Министерство цифрового развития, связи и массовых коммуникаций

Российской Федерации

Ордена Трудового Красного Знамени

Федеральное государственное образовательное бюджетное учреждение

высшего образования

«МОСКОВСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

СВЯЗИ И ИНФОРМАТИКИ»

Факультет Информационных технологий

Кафедра «Интеллектуальные системы в управлении и автоматизации»

Курсовая работа по дисциплине

«Базы данных»

На тему

«Разработка БД для АСУ “Кинотеатр”»

Выполнил студент группы БЭИ2101 Денисова П. П.

Проверил доцент кафедры «Интеллектуальные системы в управлении и автоматизации» Воронов В. И.

Москва, 2023

**СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ** 3](#_Toc135702641)

[**ГЛАВА 1. СИСТЕМНЫЙ АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ** 6](#_Toc135702642)

[**1.1** **Анализ объекта автоматизации «Кинотеатр»** 6](#_Toc135702643)

[**1.2** **Обзор информационных технологий** 7](#_Toc135702644)

[**1.3** **Обзор продуктов-аналогов** 10](#_Toc135702645)

[**1.4** **Требования к разрабатываемой базе данных** 11](#_Toc135702646)

[**Выводы** 12](#_Toc135702647)

[**ГЛАВА 2. ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ** 14](#_Toc135702648)

[**2.1 Разработка инфологической модели** 14](#_Toc135702649)

[**2.2 Обоснование выбора модели данных** 18](#_Toc135702650)

[**2.3 Даталогическое проектирование** 19](#_Toc135702651)

[**2.4** **Нормализация, схема БД** 23](#_Toc135702652)

[**Выводы** 25](#_Toc135702653)

[**ГЛАВА 3. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ** 26](#_Toc135702654)

[**3.1. Анализ и выбор СУБД** 26](#_Toc135702655)

[**3.2. Физическое проектирование базы данных в СУБД** 27](#_Toc135702656)

[**3.3. Разработка представлений, форм, отчетов** 34](#_Toc135702657)

[**3.4. Реализация ограничений, автоматизация обработки данных в БД** 48](#_Toc135702658)

[**3.5 Безопасность и контроль** 51](#_Toc135702659)

[**Выводы** 54](#_Toc135702660)

[**ЗАКЛЮЧЕНИЕ** 55](#_Toc135702661)

[**СПИСОК ИСТОЧНИКОВ И ЛИТЕРАТУРЫ** 57](#_Toc135702662)

[**ПРИЛОЖЕНИЯ** 59](#_Toc135702663)

# **ВВЕДЕНИЕ**

Базы данных используются в современном мире для эффективной организации, хранения и извлечения больших объемов данных. Они используются в различных отраслях, включая финансы, здравоохранение, образование, электронную коммерцию и многое другое. Базы данных помогают организациям управлять своими данными структурированным и организованным образом, что приводит к лучшему принятию решений и повышению операционной эффективности. Некоторые распространенные области применения баз данных включают управление взаимоотношениями с клиентами, управление запасами, отслеживание продаж, управление персоналом и ведение финансовой отчетности. С ростом использования технологий и ростом больших данных важность баз данных только возрастает.

Базой данных (БД) называют именованную совокупность данных, отражающую состояние объектов и их отношений в рассматриваемой предметной области.

Для создания и использования БД служат системы управления базами данных (СУБД), которые занимают особое место в мире программного обеспечения и нашей повседневной жизни. СУБД предназначены для создания, ведения и совместного использования БД многими пользователями. Как правило, работа этих систем осуществляется в локальных вычислительных сетях. Подавляющее большинство современных СУБД представляют собой системы реляционного типа, т.е. использующие реляционную модель данных. Данные в реляционных БД хранятся в таблицах – отношениях (relation). Реляционные СУБД (РСУБД) – это компьютеризованные системы хранения записей в табличном виде. Стандартным языком взаимодействия с реляционными БД является язык SQL-запросов, который реализуется в РСУБД на основе операций реляционной алгебры и реляционного исчисления.

Целью курсовой работы является разработка базы данных для автоматизированной системы управления "кинотеатр". Также целью данной работы является ознакомление с основными принципами построения, использования и оптимизации реляционных баз данных.

В данной курсовой работе поставлены следующие задачи:

• провести системный анализ предметной области кинотеатр;

• провести обзор информационных технологий, подходящих для разработки;

• изучить продукты-аналоги данной предметной области;

• определить требования, предъявляемые к разрабатываемой базе данных;

• разработать инфологическую модель базы данных;

• обосновать выбор модели данных

• осуществить даталогическое проектирование базы данных;

• нормализовать спроектированную модель и составить схему базы данных;

• осуществить реализацию БД на выбранной СУБД;

Актуальность данной темы объясняется желанием людей погрузиться в атмосферу просмотра фильмов и мультфильмов, а также насладиться просмотром шедевров киноиндустрии на больших экранах.

Поставленная задача будет решаться с помощью анализа предметной области, ее формализации с помощью функциональных зависимостей. Затем необходимо провести этапы минимизации системы функциональных зависимостей, описывающих предметную область, и на основании полученной редуцированной системы спроектировать требуемую модель базы данных.

В данной курсовой работе представлены следующие разделы: Глава 1, содержащая анализ предметной области, обзор информационных технологий и продуктов-аналогов, требования к БД; Глава 2, включающая в себя разработку инфологической модели, обоснования выбора модели данных, даталогическое проектирование, нормализацию и схему БД; Глава 3, заключающаяся в программной реализации БД; Выводы в конце каждой главы; Заключение по курсовой работе; Список источников и литературы; Приложения.

# **ГЛАВА 1. СИСТЕМНЫЙ АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**

Кинотеатр должен предоставлять клиентам быстрый и удобный способ приобретения билетов для последующего посещения киносеанса. База данных будет использоваться работниками для ведения учета посещений сеансов фильмов.

При работе с базой данных каждый посетитель может получить информацию о фильмах, показываемых в кинотеатре, свободных местах на определенный сеанс и ценах на билеты. Эта информация может быть полезна для людей, которые хотят узнать больше о фильме перед его просмотром, или для тех, кто ищет фильм определенного типа для просмотра.

## **Анализ объекта автоматизации «Кинотеатр»**

Область кинотеатра представляет собой сложную систему, в которой задействовано множество действующих лиц и процессов. Основная задача кинотеатра — развлекать своих посетителей, показывая фильмы на большом экране. Посетители приходят в кинотеатр, чтобы насладиться фильмами и хорошо провести время со своими друзьями и семьей.

Клиенты могут получить информацию о показываемых фильмах, прочесть их краткое описание. Помимо этого, им предоставляется возможность узнать цену билета на конкретный фильм, а также забронировать время и место.

«Кинотеатр» предлагает своим посетителям широкий выбор фильмов и развлечений. «Кинотеатр» обслуживает разнообразную аудиторию и предлагает широкий выбор жанров кино, включая боевики, приключения, комедии, ужасы, драмы и анимацию. Также предусмотрены различные скидки и акции, такие как скидки для студентов и программы лояльности, чтобы сделать поход в кино более доступным и доступным для всех. В целом их предметная область — индустрия развлечений, в частности, сектор кинотеатров в России.

Клиенты могут посмотреть следующую информацию:

• Фильмы (название, описание, страна производства, дата выпуска, жанр, длительность);

• Места и ряды (зал, ряд, место, категория, возможность бронирования);

• Сеансы (название фильма, зал, дата, время);

• Цены (цена билета, время, категория).

## **Обзор информационных технологий**

Система управления базами данных (СУБД) – совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями.

Программное обеспечение СУБД обеспечивает логический и структурированный подход к управлению данными, позволяя пользователям получать доступ к данным и манипулировать ими различными способами.

Существует несколько типов систем управления базами данных, в том числе:

1. Иерархическая— это тип системы управления базами данных, которая организует данные в древовидную структуру. В системе этого типа данные организованы иерархически, при этом каждая запись имеет родительскую запись и одну или несколько дочерних записей. Родительская запись находится на вершине иерархии, а каждая дочерняя запись — ниже нее. Дочерние записи могут иметь свои собственные дочерние записи.
2. Сетевая — это тип системы управления базами данных, которая хранит данные в сетевой модели. В системе этого типа данные организованы в виде древовидной структуры с узлами, представляющими записи, и ветвями, представляющими отношения между записями. Каждая запись может иметь несколько родительских и дочерних записей. Этот тип системы очень гибок и эффективен для обработки сложных взаимосвязей данных.
3. Реляционная — это программная система, используемая для создания, обслуживания и управления реляционными базами данных. В реляционной базе данных данные организованы в виде таблиц, содержащих строки и столбцы. РСУБД используют SQL (язык структурированных запросов) для обработки и запроса данных, хранящихся в таблицах.
4. Объектно-ориентированная — это тип системы управления базами данных, в которой данные хранятся в виде объектов, подобно объектно-ориентированным языкам программирования. Эти объекты могут содержать как данные, так и методы, работающие с этими данными, что позволяет создавать более сложные и гибкие структуры данных.
5. Объектно-реляционная — это тип системы управления базами данных (СУБД), который сочетает в себе возможности реляционных баз данных с гибкостью объектно-ориентированного программирования. Он может хранить как структурированные данные (с использованием реляционных таблиц), так и неструктурированные данные (с использованием концепций объектно-ориентированного программирования).

