.id = movie\_genre.movie\_id

WHERE playlist.user\_id = p\_user\_id

GROUP BY movie\_genre.genre\_id

ORDER BY COUNT(\*) DESC

LIMIT 1;

-- Рекомендовать фильм того же жанра, который пользователь еще не смотрел и который еще не в плейлисте

RETURN QUERY

SELECT movies.id, movies.title

FROM movies

JOIN movie\_genre ON movies.id = movie\_genre.movie\_id

WHERE movie\_genre.genre\_id = v\_genre\_id

AND NOT EXISTS (

SELECT 1

FROM views

JOIN online\_movies ON views.online\_movie\_id = online\_movies.id

WHERE online\_movies.movie\_id = movies.id

AND views.user\_id = p\_user\_id

)

AND NOT EXISTS (

SELECT 1

FROM playlist\_movies

JOIN playlist ON playlist\_movies.playlist\_id = playlist.id

WHERE playlist\_movies.movie\_id = movies.id

AND playlist.user\_id = p\_user\_id

)

LIMIT 1;

END;

$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION generate\_ad\_view()

RETURNS TRIGGER AS $$

DECLARE

v\_ad\_id INTEGER;

BEGIN

-- Найти рекламу с наивысшим приоритетом

SELECT ad.id INTO v\_ad\_id

FROM advertisements ad

JOIN adv\_type at ON ad.type\_id = at.id

ORDER BY at.priority DESC

LIMIT 1;

-- Создать запись просмотра рекламы

INSERT INTO ad\_views (ad\_id, movie\_id, user\_id, viewed\_at, successful)

VALUES (v\_ad\_id, (SELECT movie\_id FROM online\_movies WHERE id = NEW.online\_movie\_id), NEW.user\_id, NOW(), FALSE);

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER generate\_ad\_view\_trigger

BEFORE INSERT ON views

FOR EACH ROW EXECUTE PROCEDURE generate\_ad\_view();

CREATE OR REPLACE FUNCTION check\_age\_restriction() RETURNS TRIGGER AS $$

DECLARE

user\_age INT;

movie\_age\_restriction INT;

BEGIN

SELECT age INTO user\_age FROM users WHERE id = NEW.user\_id;

SELECT age\_restriction\_id INTO movie\_age\_restriction FROM movies WHERE id = (SELECT movie\_id FROM showtimes WHERE id = NEW.showtime\_id);

IF user\_age < movie\_age\_restriction THEN

RAISE EXCEPTION 'User does not meet the age restriction for this movie.';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql; CREATE OR REPLACE FUNCTION check\_age\_restriction()

RETURNS TRIGGER AS $$

DECLARE

v\_user\_age INT;

v\_movie\_age\_restriction INT;

BEGIN

-- Получить возраст пользователя

SELECT age INTO v\_user\_age

FROM users

WHERE id = NEW.user\_id;

-- Получить возрастное ограничение фильма

SELECT age\_restriction INTO v\_movie\_age\_restriction

FROM movies m

JOIN showtimes s ON m.id = s.movie\_id

JOIN tickets t ON s.id = t.showtime\_id

WHERE t.id = NEW.id;

**ПРИЛОЖЕНИЕ Б  
(обязательное)  
Схема базы данных**



**ПРИЛОЖЕНИЕ В  
(обязательное)  
UML usecase диаграмма**

