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  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_В.С.Плиска |
|  | \_\_\_.\_\_\_\_.2023 |

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

на тему

**БАЗА ДАННЫХ ПРИЛОЖЕНИЯ ОНЛАЙН-КИНОТЕАТРА**

БГУИР КП 1-40 04 01

|  |  |
| --- | --- |
|  | Выполнил студент группы 053505  Вашкевич Николай Владимирович  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |
|  | Курсовой проект представлен на проверку \_\_\_.\_\_\_\_.2023  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |

Минск 2023

**СОДЕРЖАНИЕ**

[1 Анализ предметной области 5](#_Toc153529165)

[1.1 Обзор онлайн ресурса для покупки билетов Bycard.by 5](#_Toc153529166)

[1.2 Обзор онлайн ресурса для бронирования билетов Relax.by 7](#_Toc153529167)

[1.3 Обзор онлайн-кинотеатра Okko.tv 9](#_Toc153529168)

[1.4 Обзор онлайн-кинотеатра Кинопоиск HD 12](#_Toc153529169)

[2 Формирование функциональых требований 14](#_Toc153529170)

[2.1 Функциональные требования к разрабатываемой базе данных 14](#_Toc153529171)

[2.2 Анализ существующих подходов к разработке баз данных 14](#_Toc153529172)

[2.3 Выбор базы данных для разработки предметной области 23](#_Toc153529173)

[3 Проектирование базы данных 25](#_Toc153529174)

[3.1 Инфологическая модель 25](#_Toc153529175)

[3.2 Разработка даталогической модели базы данных 29](#_Toc153529176)

[4 Разработка базы данных 31](#_Toc153529177)

[4.1 Создание исходных таблиц, индексов и ограничений 31](#_Toc153529178)

[4.2 Создание хранимых процедур 31](#_Toc153529179)

[4.3 Реализация триггеров 35](#_Toc153529180)

[Заключение 38](#_Toc153529181)

[Список литературных источников 39](#_Toc153529182)

[Приложение А (обязательное) Листинг кода программы 41](#_Toc153529183)

[Приложение Б (обязательное) ER-диаграмма базы данных 55](#_Toc153529184)

**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 Okko.tv;

– онлайн-кинотеатр Кинопоиск HD.

## 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 [1]. На рисунке 1.1 представлен графический интерфейс приложения:

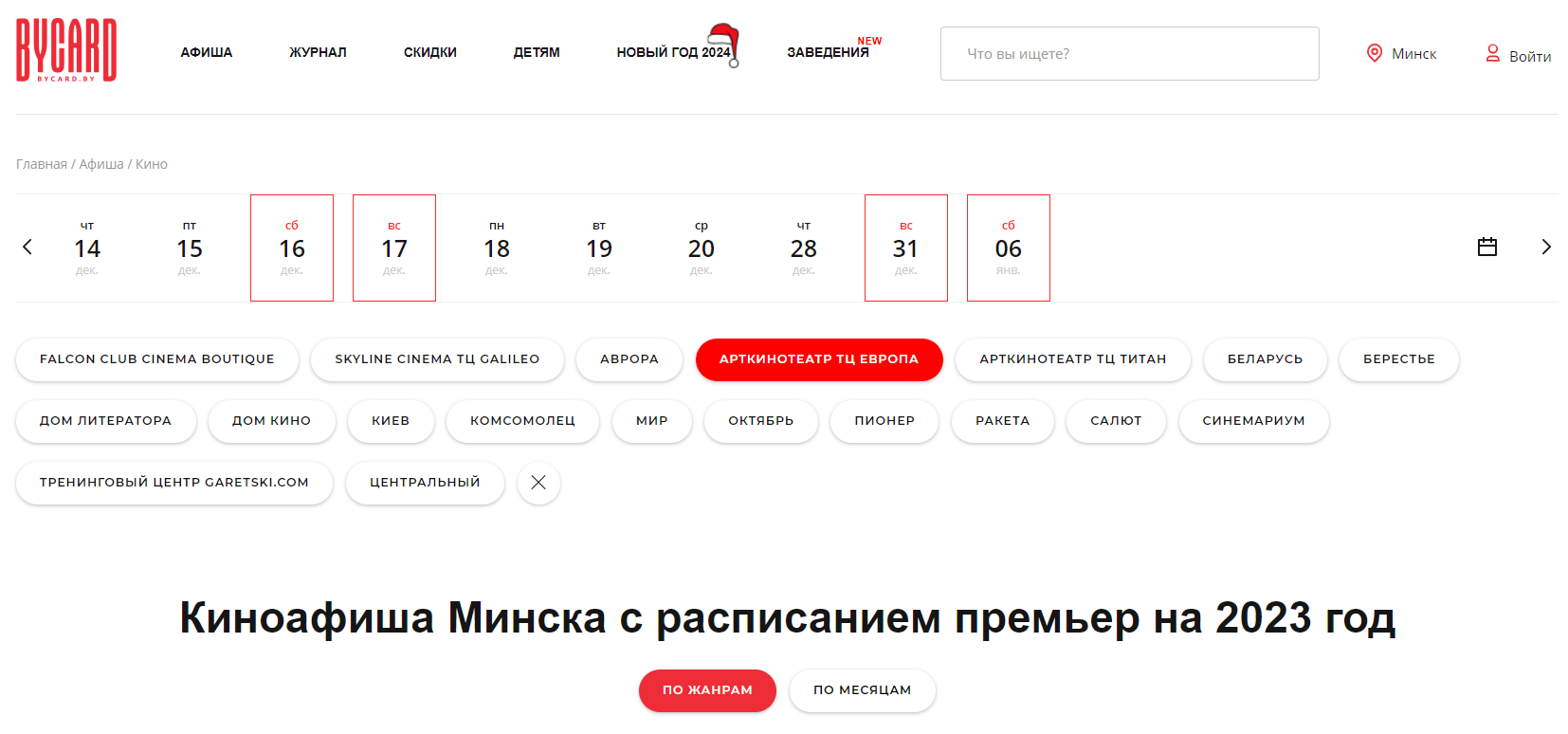


Рисунок 1.1 – Графический интерфейс Bycard.by

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.