Инструменты автоматизированной разработки программного обеспечения (CASE) — это программные приложения, поддерживающие разработку высококачественных программных систем. Они автоматизируют жизненный цикл разработки программного обеспечения, от сбора требований до проектирования, кодирования, тестирования и обслуживания.

* Microsoft Access — это система управления реляционными базами данных, разработанная Microsoft. Это позволяет пользователям создавать и управлять базами данных, которые представляют собой наборы данных, организованных в таблицы, с установленными отношениями между таблицами. Access предоставляет пользователям удобный графический интерфейс для проектирования и создания баз данных, а также включает в себя различные инструменты для запросов, анализа и составления отчетов по данным в базе данных. Он также позволяет создавать формы и отчеты для упрощения ввода и анализа данных.
* MySQL — это система управления реляционными базами данных с открытым исходным кодом. Это одна из самых популярных систем управления базами данных, используемых сегодня, особенно для веб-приложений. MySQL обладает широкими возможностями настройки и может работать на различных платформах, включая Windows, Linux и UNIX. С помощью MySQL пользователи могут создавать и управлять базами данных, таблицами и данными. Он поддерживает широкий спектр типов данных, включая текстовые, числовые значения и значения даты/времени. MySQL также включает полный набор инструментов для управления и оптимизации производительности базы данных.
* Oracle Database — система управления реляционными базами данных, разработанная корпорацией Oracle. Это одна из самых популярных и широко используемых систем управления базами данных в мире. База данных Oracle предназначена для хранения и управления большими объемами данных и часто используется предприятиями и организациями всех размеров. Некоторые ключевые функции Oracle Database включают высокую доступность, масштабируемость, безопасность и производительность. Он поддерживает несколько платформ и операционных систем, и его можно настроить в соответствии с конкретными потребностями конкретной организации. Он также включает в себя множество инструментов и функций для администрирования баз данных, резервного копирования и восстановления, а также анализа данных.
* PostgreSQL — это мощная система управления реляционными базами данных с открытым исходным кодом, которая используется для хранения, организации и управления большими объемами данных. Он известен своей надежностью, масштабируемостью и расширяемостью, а также способностью обрабатывать сложные запросы и транзакции. PostgreSQL основан на языке SQL, но также предлагает поддержку расширенных функций, таких как JSON, XML и массивы. Он также предоставляет широкий спектр встроенных функций для манипулирования данными, а также возможность создавать собственные функции и расширения.

## **Обзор продуктов-аналогов**

В настоящее время на рынке информационных систем позиционируются продукты, имеющие аналогичные с разрабатываемой ИС цели объекты автоматизации.

Для обеспечения конкурентоспособности требуется ознакомится с возможностями конкурентов, узнать их слабые и сильные стороны.

* UCS

Компания UCS предлагает одно- и двухзальным кинотеатрам, открывающимся по программе поддержки Фонда кино, программное обеспечение «Премьера-лайт». «Премьера-лайт» включает в себя автоматизированную систему продажи билетов, модуль «Управляющий», модули отчетности в системы comScore/Rentrak и ЕАИС. Для подключения «Премьеры-лайт» достаточно одного компьютера, при этом сервер не является обязательным компонентом системы. В рамках подключения данного ПО компания UCS также берет на себя дистанционную пуско-наладку и дистанционное обучение.

* Ticket Soft

Компания Ticket Soft предлагает субсидированным кинотеатрам как базовый комплект ПО и оборудования, наиболее подходящий для муниципальных площадок, так и более расширенный функционал. Комплексное предложение включает в себя как программное обеспечение для автоматизации рабочих мест кассира, администратора (менеджера, управляющего) и отправки данных в ЕАИС, так оборудование: профессиональный билетный принтер и мощный компьютер с дополнительным монитором посетителя. Установка системы и обучение персонала тоже входят в предложение и осуществляются специалистами компании по уже многократно отработанной схеме. Бонусом идет стартовый тираж билетов и бесплатное подключение к сервису «КиноХод» для реализации билетов на сайте кинотеатра.

* «Киноплан»

Компания «Киноплан» специально для кинотеатров, поддержанных Фондом кино, предлагает комплект оборудования и программного обеспечения, необходимого для быстрого запуска работы и эффективной оптимизации процесса продаж (рабочие места кассира и администратора, продажа билетов через Интернет, отчетность в Excel, отправка показаний в ЕАИС и comScore/Rentrak, 3-годичная круглосуточная поддержка и пр.). Далее следует определить функциональные возможности системы. Предполагается, что проектируемый продукт будет иметь общественно-социальную значимость в первую очередь для пользователей, заинтересованных в получении информации о кинотеатре, фильмах, сеансах и покупке билетов. Реляционная система «Кинотеатр» создается с целью облегчения работы по поиску и получении информации клиентами кинотеатра.

Теперь необходимо описать возможности, предоставленные клиенту «Кинотеатра»:

• информация об интересующем фильме и дате сеанса;

• просмотр ассортимента кинотеатра, информации о нужном сеансе, месте в зале, цене билета;

• бронирование мест на определенные дату и время.

## **Требования к разрабатываемой базе данных**

Данная БД создается для удобства клиентов в выборе фильма и бронировании билетов. Также с базой данных могут работать кассиры и администратор.

При работе с БД клиент может посмотреть информацию о:

* Фильмах;
* Актерах;
* Режиссерах;
* Табеле заполнения;
* Расписании сеансов;
* Цене билетов.
* Льготах
* Скидках

Кассир может просматривать все таблицы и заполнять следующие:

* Билеты;
* Табель заполнения;

Администратор работает с:

* Фильмами;
* Актерами;
* Режиссерами;
* Расписанием сеансов;
* Залами;
* Ценами;
* Скидками;
* Льготами;
* Билетами;
* Табелем заполнения

Для данной базы данных требуется предусмотреть следующие ограничения:

1. Нельзя в одном зале одновременно показывать более 1 фильма;
2. Нельзя бронировать или покупать билет на купленное место;
3. Величина скидки не может быть больше 100%;
4. Величина снижения цены по льготам не может быть больше 100%.

## **Выводы**

В данной главе был проведен анализ предметной области, выявлена необходимость создания базы данных кинотеатра и приведены примеры существующих комплексов программ для управления кинотеатром. Также были описаны основные информационные технологии, с помощью которых можно создать СУБД. Помимо этого, были сформулированы требования к создаваемой базе данных.

# **ГЛАВА 2. ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ**

В данной главе разработаем инфологическую модель базы данных кинотеатра. Проанализируем существующие даталогические модели данных и обоснуем выбор реляционной модели. На основе построенной инфологической модели проведем логическое проектирование базы данных, опишем каждую сущность и построим реляционную модель базы данных кинотеатра.

## **2.1 Разработка инфологической модели**

Описывается разработка инфологической модели (ER-модели). Инфологическая модель должна содержать не менее 10 сущностей и одну-две связи «многие ко многим».

Целью инфологического проектирования является создание структурированной информационной модели предметной области, для которой будет разрабатываться база данных.

Инфологическая модель должна удовлетворять следующим критериям:

1. Точность. Модель должна точно представлять реальную систему или моделируемый процесс.

2. Полнота: модель должна включать все соответствующие объекты и отношения.

3. Последовательность: Модель должна быть свободна от противоречий или конфликтов.

4. Ясность. Модель должна быть проста для понимания и интерпретации.

5. Релевантность. Модель должна соответствовать решаемой проблеме.

6. Гибкость. Модель должна быть способна приспосабливаться к изменениям и обновлениям.

7. Масштабируемость. Модель должна быть способна обрабатывать большие объемы данных.

8. Ремонтопригодность. Модель должна быть проста в обслуживании и обновлении.

9. Достоверность. Модель должна основываться на точных и релевантных данных.

10. Удобство использования. Модель должна быть удобной для пользователя и доступной для всех заинтересованных сторон, которым необходимо ее использовать.

На основе проведенного системного анализа предметной области выделены следующие сущности:

1. Билеты

* № билета
* Фильм
* Цена
* Сеанс
* Место
* Льготная категория
* Скидка

1. Фильмы:

* Название
* Описание
* Страна производства
* Дата выпуска
* Жанр
* Актеры
* Режиссер
* Длительность

1. Актеры:

* Имя
* Фамилия
* Год рождения
* Фильмы(снимались)

1. Режиссеры:

* Имя
* Фамилия
* Фильмы(снимались)

5. Табель заполнения:

* Зал
* Ряд
* Место
* Статус

6. Расписание сеансов:

* Фильм
* Зал
* Дата
* Время

7. Льготные категории:

* Наименование
* Цена

8. Скидки:

* Наименование скидки
* Сумма

9. Цены:

* Цена
* Время
* День недели

10. Залы

* Зал
* Кол-во мест

Связи между сущностями:

* Билеты – Фильмы (один ко многим)
* Фильмы – Актеры (многие ко многим)
* Фильмы – Режиссеры (многие ко многим)
* Билеты – Расписание сеансов (один ко многим)
* Расписание сеансов – Залы (один ко многим)
* Билеты – Цены (один ко многим)
* Билеты – Льготные категории (один ко многим)
* Билеты – Скидки (один ко многим)
* Билеты – Табель заполнения (многие ко многим)
* Табель заполнения – Залы (один ко многим)

Инфологическая модель представлена на рисунке 2.1.



Рисунок 2.1 – Инфологическая модель

## **2.2 Обоснование выбора модели данных**

Модель данных — это концептуальное представление объектов данных, их отношений друг к другу и правил, регулирующих эти отношения. Это помогает организовать и структурировать данные таким образом, чтобы их было легко понять и использовать. Модели данных можно использовать для проектирования баз данных, программных приложений и других информационных систем. Существуют различные типы моделей данных, такие как реляционные, иерархические, сетевые, объектно-ориентированные и объектно-реляционные модели данных. Эти модели помогают разработчикам создать схему базы данных, которая используется для определения, хранения, обработки и извлечения данных.

В данной курсовой работе я выбрала реляционную модель данных. Она широко используется в системах управления базами данных благодаря своим многочисленным преимуществам. Вот некоторые из ключевых преимуществ реляционной модели данных:

1. Простота понимания. Реляционную модель данных легко понять, поскольку она основана на простой концепции таблиц данных со строками и столбцами. Это упрощает для разработчиков проектирование, создание и управление базами данных.

2. Целостность данных. Реляционная модель данных обеспечивает целостность данных, что означает, что данные, вводимые в базу данных, должны соответствовать определенным правилам или ограничениям. Это помогает гарантировать, что в базе данных хранятся только действительные данные.

3. Гибкость. Реляционная модель данных очень гибкая, что позволяет разработчикам добавлять, изменять и удалять данные по мере необходимости. Это упрощает адаптацию к изменяющимся требованиям бизнеса.

4. Масштабируемость. Реляционная модель данных является масштабируемой, что означает, что она может обрабатывать большие объемы данных и может расширяться по мере необходимости.

5. Безопасность. Реляционная модель данных обеспечивает надежные механизмы безопасности, которые позволяют создавать учетные записи пользователей с различными уровнями доступа к базе данных. Это помогает предотвратить несанкционированный доступ и защитить конфиденциальные данные.

6. Запросы. Реляционная модель данных поддерживает сложные запросы, что позволяет разработчикам извлекать данные из нескольких таблиц с использованием различных критериев и условий.

В целом, реляционная модель данных является популярным выбором для систем управления базами данных из-за ее простоты, гибкости, масштабируемости, целостности данных, безопасности и возможностей запросов.

## **2.3 Даталогическое проектирование**

В реляционных базах данных (РБД) даталогическое проектирование приводит к разработке схемы БД, т.е. совокупности схем отношений, адекватно моделирующих объекты ПО и семантических связей между ними.

Вот общие шаги для перехода от информационной модели к реляционной модели данных:

1. Определите сущности и отношения. Просмотрите информационную модель и определите все сущности и отношения, относящиеся к предметной области.

2. Определите первичные ключи. Для каждого объекта определите первичный ключ, который однозначно идентифицирует каждый экземпляр этого объекта.

3. Создайте таблицы: создайте таблицу для каждого объекта. Таблица должна включать столбцы для каждого атрибута объекта и столбец для первичного ключа.

4. Установите отношения: создайте столбцы внешнего ключа в таблицах, которые представляют объекты, имеющие отношения с другими таблицами. Эти столбцы должны ссылаться на первичный ключ связанной таблицы.

5. Нормализуйте таблицы: примените правила нормализации, чтобы убедиться, что каждая таблица находится в наиболее эффективной и логичной форме.

6. Определите ограничения. Определите любые ограничения, необходимые для поддержания целостности данных, такие как ограничения уникальности или ограничения ссылочной целостности.

7. Напишите правила журнала данных. Напишите правила журнала данных, чтобы выразить отношения между объектами и запросами, которые будут использоваться для извлечения информации из базы данных.

8. Протестируйте базу данных. Протестируйте базу данных, чтобы убедиться, что она работает правильно и возвращает ожидаемые результаты.

В целом процесс включает преобразование информационной модели в набор таблиц, организованных в соответствии с реляционной моделью, а затем использование журнала данных для выражения взаимосвязей между таблицами.

Далее опишем каждую сущность даталогической модели.

1. Билеты

* № билета – INTEGER NOT NULL PK
* Фильм – INTEGER NOT NULL FK
* Цена – INTEGER NOT NULL FK
* Сеанс – INTEGER NOT NULL FK
* Место – INTEGER NOT NULL FK
* Льготная категория – VARCHAR(25) NOT NULL FK
* Скидка – VARCHAR(25) NOT NULL FK

1. Фильмы:

* ID фильма – INT NOT NULL PK
* Название – VARCHAR(50) NOT NULL
* Описание – VARCHAR(1500) NOT NULL
* Страна производства – VARCHAR(15) NOT NULL
* Дата выпуска – DATE NOT NULL
* Жанр – VARCHAR(15) NOT NULL
* Актеры – INT NOT NULL FK
* Режиссеры – INT NOT NULL FK
* Длительность – VARCHAR(9) NOT NULL

1. Актеры:

* ID актера – INT NOT NULL PK
* Имя – VARCHAR(15) NOT NULL
* Фамилия – VARCHAR(20) NOT NULL
* Год рождения – DATE NOT NULL
* Фильмы(снимались) – VARCHAR(50) NOT NULL

1. Режиссеры:

* ID режиссера – INT NOT NULL
* Имя – VARCHAR(15) NOT NULL
* Фамилия – VARCHAR(20) NOT NULL
* Фильмы(снимали) – VARCHAR(50) NOT NULL

5. Табель заполнения:

* ID места – INT NOT NULL PK
* Зал – INT NOT NULL FK
* Ряд – INT NOT NULL
* Место – INT NOT NULL
* Статус – VARCHAR(15) NOT NULL

6. Расписание сеансов:

* ID сеанса – INT NOT NULL PK
* Фильм – VARCHAR(50) NOT NULL
* Зал – INT NOT NULL FK
* Дата – DATE NOT NULL
* Время – VARCHAR(5) NOT NULL

7. Льготные категории:

* Наименование – VARCHAR(25) NOT NULL PK
* Цена – INT NOT NULL

8. Скидки:

* Наименование скидки – VARCHAR(25) NOT NULL PK
* Сумма – INT NOT NULL

9. Цены:

* ID цены – INT NOT NULL PK
* Время – VARCHAR(5) NOT NULL
* День недели – VARCHAR(11) NOT NULL
* Цена – INT NOT NULL

10. Залы

* Зал – INT NOT NULL PK
* Кол-во мест – INT NOT NULL

Даталогическая модель представлена на рисунке 2.2.

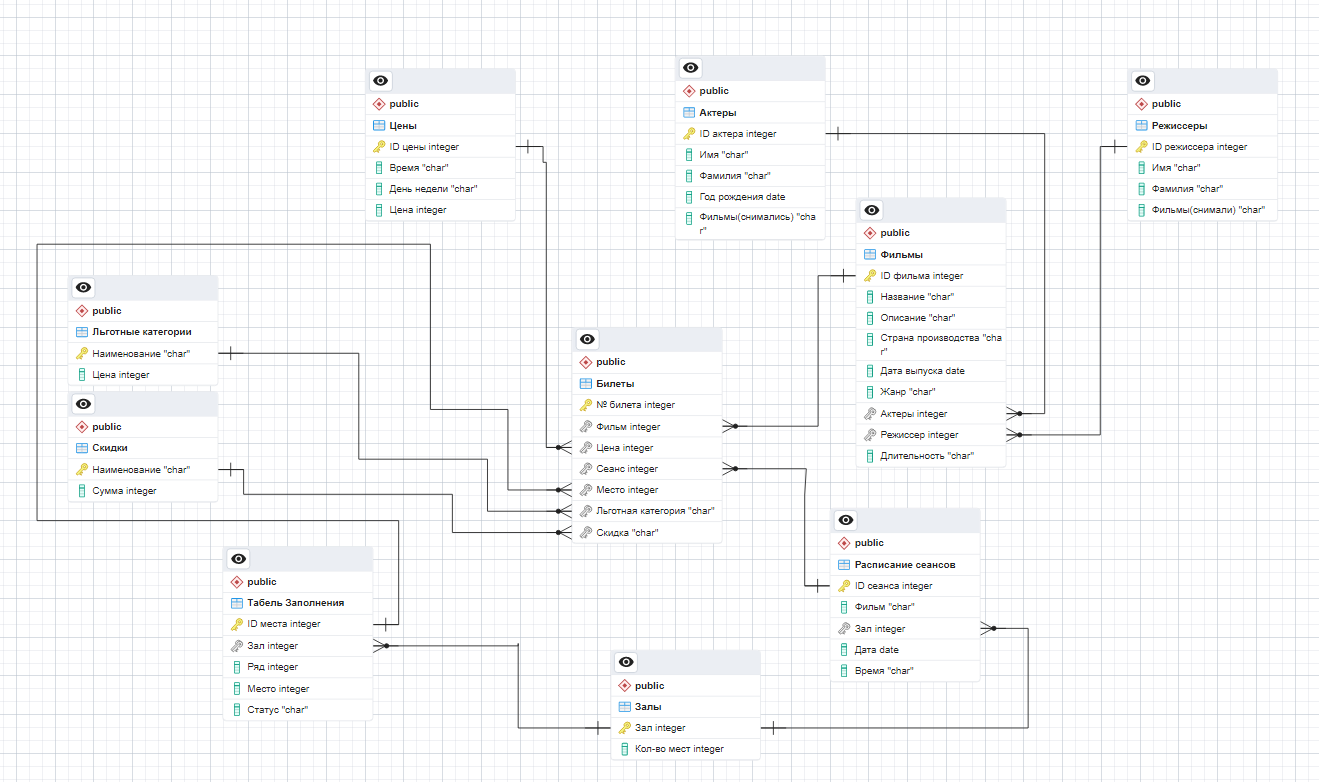


Рисунок 2.2 – Даталогическая модель

## **Нормализация, схема БД**

Нормальные формы базы данных — это набор правил, которые помогают обеспечить организацию и легкий доступ к данным в реляционной базе данных.