Такой информацией может являться более конкретная информация по каждому из мероприятий, пример которой представлен на рисунке 1.2:

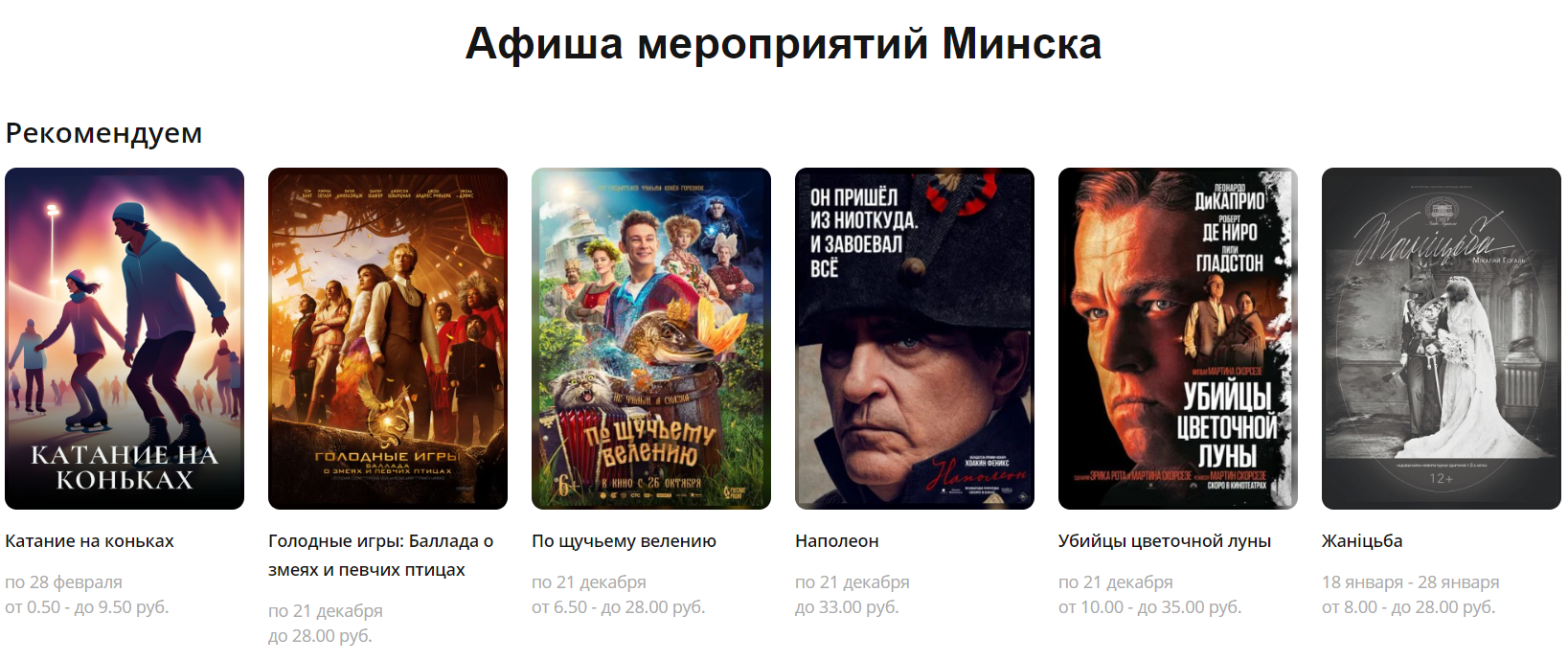


Рисунок 1.2 – Список мероприятий в Bycard.by

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 1.3:

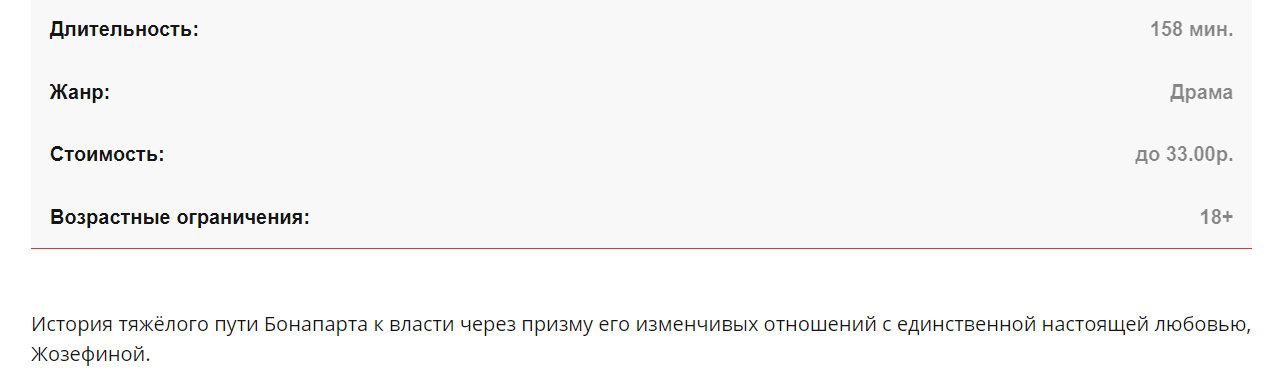


Рисунок 1.3 – Описание выбранного фильма

Исходя из изображенной на рисунке 1.3 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 1.4 представлена следующим образом [2]:



Рисунок 1.4 – 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 1.5:

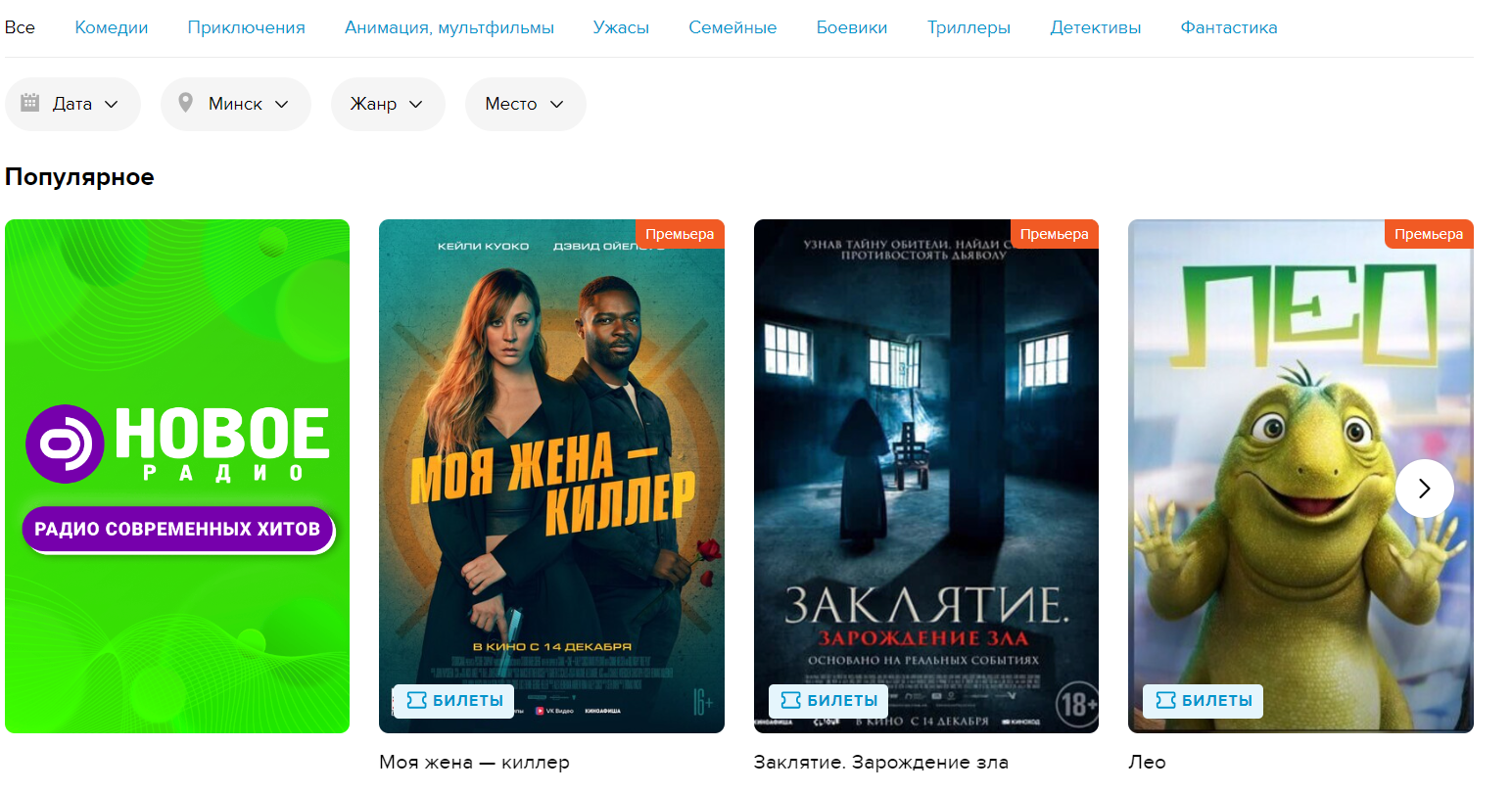


Рисунок 1.5 – Страница с информацией о фильмах

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, пример рекламного баннера представлен на рисунке 1.6:



Рисунок 1.6 – Рекламный баннер Relax.by

При разработке базы данных для онлайн-кинотеатра этот подход, 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.

## Обзор онлайн-кинотеатра Okko.tv

Okko.tv — это один из крупнейших легальных онлайн-кинотеатров в России, который предлагает пользователям смотреть фильмы, сериалы и мультфильмы в высоком качестве и без рекламы.

Главное окно при входе на ресурс представлено на рисунке 1.7:

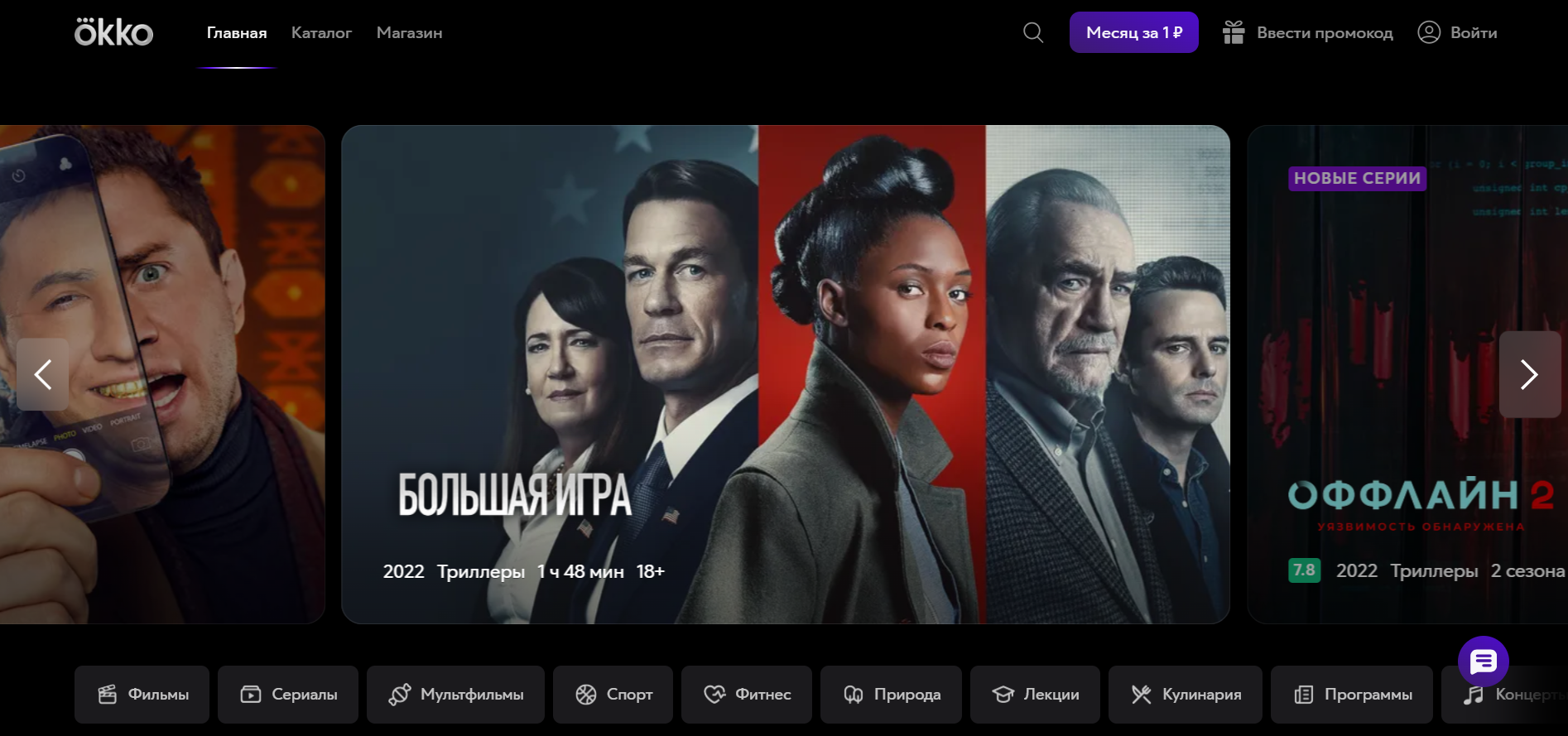


Рисунок 1.7 – Главное меню Okko.tv

По первому впечатлению от главного окна Okko.tv можно сделать предварительный вывод, что здесь в рамках киноискусства следует выделить сущность фильма с возможностью выбора жанра фильма, а также возрастного ограничения. Касательно общей картины работы данного онлайн-кинотеатра можно выделить следующие сущности [3]:

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

2 Подписка — это сущность, которая представляет собой услугу, предоставляемую ресурсом, которая дает доступ к определенному набору фильмов за определенную плату. Атрибуты подписки могут быть название, цена, срок действия, количество фильмов и т.д. Ключом подписки может быть ее уникальный идентификатор или название. Помимо этого, сервис предоставляет разные виды подписок, которые могут быть представлены в виде неотрицательного целочисленного поля, либо в виде отдельной сущности базы данных.