Каждой нормальной форме соответствует некоторый определенный набор ограничений, и отношение находится в некоторой нормальной форме, если удовлетворяет свойственному ей набору ограничений.

В теории реляционных БД обычно выделяется следующая последовательность нормальных форм:

* первая нормальная форма (1NF);
* вторая нормальная форма (2NF);
* третья нормальная форма (3NF);
* нормальная форма Бойса— Кодда (BCNF);
* четвертая нормальная форма (4NF);
* пятая нормальная форма, или форма проекции-соединения (5NF или PJNF).

Основные свойства нормальных форм:

* каждая следующая нормальная форма в некотором смысле улучшает свойства предыдущей;
* при переходе к следующей нормальной форме свойства предыдущих нормальных форм сохраняются.

Функциональной зависимостью набора атрибутов В отношения R от набора атрибутов А того же отношения, обозначаемой как

R.A -> R.B или А -> В

называется такое соотношение проекций R[A] и R[B], при котором в каждый момент времени любому элементу проекции R[A] соответствует только один элемент проекции R[B] , входящий вместе с ним в какой-либо кортеж отношения R.

Функциональные зависимости определяют не текущее состояние БД, а все возможные ее состояния, то есть они отражают те связи между атрибутами, которые присущи реальному объекту, который моделируется с помощью БД.

Отношение находится в первой нормальной форме тогда и только тогда, когда на пересечении каждого столбца и каждой строки находятся только элементарные значения атрибутов.

Первая нормальная форма (1NF) является основным требованием при проектировании базы данных и гарантирует, что данные в каждом столбце таблицы являются атомарными, то есть содержат только одно значение.

Чтобы соответствовать требованиям первой нормальной формы, таблица должна соответствовать следующим критериям:

1. Каждый столбец содержит атомарные (неделимые) значения данных, то есть каждое поле должно содержать только одно значение.

2. Значения в столбце должны быть одного типа.

3. Каждый столбец должен иметь уникальное имя, описывающее содержащиеся в нем данные.

4. Порядок строк и столбцов не должен иметь значения.

Приведенная база данных удовлетворяет требованиям первой нормальной формы.

Отношение находится во второй нормальной форме тогда и только тогда, когда оно находится в первой нормальной форме и не содержит неполных функциональных зависимостей непервичных атрибутов от атрибутов первичного ключа.

Функциональная зависимость R.A -> R.B называется полной, если набор атрибутов В функционально зависит от А и не зависит функционально от любого подмножества А, то есть R.A -> R.B называется полной, если:

любое А1 с А=> R.A -/-> R.B, что читается следующим образом:

для любого А1, являющегося подмножеством A, R.B функционально не зависит от R.A, в противном случае зависимость R.A -> R.B называется неполной.

Отношения приведенной базы данных находятся в 1НФ и не содержат неполных функциональных зависимостей, следовательно, они находится в 2НФ.

Отношение находится в третьей нормальной форме тогда и только тогда, когда оно находится во второй нормальной форме и не содержит транзитивных зависимостей.

Функциональная зависимость R.A -> R.B называется транзитивной, если существует набор атрибутов С такой, что:

1.С не является подмножеством А.

2.С не включает в себя В.

3.Существует функциональная зависимость R.A -> R.C.

4.Не существует функциональной зависимости R.C -> R.A.

5.Существует функциональная зависимость R.C -> R.B.

Это отношение находится в третьей нормальной форме, потому что неполных функциональных зависимостей непервичных атрибутов от атрибутов возможного ключа здесь не присутствует и нет транзитивных зависимостей.

3НФ является достаточной для реализации базы данных, т.е. нормализацию проводить не нужно.

## **Выводы**

Во второй главе была разработана и обоснована инфологическая модель базы данных кинотеатра. Помимо этого, было проведено даталогическое проектирование, нормализация и создание схемы базы данных.

# **ГЛАВА 3. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ**

В данной главе проведем анализ и выбор СУБД. База данных кинотеатра «Кинотеатр» содержит 10 таблиц, которые будут спроектированы в разделе Физическое проектирование БД. Будут разработаны представления, формы и отчеты, а также написаны 3 триггера и показана их реализация.

## **3.1. Анализ и выбор СУБД**

В качестве СУБД я выбрала PostgreSQL. PostgreSQL — это свободная система управления реляционными базами данных (RDBMS, или СУРБД) с открытым исходным кодом. Она поддерживает такие вещи как транзакции, схемы и внешние ключи, и часто позиционируется как более строго соответствующая стандартам SQL и более безопасная по умолчанию, чем любая другая база данных, коммерческая или нет.

Он обладает следующими преимуществами:

• Высокая производительность: PostgreSQL предназначен для эффективной обработки больших объемов данных и сложных запросов с минимальной задержкой.

• Открытый исходный код: PostgreSQL — это программное обеспечение с открытым исходным кодом, что означает, что его можно использовать бесплатно и можно настроить для удовлетворения конкретных потребностей бизнеса без каких-либо лицензионных сборов.

• Масштабируемость: PostgreSQL может масштабироваться для обработки больших объемов данных, что делает его идеальным для предприятий, которым требуется высокая степень гибкости и производительности.

• Соответствие ACID: PostgreSQL поддерживает свойства ACID (атомарность, согласованность, изоляция и долговечность), которые обеспечивают надежную и безопасную обработку транзакций.

• Расширяемость: PostgreSQL предлагает широкий спектр расширений и подключаемых модулей, которые можно использовать для расширения его функциональности и поддержки различных типов данных.

• Сильная поддержка сообщества: PostgreSQL имеет сильное сообщество разработчиков и пользователей, которые вносят свой вклад в его развитие и предлагают пользователям поддержку и рекомендации.

• Безопасность: PostgreSQL имеет ряд встроенных функций безопасности, включая SSL-шифрование, аутентификацию пользователей и контроль доступа, которые помогают защитить данные и обеспечить соответствие нормативным требованиям.

## **3.2. Физическое проектирование базы данных в СУБД**

На основе реляционной модели произведена программная реализация. База данных содержит 10 таблиц:

• Билеты

• Фильмы

• Актеры

• Режиссеры

• Табель заполнения

• Расписание сеансов

• Льготные категории

• Скидки

• Цены

• Залы

Далее рассмотрим реализацию базы данных в PostgreSQL.

1) Залы

SQL-запрос создания таблицы:

CREATE TABLE halls (

hall integer PRIMARY KEY,

places\_count integer NOT NULL

);

Таблица «Залы» представлена на рисунке 3.2.1.

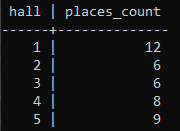


Рисунок 3.2.1 – Таблица Залы

2) Цены

SQL-запрос создания таблицы:

CREATE TABLE prices (

price\_ID integer PRIMARY KEY,

time varchar(5) NOT NULL,

week\_day varchar(11) NOT NULL,

price integer NOT NULL

);

Таблица «Цены» представлена на рисунке 3.2.2.

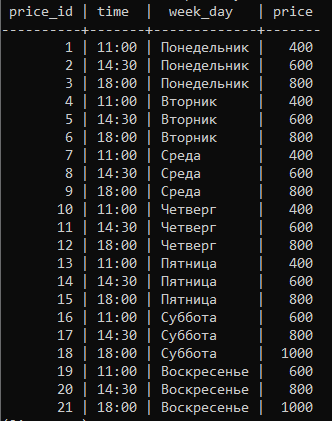


Рисунок 3.2.2 – Таблица «Цены»

3) Скидки

SQL-запрос создания таблицы:

CREATE TABLE discounts (

discount\_name varchar(25) PRIMARY KEY,

summa integer NOT NULL

);

Таблица «Скидки» представлена на рисунке 3.2.3.

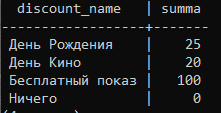


Рисунок 3.2.3 – Таблица «Скидки»

4) Льготные категории

SQL-запрос создания таблицы:

CREATE TABLE preferential\_categories (

name varchar(25) PRIMARY KEY,

price integer NOT NULL

);

Таблица «Льготные категории» представлена на рисунке 3.2.4.

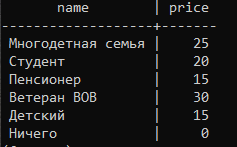


Рисунок 3.2.4 – Таблица «Льготные категории»

5) Расписание сеансов

SQL-запрос создания таблицы:

CREATE TABLE session\_schedule (

session\_ID integer PRIMARY KEY,

film varchar(50) NOT NULL,

hall integer REFERENCES halls,

session\_date date NOT NULL,

time varchar(5) NOT NULL

);

Таблица «Расписание сеансов» представлена на рисунке 3.2.5.

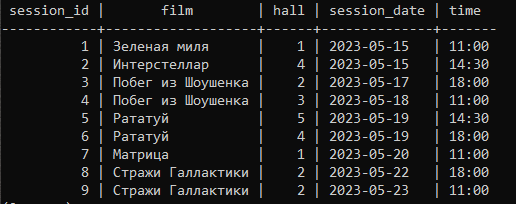


Рисунок 3.2.5 – Таблица «Расписание сеансов»

6) Табель заполнения

SQL-запрос создания таблицы:

CREATE TABLE report\_card (

place\_ID integer PRIMARY KEY,

hall integer REFERENCES halls,

row integer NOT NULL,

plase integer NOT NULL,

status varchar(15) NOT NULL

);

Таблица «Табель заполнения» представлена на рисунке 3.2.6.

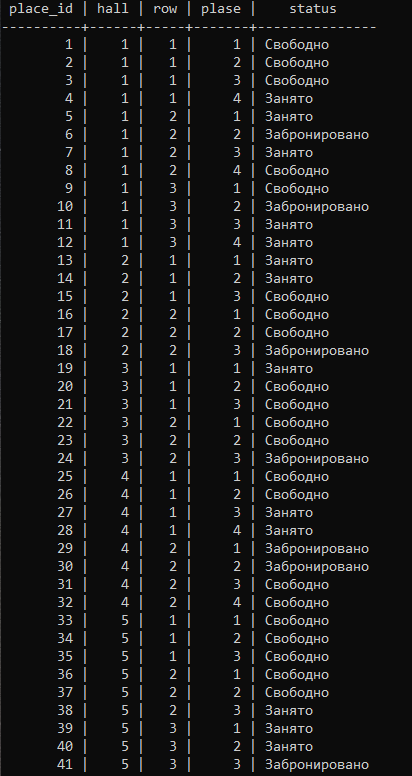


Рисунок 3.2.6 – Таблица «Табель заполнения»

7) Режиссеры

SQL-запрос создания таблицы:

CREATE TABLE producers (

producer\_ID integer PRIMARY KEY,

name varchar(15) NOT NULL,

surname varchar(20) NOT NULL,

made\_films varchar(50) NOT NULL

);

Таблица «Режиссеры» представлена на рисунке 3.2.7.

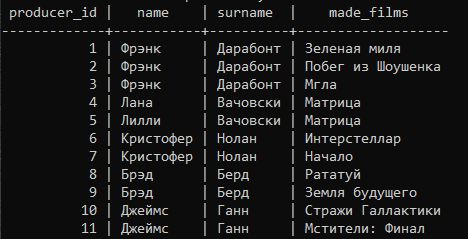


Рисунок 3.2.7 – Таблица «Режиссеры»

8) Актеры

SQL-запрос создания таблицы:

CREATE TABLE actors (

actor\_ID integer PRIMARY KEY,

name varchar(15) NOT NULL,

surname varchar(20) NOT NULL,

birthday date NOT NULL,

starred\_films varchar(50) NOT NULL

);

Таблица «Актеры» представлена на рисунке 3.2.8.



Рисунок 3.2.8 – Таблица «Актеры»

9) Фильмы

SQL-запрос создания таблицы:

CREATE TABLE films (

film\_ID integer PRIMARY KEY,

title varchar(50) NOT NULL,

description varchar(1500) NOT NULL,

production\_country varchar(15) NOT NULL,

release\_date date NOT NULL,

genre varchar(15) NOT NULL,

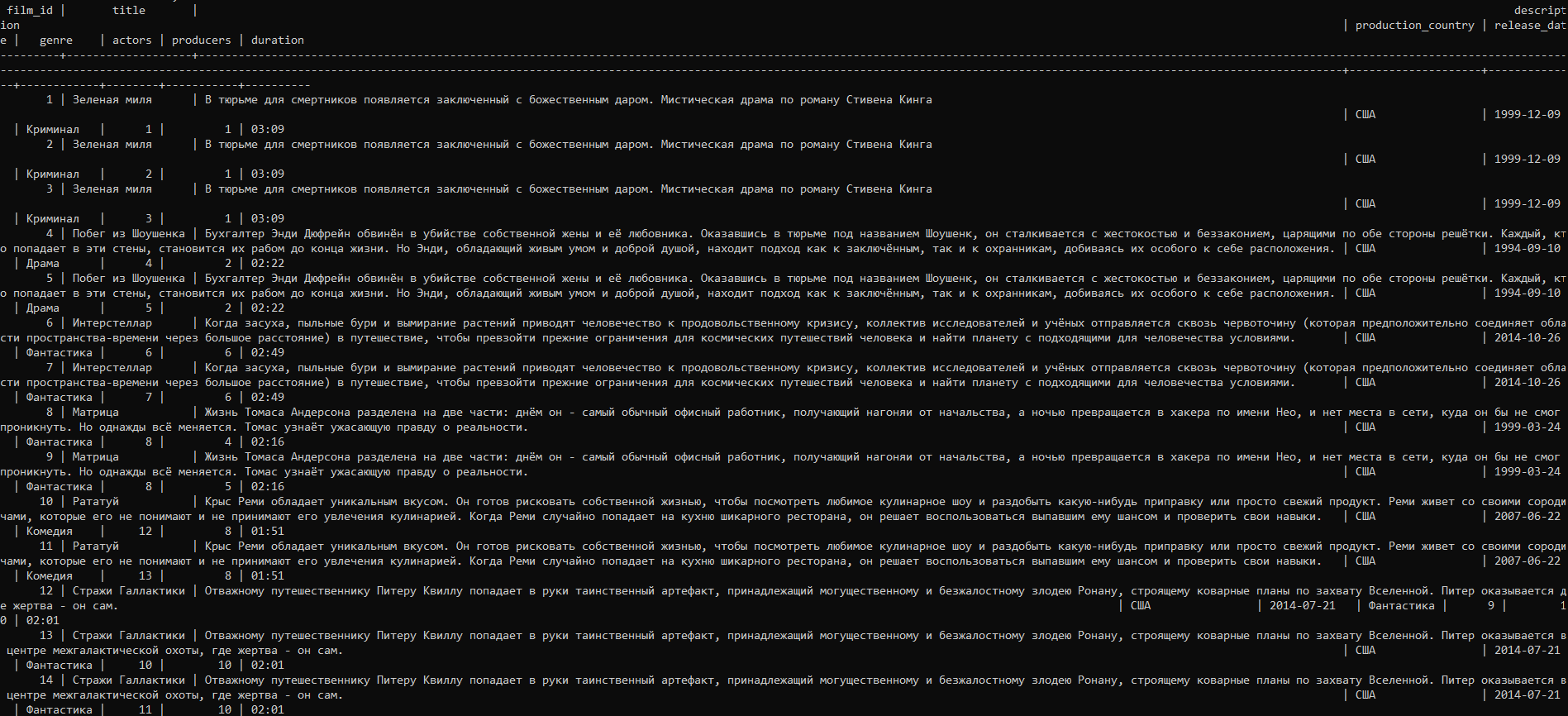
actors integer REFERENCES actors,

producers integer REFERENCES producers,

duration varchar(9) NOT NULL

);

Таблица «Фильмы» представлена на рисунке 3.2.9.

Рисунок 3.2.9 – Таблица «Фильмы»

10) Билеты

SQL-запрос создания таблицы:

CREATE TABLE tickets (

ticket\_no integer PRIMARY KEY,

film integer REFERENCES films,

price integer REFERENCES prices,

session integer REFERENCES session\_schedule,

place integer REFERENCES report\_card,

preferential\_category varchar(25) REFERENCES

preferential\_categories,

discount varchar(25) REFERENCES discounts

);

Таблица «Билеты» представлена на рисунке 3.2.10.

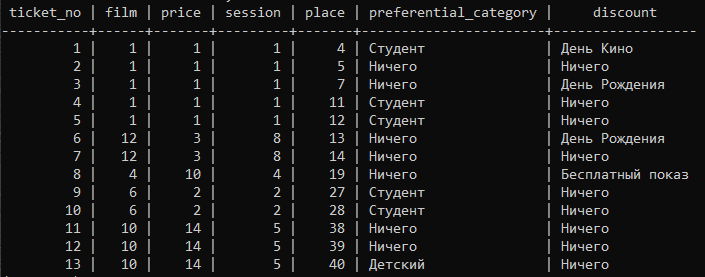


Рисунок 3.2.10 – Таблица «Билеты»

## **3.3. Разработка представлений, форм, отчетов**

Создадим несколько представлений, которые в дальнейшем будут использоваться при разработке отчетов.

Первое представление показывает информацию о названии, жанре, стране и дате выпуска, длительности фильмов, на которые ходят студенты.

CREATE OR REP LACE VIEW movie\_student\_example1 AS

SELECT DISTINCT ON (title) title, genre, production\_country,

release\_date, duration

FROM films, tickets

WHERE tickets.film=films.film\_ID AND

tickets.preferential\_category=’Студент’

ORDER BY title;

Представление movie\_student показано на рисунке 3.3.1.