3 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 и т.д. Ключом пользователя может быть его уникальный идентификатор или логин.

4 Заказ — это сущность, которая представляет собой действие, которое впоследствии будет совершать пользователь, когда он покупает подписку или фильм на ресурсе. Атрибуты заказа могут быть номер, дата, время, сумма, способ оплаты, статус и т.д. Ключом заказа может быть его уникальный номер.

5 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.

Также стоит заметить отсутствие в данном ресурсе возможности собирать существующие фильмы в плейлисты, как это обычно делается в музыкальных приложениях, чтобы можно было ими делиться с другими пользователями или составить рекомендацию не только на основе последних просмотренных фильмов, а иметь возможность рекомендовать фильмы пользователю на основе плейлистов, если они вдруг были составлены для просмотра фильмов по настроению.

Также в Okko.tv реализован механизм промокодов, которые используются в качестве маркетингового инструмента для привлечения новых пользователей и удержания старых, пример которого представлен на рисунке 1.8:

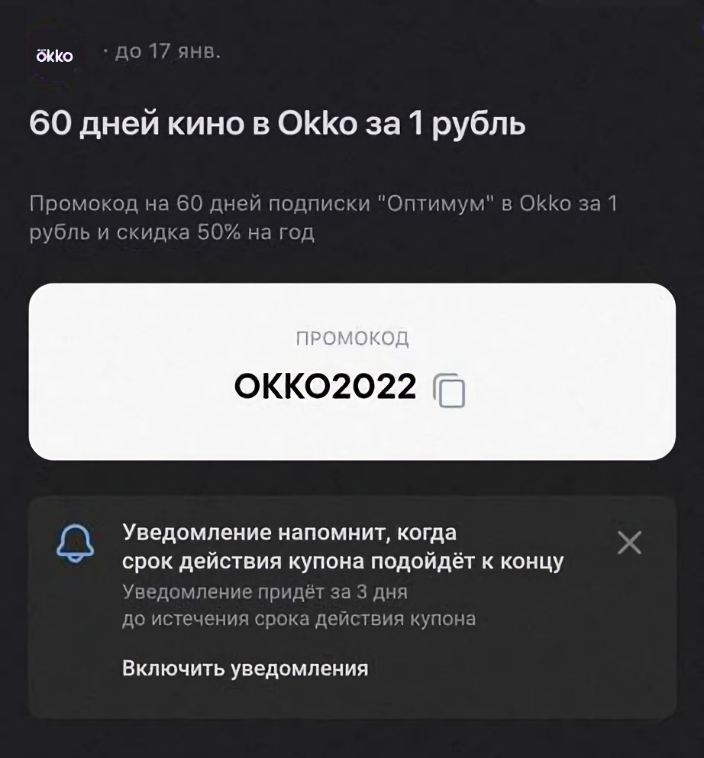


Рисунок 1.8 – Сообщение с промокодом от Okko.tv

Таким образом данное окно содержит сам промокод, а также уведомление о том, до какого периода он действителен.

## Обзор онлайн-кинотеатра Кинопоиск HD

Меню с выбранным фильмом в данном кинотеатре выглядит следующим образом:

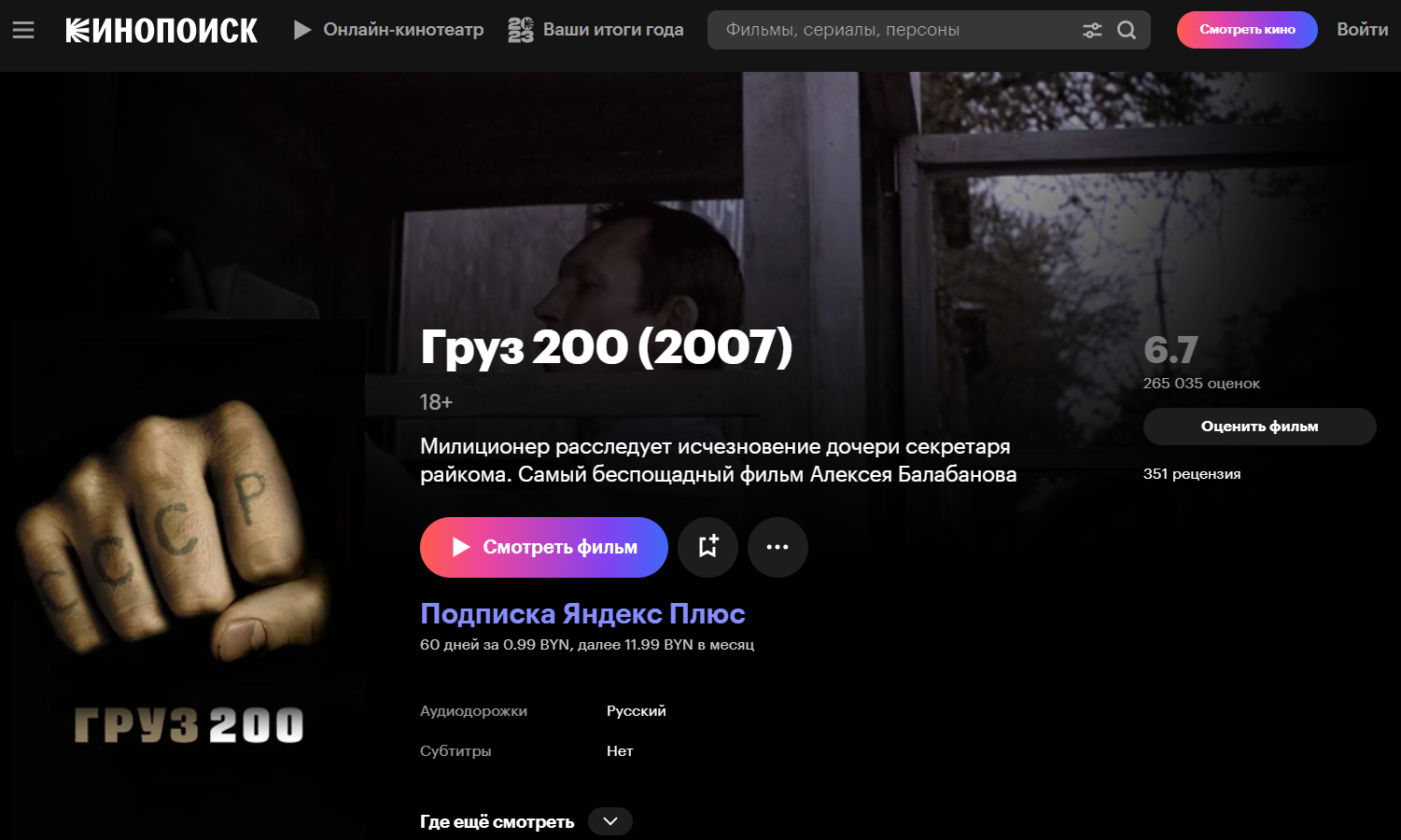


Рисунок 1.9 – Меню с информацией о выбранном фильме

Из данного рисунка видно, что отличительной чертой от других сервисов, является наличие информации о среднем рейтинге фильма, о количестве оценок и рецензий [4]. 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.