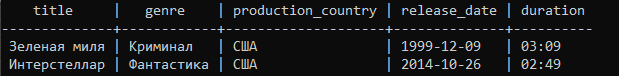


Рисунок 3.3.1 – Представление movie\_student

Второе представление показывает информацию о количестве посетителях, купивших или забронировавших место на фильм Интерстеллар.

CREATE OR REPLACE VIEW count\_customers\_on\_Interstellar AS

SELECT COUNT(place\_ID)

FROM report\_card, session\_schedule

WHERE session\_schedule.film = ’Интерстеллар’ AND session\_schedule.hall = report\_card.hall AND (report\_card.status=’Занято’ OR report\_card.status=’Забронировано’);

Представление count\_customers\_on\_Interstellar показано на рисунке 3.3.2



Рисунок 3.3.2 – Представление count\_customers\_on\_Interstellar

Третье представление показывает информацию о посещениях клиентов (фильм, стоимость, скидка, льготы) с итоговой стоимостью выше 300 рублей.

CREATE OR REPLACE VIEW prices\_more\_300 AS

SELECT title, prices.price \* (preferential\_categories.price + discounts.summa)\*0.01 AS price, preferential\_categories.name AS preferential\_category, discounts.discount\_name AS discount

FROM tickets, preferential\_categories, discounts, films, prices

WHERE tickets.preferential\_category = preferential\_categories.name AND tickets.discount = discounts.discount\_name AND tickets.film = films.film\_ID AND (prices.price \* (preferential\_categories.price + discounts.summa) \* 0.01) > 300;

Представление prices\_more\_300 показано на рисунке 3.3.3.

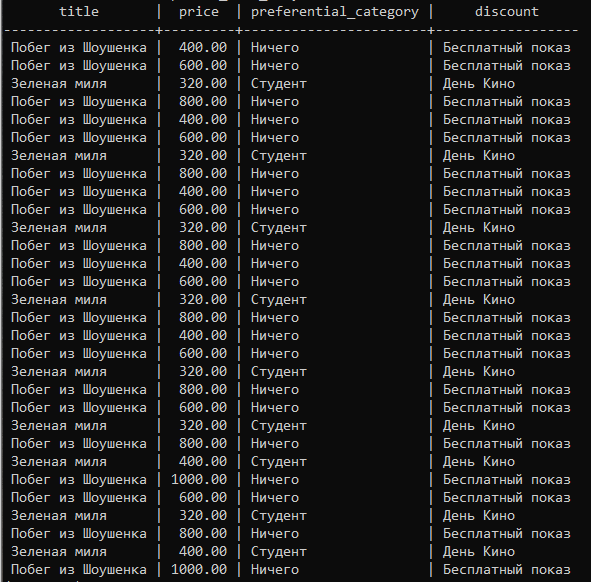


Рисунок 3.3.3 – Представление prices\_more\_300

Далее перейдем к просмотру форм.

Форма Билеты представлена на рисунке 3.3.4.

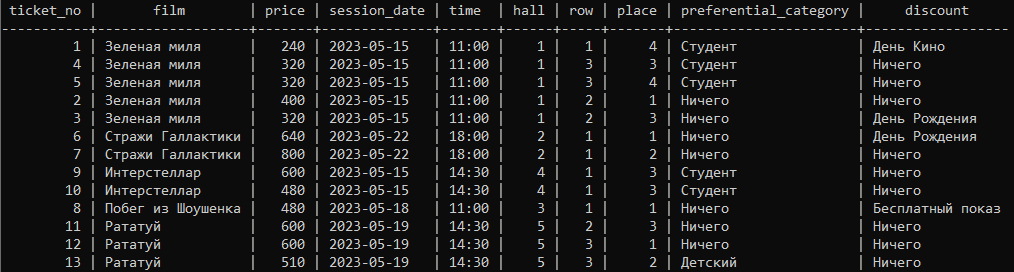


Рисунок 3.3.4 – Форма Билеты

Форма Фильмы представлена на рисунке 3.3.5.

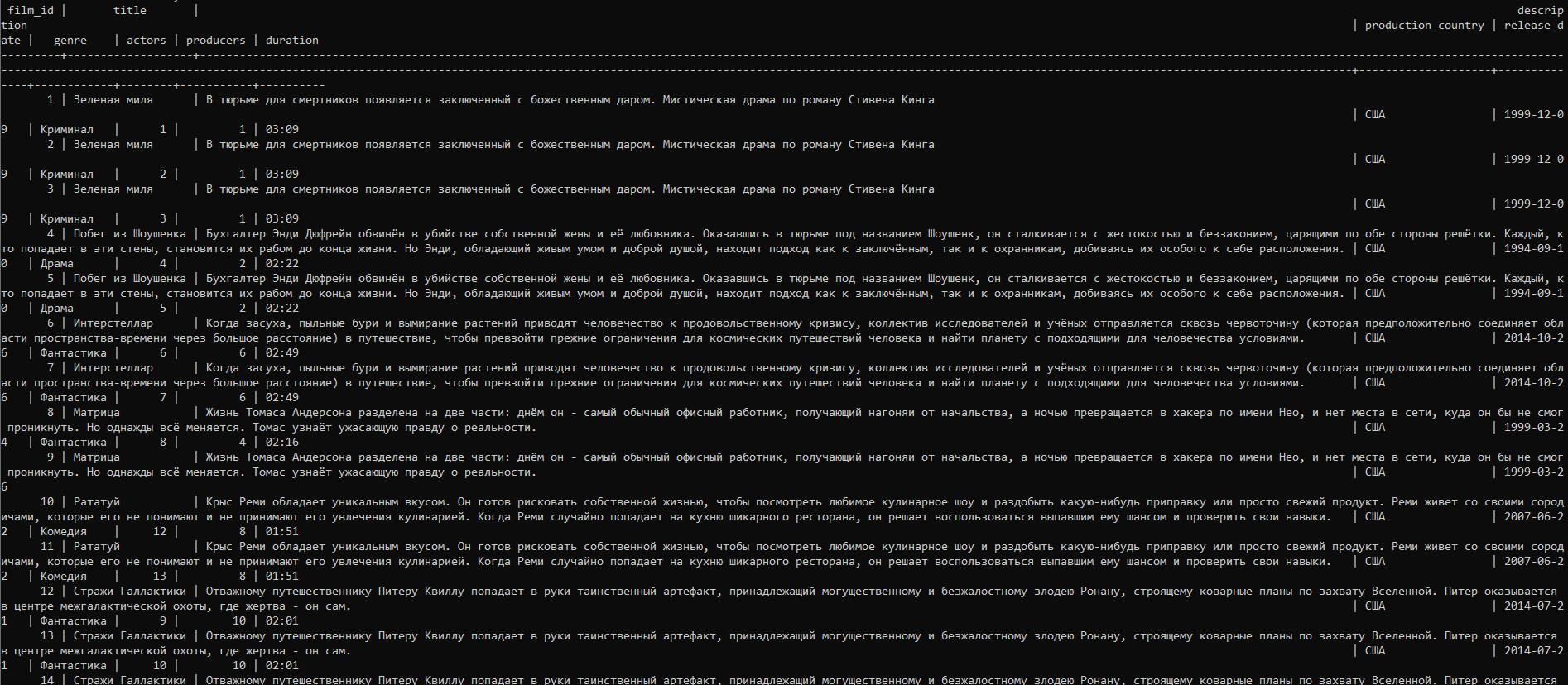


Рисунок 3.3.5 – Форма Фильмы

Форма Актеры представлена на рисунке 3.3.6.



Рисунок 3.3.6 – Форма Актеры

Форма Режиссеры представлена на рисунке 3.3.7.

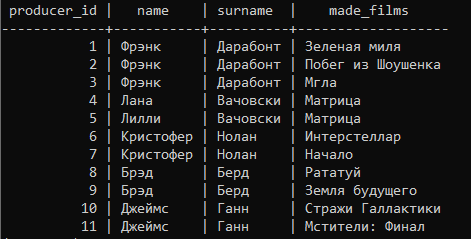


Рисунок 3.3.7 – Форма Режиссеры

Форма Табель заполнения представлена на рисунке 3.3.8.

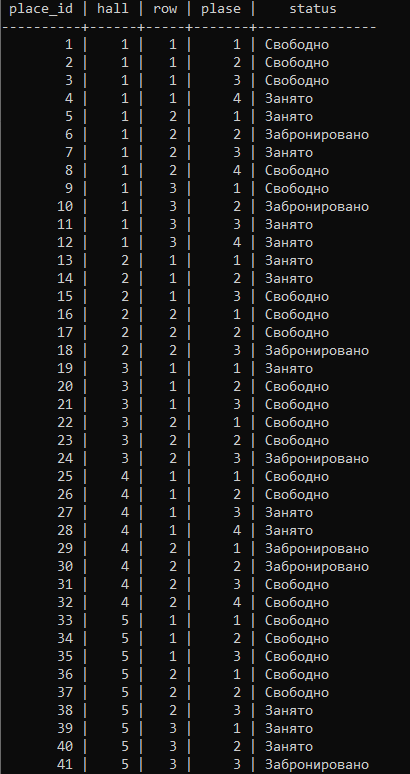


Рисунок 3.3.8 – Форма Табель заполнения

Форма Расписание сеансов представлена на рисунке 3.3.9.