# 2 ФОРМИРОВАНИЕ ФУНКЦИОНАЛЬЫХ 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:

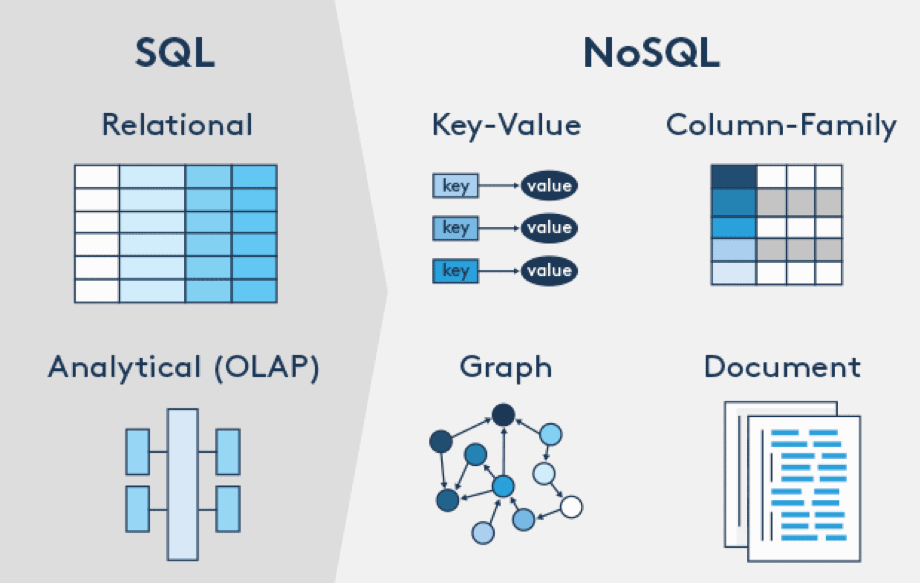


Рисунок 2.1 – Основные виды SQL и NoSQL баз данных

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

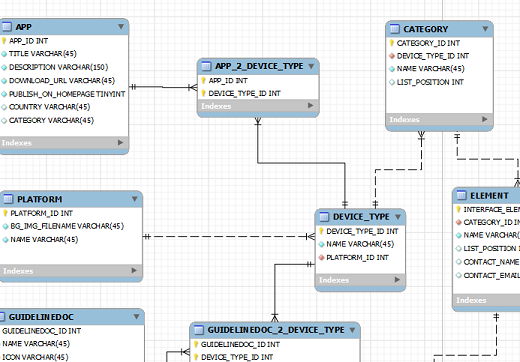


Рисунок 2.2 – Пример организации данных в реляционной теории

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

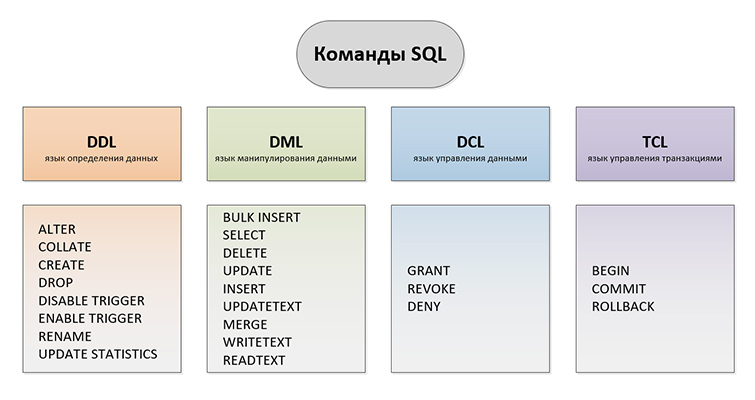


Рисунок 2.3 – Структура команд SQL

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:

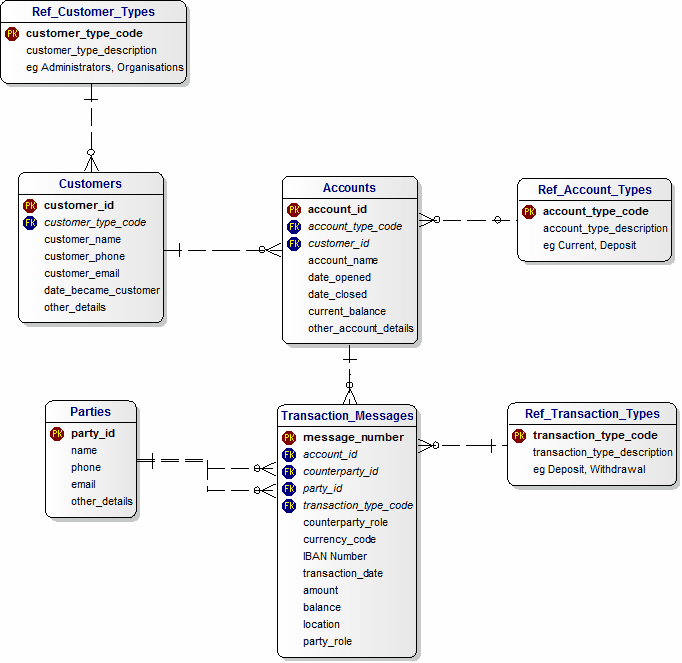


Рисунок 2.4 – Логическое представление базы данных

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:

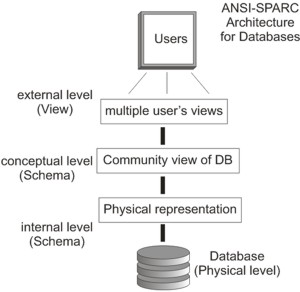


Рисунок 2.4 – Схема организации базы данных с физическим уровнем

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.

# 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:

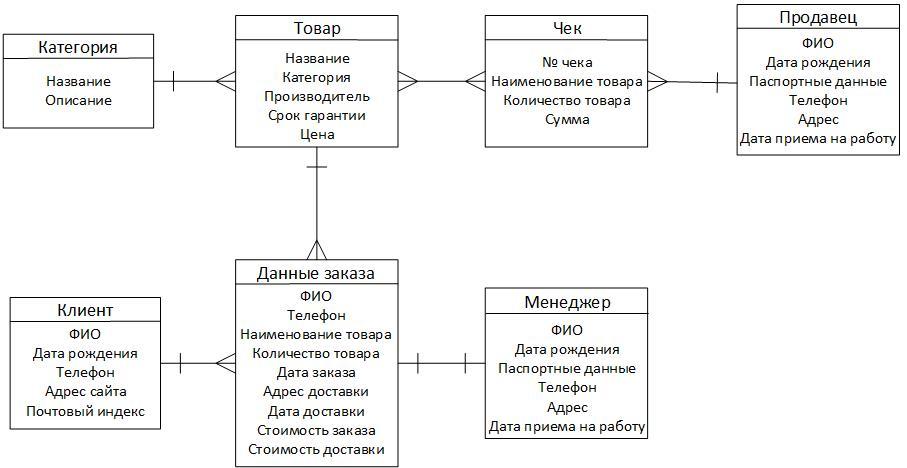


Рисунок 3.1 – Графический пример инфологической модели

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

1 User (пользователь):

– first\_name – имя.