Рисунок 3.3.9 – Форма Расписание сеансов

Форма Льготные категории представлена на рисунке 3.3.10.

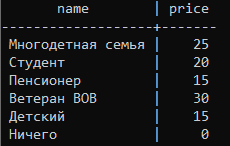


Рисунок 3.3.10 – Форма Льготные категории

Форма Скидки представлена на рисунке 3.3.11.

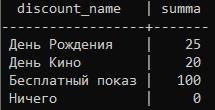


Рисунок 3.3.11 – Форма Скидки

Форма Цены представлена на рисунке 3.3.12.

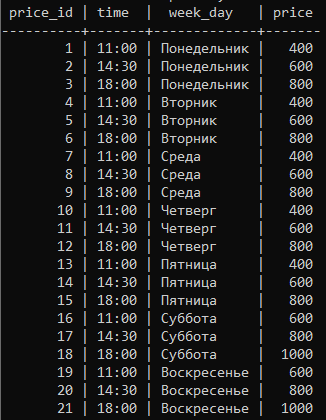


Рисунок 3.3.12 – Форма Цены

Форма Залы представлена на рисунке 3.3.13.

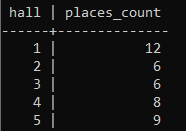


Рисунок 3.3.13 – Форма Залы

Далее разработаем отчеты для более удобного ввода данных

Для вывода отчета о билетах я создала функцию insert\_tickets:

CREATE OR REPLACE FUNCTION insert\_tickets (ticket\_no integer, film integer, price integer, session integer, place integer, preferential\_category varchar(25), discount varchar(25))

RETURNS void AS $$

BEGIN

INSERT INTO public.tickets (ticket\_no, film, price, session, place, preferential\_category, discount)

VALUES (ticket\_no, film, price, session, place, preferential\_category, discount);

END;

$$ LANGUAGE plpgsql;

Запись добавляется командой select\*from insert\_tickets(15, 8, 16, 7, 1, 'Ничего', 'Ничего');

Отчет Билеты представлен на рисунке 3.3.14.

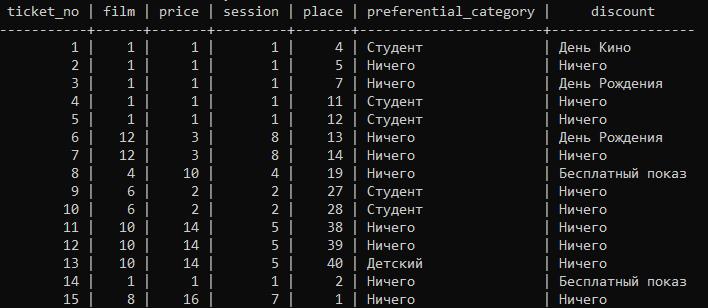


Рисунок 3.3.14 – Отчет Билеты

Для вывода отчета о фильмах я создала функцию insert\_films:

CREATE OR REPLACE FUNCTION insert\_films (film\_ID integer, title varchar(50), description varchar(1500), production\_country varchar(15), release\_date date, genre varchar(15), actors integer, producers integer, duration varchar(9))

RETURNS void AS $$

BEGIN

INSERT INTO public.films (film\_ID, title, description, production\_country, release\_date, genre, actors, producers, duration)

VALUES (film\_ID, title, description, production\_country, release\_date, genre, actors, producers, duration);

END;

$$ LANGUAGE plpgsql;

Запись добавляется командой select\*from insert\_films (15, 'Рассвет', 'Прекрасное весеннее утро. Тишина, спокойствие, а где-то вдали лежат части человеческого тела, гниющие больше недели...', 'Россия', '2003-10-07', 'Апокалипсис', 1, 1, '02:43');

Отчет Фильмы представлен на рисунке 3.3.15.

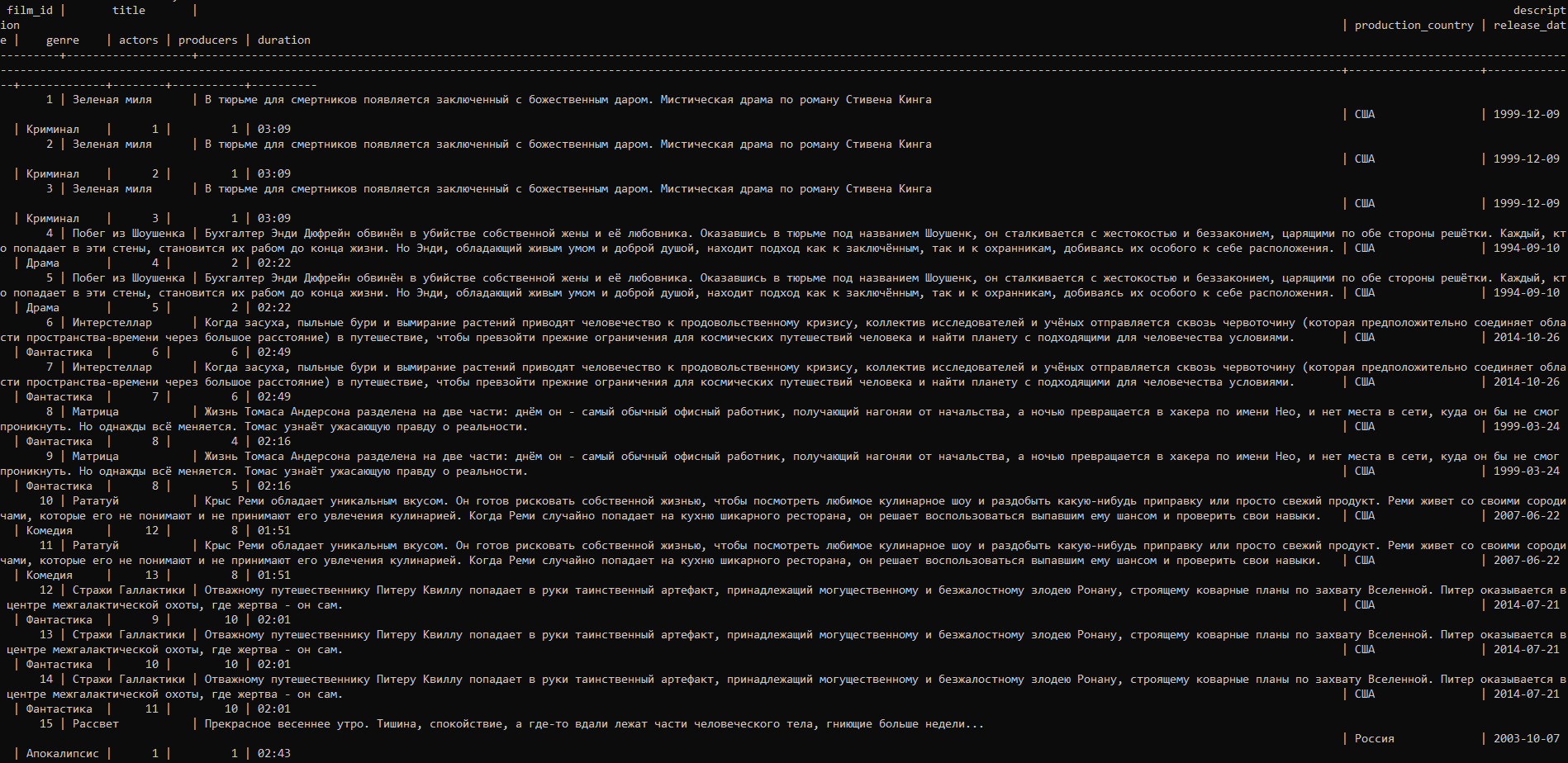


Рисунок 3.3.15 – Отчет Фильмы

Для вывода отчета об актерах я создала функцию insert\_actors:

CREATE OR REPLACE FUNCTION insert\_actors (actor\_ID integer, name varchar(15), surname varchar(20), birthday date, starred\_films varchar(50))

RETURNS void AS $$

BEGIN

INSERT INTO public.actors (actor\_ID, name, surname, birthday, starred\_films)

VALUES(actor\_ID, name, surname, birthday, starred\_films);

END;

$$ LANGUAGE plpgsql;

Запись добавляется командой select\*from insert\_actors (14, 'Крис', 'Хемсворт', '1982-02-04', 'Мстители');

Отчет Актеры представлен на рисунке 3.3.16.



Рисунок 3.3.16 – Отчет Актеры

Для вывода отчета о режиссерах я создала функцию insert\_producers:

CREATE OR REPLACE FUNCTION insert\_producers (producer\_ID integer, name varchar(15), surname varchar(20), made\_films varchar(50))

RETURNS void AS $$

BEGIN

INSERT INTO public.producers (producer\_ID, name, surname, made\_films)

VALUES(producer\_ID, name, surname, made\_films);

END;

$$ LANGUAGE plpgsql;

Запись добавляется командой select\*from insert\_producers(12, 'Джеймс', 'Ганн', 'Мстители');

Отчет Режиссеры представлен на рисунке 3.3.17.