– last\_name – фамилия.

– email – почта.

– password – пароль.

– contact\_num – контактный телефон.

– role\_id – идентификатор роли пользователя (внешний ключ к сущности Roles).

– age – возраст.

2 User\_roles (роли пользователя):

– name – название роли.

3 Movies (фильмы):

– title – название фильма.

– description – описание.

– release\_date – дата выхода.

– rating – рейтинг.

– duration – продолжительность.

– genre\_id – идентификатор жанра (внешний ключ к сущности Genre).

– country\_id – идентификатор страны (внешний ключ к сущности Country).

– age\_restriction\_id – идентификатор возрастного ограничения (внешний ключ к сущности Age\_restriction).

4 Movie\_director (режиссеры фильмов):

– movie\_id – идентификатор фильма (внешний ключ к сущности Movie).

– director\_id – идентификатор режиссера (внешний ключ к сущности Director).

5 Genres (жанры кино)

– name – название жанра.

6 Directors (режиссеры)

– first\_name – имя режиссера.

– last\_name – фамилия режиссера.

7 Actors (актеры)

– first\_name – имя актера.

– last\_name – фамилия актера.

8 Cinemas (кинофильм)

– name – название кинотеатра.

– address – адрес.

9 Halls (кинозал)

– cinema\_id – идентификатор кинотеатра (внешний ключ к сущности Cinemas).

– hall\_num – переменная для конкретизации сеанса.

– capacity – вместимость кинозала.

10 Seats (сиденья)

– hall\_id – идентификатор зала (внешний ключ к сущности Halls)

– row\_num – номер ряда.

– seat\_num – номер сиденья.

– seat\_type\_id – идентификатор типа сиденья (внешний ключ к сущности SeatType).

11 Showtimes (сеанс)

– movie\_id – идентификатор фильма (внешний ключ к сущности Movies).

– hall\_id – идентификатор кинозала (внешний ключ к сущности Halls).

– startime – время начала сеанса.

12 Tickets (билеты)

– showtime\_id – идентификатор сеанса (внешний ключ к сущности Showtimes).

– seat\_id – идентификатор места (внешний ключ к сущности Seats).

– price – стоимость билета.

13 Reservations (бронь)

– reservation\_time – время бронирования.

– user\_id – идентификатор пользователя (внешний ключ к сущности Users).

14 Online\_Movies (онлайн-кинофильм).

– movie\_id – идентификатор фильма (внешний ключ к сущности Movies).

– language\_id идентификатор языка (внешний ключ к сущности Languages).

– url – уникальная ссылка на фильм.

– created\_at – время создания записи.

– updated\_at – время изменения записи.

15 Views (просмотр)

– online\_movie\_id идентификатор онлайн-фильма (внешний ключ к сущности Online\_Movies).

– view\_time – время просмотра.

– view\_dur\_time – продолжительность просмотра.

– user\_id идентификатор пользователя (внешний ключ к сущности Users).

16 Reviews (отзыв)

– movie\_id – идентификатор фильма (внешний ключ к сущности Movies).

– rating – оценка фильма.

– comment – текстовое описание оценки.

– created\_at – время создания отзыва.

– updated\_at – время изменения отзыва.

17 Payments(платежи)

– reservation\_id идентификатор брони (внешний ключ к сущности Reservations).

– payment\_method\_id идентификатор способа оплаты (внешний ключ к сущности Payment\_Methods).

– created\_at – время создания платежа.

– amount – сумма платежа.

18 Payment\_Methods (способы оплаты)

– name – название способа оплаты.

19 Advertisements (Реклама)

– title – название рекламы.

– description – описание рекламы,

– url – ссылка на источник,

– type\_id идентификатор типа рекламы (внешний ключ к сущности Adv\_Type).

20 Adv\_Type (Тип рекламы)

– name – название типа рекламы.

– priority – рекламный приоритет.

21 Ad\_Views (Просмотр рекламы)

– ad\_id – идентификатор сущности рекламы (внешний ключ к сущности Advertisments)

– view\_id – идентификатор просмотра онлайн фильма (внешний ключ к сущности Halls)

– viewed\_at – время просмотра

22 countries (страны)

– name – название страны.

23 languages (языки)

– name – название языка.

24 age\_restrictions

– age\_restriction – тип возрастного ограничения.

25 seat\_types (типы сидений)

– name – название типа.

26 movie\_actor (фильмы и актеры)

– movie\_id идентификатор фильма (внешний ключ к сущности Movies).

– actor\_id идентификатор актера (внешний ключ к сущности Actors).

27 movie\_genre (фильмы и жанры)

– movie\_id идентификатор фильма (внешний ключ к сущности Halls).

– genre\_id идентификатор жанра (внешний ключ к сущности Genre)

28 Playlist (плейлист)

– user\_id – идентификатор пользователя (внешний ключ к сущности Users).

– name – название плейлиста.

29 playlist\_movies (плейлисты и фильмы)

– movie\_id идентификатор фильма (внешний ключ к сущности Movies)

– playlist\_id идентификатор фильма (внешний ключ к сущности Playlist)

– created\_at время создания записи в плейлист.

30 Recommendation (рекомендация фильма)

– user\_id идентификатор пользователя (внешний ключ к сущности Users)

– movie\_id идентификатор фильма (внешний ключ к сущности Movies)

– created\_at" DATETIME

Собственно, 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.

Каждая из этих частей имеет свои особенности и требования, которые необходимо учесть при проектировании базы данных. Например, сервис просмотра кинофильмов должен хранить информацию о жанрах, режиссерах, актерах, рейтингах и отзывах о фильмах, а также предоставлять возможность поиска и фильтрации по различным критериям.

Сервис для продажи билетов в кинотеатр должен учитывать расписание сеансов, количество и стоимость мест, способы оплаты и доставки билетов, а также обеспечивать безопасность и надежность транзакций.

Рекламный агреггатор для онлайн-кинотеатра должен анализировать поведение и предпочтения пользователей, чтобы показывать им наиболее подходящие и эффективные рекламные объявления, а также отслеживать их взаимодействие с рекламой и результаты конверсии. .

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

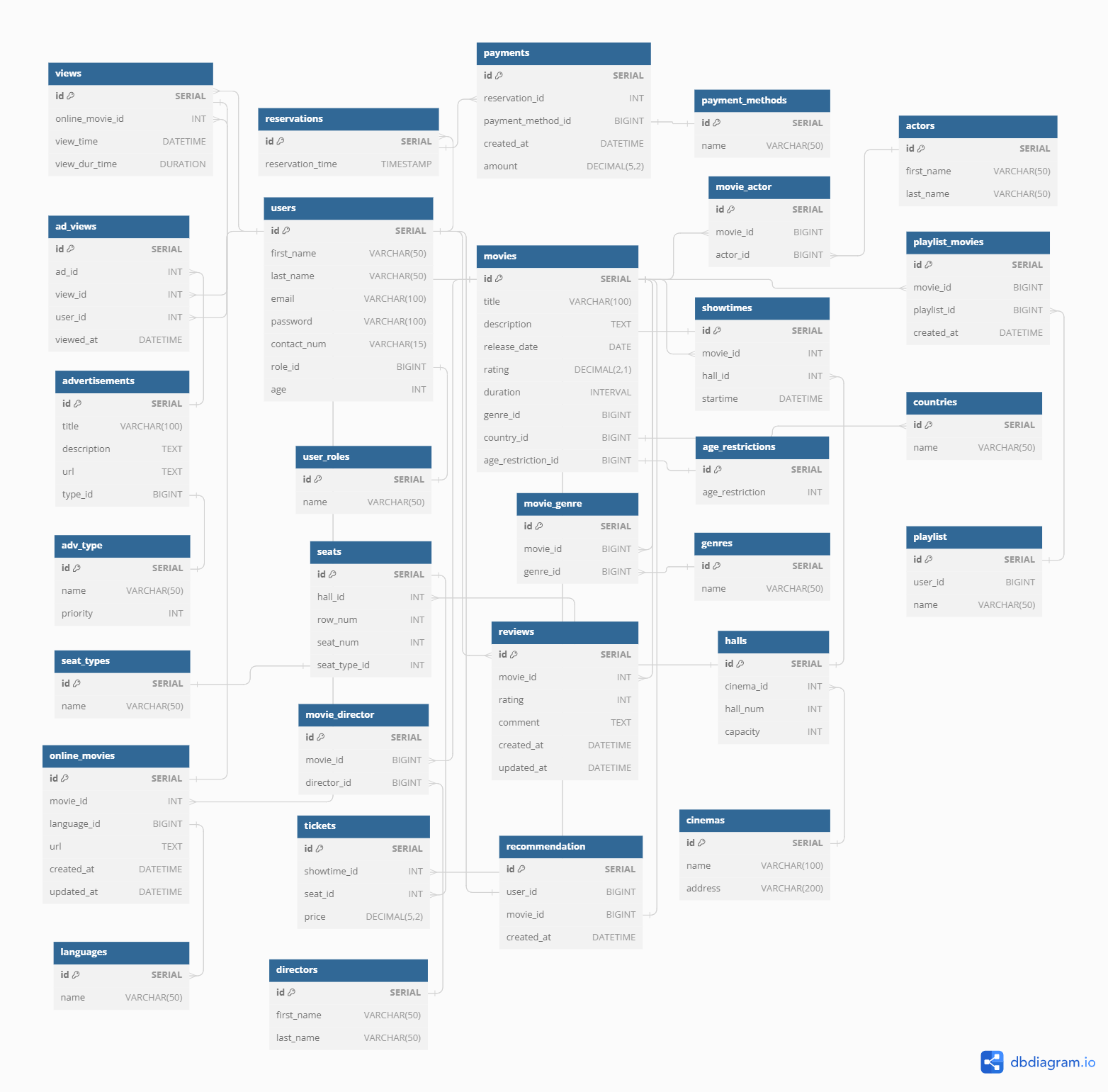


Рисунок 3.2 – ER-диаграмма базы данных

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.

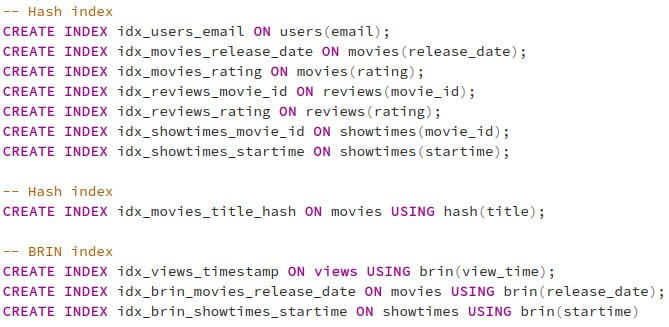


Рисунок 4.1 – Набор индексов, используемых в проектируемой базе данных

## 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:

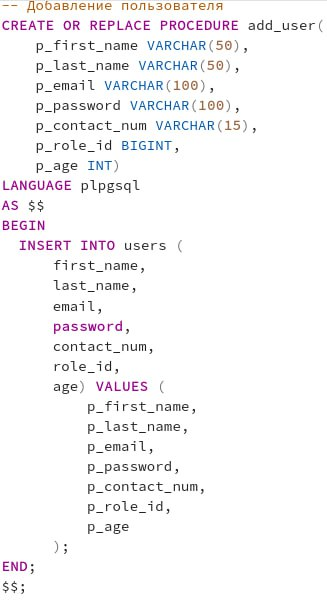


Рисунок 4.3 – Процедура для добавления нового пользователя

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

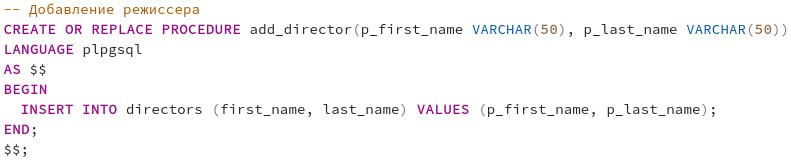


Рисунок 4.4 – Процедура для добавления нового директора

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:

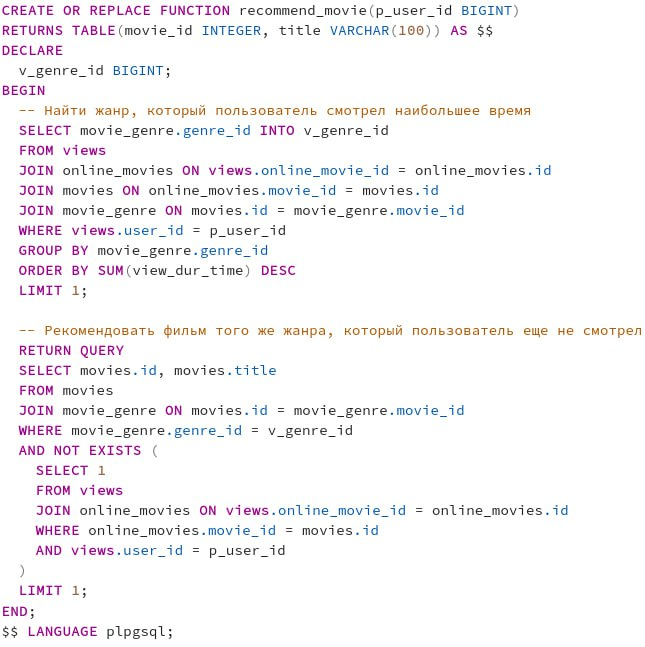


Рисунок 4.5 – Процедура рекомендации фильма по просмотрам

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:

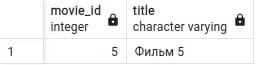


Рисунок 4.8 – Результат работы процедуры

Помимо этой рекомендательной процедуры имеется процедура, которая рекомендует пользователю фильм по жанру, который наберет наибольшее количество среди всех пользовательских плейлистов. Реализация данной процедуры представлена на рисунке 4.9:

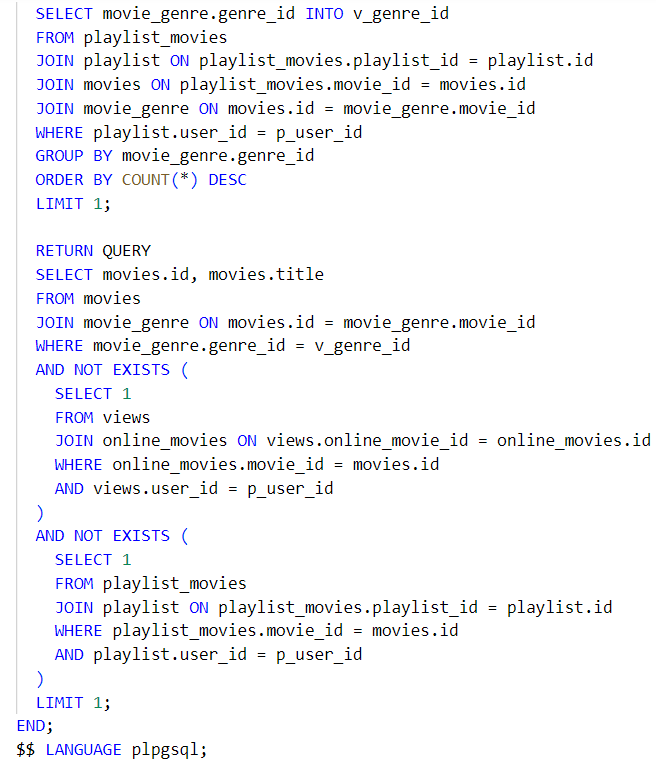


Рисунок 4.9 – Реализация процедуры рекомендации кинофильма по жанрам в плейлистах

В тестировании разработанная процедура отрабатывает строго по заданному поведению, что может показать рисунок 4.10:

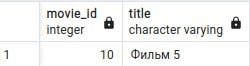


Рисунок 4.10 – Результат работы процедуры

## 4.3 Реализация триггеров

В разрабатываемой базе данных в первую очередь необходимо добавить триггеры для расчета средней оценки фильма, в зависимости от того поставили или удалили оценку. Вариант реализации такого триггера представлен на рисунке 4.11:



Рисунок 4.11 – Реализация триггера для расчета средней оценки фильма

Результаты тестирования данного триггера следующие. Пусть имеется фильм с начальным рейтингом, равным нулю. Пример представлен на рисунке 4.12:

C:\Users\joe\Downloads\Telegram Desktop\image_2023-12-15_05-01-55.png

Рисунок 4.12 – Тестовое значение фильма

Далее выполним 2 тестовые вставки значений отзыва с рейтингами 4 и 5 соответственно. Пример таких вставок представлен на рисунке 4.13:

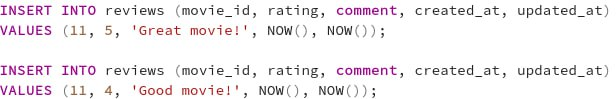


Рисунок 4.13 – Вставка тестовых значений для триггера

Результатом работы триггера является рисунок 4.14, который демонстрирует корректную работоспособность разработанного алгоритма:

C:\Users\joe\Downloads\Telegram Desktop\image_2023-12-15_05-08-04.png

Рисунок 4.14 – Результат работы триггера

Вдобавок был реализован триггер, который создает просмотр рекламы пользователем, в случае, если добавляется запись в таблицу Views.

Реализация данного триггера представлена на рисунке 4.15:



Рисунок 4.15 – Реализация процедуры генерации просмотра рекламы

# 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: 15.12.2023.

[2] Relax.by [Электронный ресурс]:Relax. Режим доступа: https://relax.by. Дата доступа: 15.12.2023.

[3] Okko.tv [Электронный ресурс]:Okko. Режим доступа: https://okko.tv. Дата доступа: 15.12.2023.

[4] Kinopoisk.ru[Электронный ресурс]:Kinopoisk. Режим доступа: https://kinopoisk.ru. Дата доступа: 15.12.2023.

[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: 15.12.2023.

[8] Как создать базу данных для бронирования мест [Электронный ресурс]: SQLPro. – Режим доступа: https://sqlpro.ru/create-database-for-booking. Дата доступа: 15.12.2023.

[9] Базы данных в киноиндустрии [Электронный ресурс]: DataArt. Режим доступа: https://dataart.ru/database-in-cinema-industry/. Дата доступа: 15.12.2023.

[10] Применение баз данных в кинотеатрах [Электронный ресурс]: CinemaDB. – Режим доступа: https://cinemadb.ru/. Дата доступа: 15.12.2023.

[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: 15.12.2023.

[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: 15.12.2023

[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: 15.12.2023.

[16] Базы данных для бронирования мест в кинотеатрах [Электронный ресурс]: DBPro. – Режим доступа: https://dbpro.ru/booking-database-for-cinemas. Дата доступа: 15.12.2023.

[17] Базы данных в киноиндустрии: новые тенденции [Электронный ресурс]: CinemaTrends. Режим доступа: https://cinematrends.ru/database-trends-in-cinema-industry/. Дата доступа: 15.12.2023.

[18] Применение баз данных в кинотеатрах: кейс-стади [Электронный ресурс]: CinemaCase. – Режим доступа: https://cinemacase.ru/. Дата доступа: 15.12.2023.

[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: 15.12.2023

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,

<<<<<<< Updated upstream

"priority" INT

=======

"priority" BIGINT

>>>>>>> Stashed changes

);

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,

<<<<<<< Updated upstream

"movie\_id" BIGINT,

"created\_at" DATETIME

);

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

=======

"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");

>>>>>>> Stashed changes

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