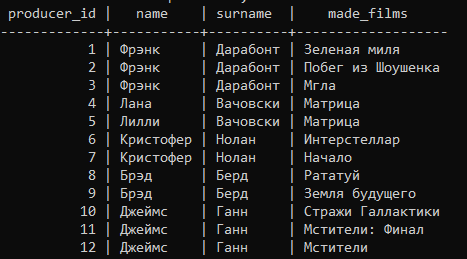


Рисунок 3.3.17 – Отчет Режиссеры

Для вывода отчета о табели заполнения создала функцию insert\_report\_card:

CREATE OR REPLACE FUNCTION insert\_report\_card (place\_ID integer, hall integer, rows integer, plase integer, status varchar(15))

RETURNS void AS $$

BEGIN

INSERT INTO public.report\_card (place\_ID, hall, row, plase, status)

VALUES(place\_ID, hall, rows, plase, status);

END;

$$ LANGUAGE plpgsql;

Запись добавляется командой select\*from insert\_report\_card(42, 5, 4, 1, 'Занято');

Отчет Табель заполнения представлен на рисунке 3.3.18.

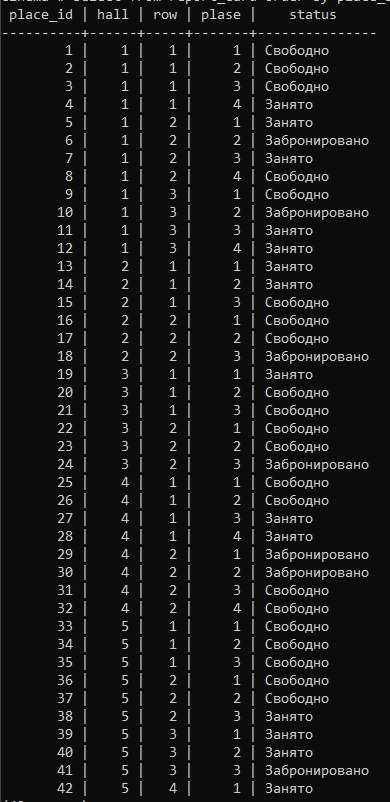


Рисунок 3.3.18 – Отчет Табель заполнения

Для вывода отчета о расписании сеансов создала функцию insert\_session\_schedule:

CREATE OR REPLACE FUNCTION insert\_session\_schedule (session\_ID integer, film varchar(50), hall integer, session\_date date, session\_time varchar(5))

RETURNS void AS $$

BEGIN

INSERT INTO public.session\_schedule (session\_ID, film, hall, session\_date, time)

VALUES(session\_ID, film, hall, session\_date, session\_time);

END;

$$ LANGUAGE plpgsql;

Запись добавляется командой select\*from insert\_session\_schedule (10, 'Мстители', 5, '2023-05-23', '14:30');

Отчет Расписание сеансов представлен на рисунке 3.3.19.

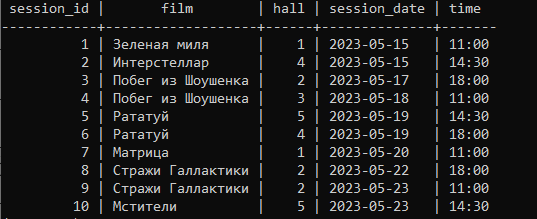


Рисунок 3.3.19 – Отчет Расписание сеансов

Для вывода отчета о льготных категориях я создала функцию insert\_ preferential\_categories:

CREATE OR REPLACE FUNCTION insert\_preferential\_categories (name varchar(25), price integer)

RETURNS void AS $$

BEGIN

INSERT INTO public.preferential\_categories (name, price)

VALUES(name, price);

END;

$$ LANGUAGE plpgsql;

Запись добавляется командой select\*from insert\_preferential\_categories ('Льготный', 5);

Отчет Льготные категории представлен на рисунке 3.3.20.

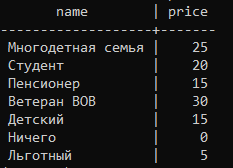


Рисунок 3.3.20 – Отчет Льготные категории

Для вывода отчета о скидках я создала функцию insert\_ discounts:

CREATE OR REPLACE FUNCTION insert\_discounts (discount\_name varchar(25), summa integer)

RETURNS void AS $$

BEGIN

INSERT INTO public.discounts(discount\_name, summa)

VALUES(discount\_name, summa);

END;

$$ LANGUAGE plpgsql;

Запись добавляется командой select\*from insert\_discounts('Группа', 15);

Отчет Скидки представлен на рисунке 3.3.21.

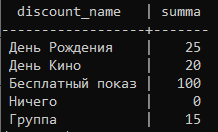


Рисунок 3.3.21 – Отчет Скидки

Для вывода отчета о ценах создала функцию insert\_ prices:

CREATE OR REPLACE FUNCTION insert\_prices (price\_ID integer, day\_time varchar(5), week\_day varchar(11), price integer)

RETURNS void AS $$

BEGIN

INSERT INTO public.prices(price\_ID, time, week\_day, price)

VALUES(price\_ID, day\_time, week\_day, price);

END;

$$ LANGUAGE plpgsql;

Запись добавляется командой select\*from insert\_prices (22, '20:30', 'Воскресенье', 700);

Отчет Цены представлен на рисунке 3.3.22.

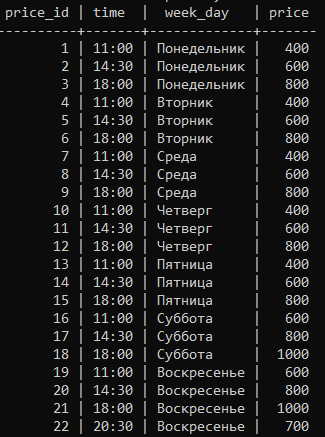


Рисунок 3.3.22 – Отчет Цены

Для вывода отчета о залах я создала функцию insert\_session\_schedule:

CREATE OR REPLACE FUNCTION insert\_halls (hall integer, places\_count integer)

RETURNS void AS $$

BEGIN

INSERT INTO public.halls(hall, places\_count)

VALUES(hall, places\_count);

END;

$$ LANGUAGE plpgsql;

Запись добавляется командой select\*from insert\_halls (6, 10);

Отчет Залы представлен на рисунке 3.3.23.

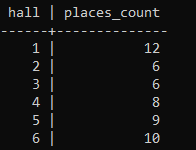


Рисунок 3.3.23 – Отчет Залы

## **3.4. Реализация ограничений, автоматизация обработки данных в БД**

Для данной базы данных требуется предусмотреть следующие ограничения:

1. Нельзя в одном зале одновременно показывать более 1 фильма;
2. Нельзя бронировать или покупать билет на купленное место;
3. Величина скидки не может быть больше 100%;
4. Величина снижения цены по льготам не может быть больше 100%.

Для реализации первого ограничения создадим функцию no\_duplicate\_sessions() и триггер no\_duplicate\_sessions\_trigger.

SQL-запрос:

CREATE FUNCTION no\_duplicate\_sessions() RETURNS TRIGGER AS $$

BEGIN

IF EXISTS (SELECT \* FROM session\_schedule WHERE hall = NEW.hall AND time = NEW.time AND session\_date=NEW.session\_date) THEN

RAISE EXCEPTION 'This session already exists';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER no\_duplicate\_sessions\_trigger

BEFORE INSERT ON session\_schedule

FOR EACH ROW

EXECUTE FUNCTION no\_duplicate\_sessions();

На рисунке 3.4.1 отображена работа триггера no\_duplicate\_sessions\_trigger.

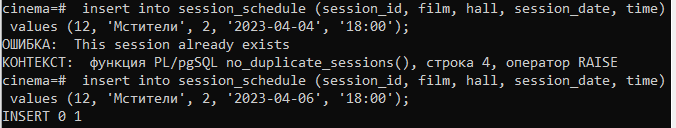


Рисунок 3.4.1 – Триггер no\_duplicate\_sessions\_trigger

Для реализации второго ограничения создадим функцию no\_booking\_when\_occupied () и триггер no\_booking\_when\_occupied\_trigger.

SQL-запрос:

CREATE FUNCTION no\_booking\_when\_occupied() RETURNS TRIGGER AS $$

BEGIN

IF EXISTS (SELECT \* FROM report\_card WHERE place\_id = NEW.place\_id AND status = 'Занято') THEN

RAISE EXCEPTION 'This seat is already occupied';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER no\_booking\_when\_occupied\_trigger

BEFORE UPDATE ON report\_card

FOR EACH ROW

EXECUTE FUNCTION no\_booking\_when\_occupied();

На рисунке 3.4.2 отображена работа триггера no\_booking\_when\_occupied\_trigger.

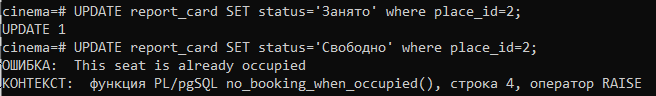


Рисунок 3.4.2 – Триггер no\_booking\_when\_occupied\_trigger

Для реализации третьего ограничения создадим функцию check\_discount () и триггер check\_discount\_trigger.

CREATE FUNCTION check\_discount() RETURNS TRIGGER AS $$

BEGIN

IF NEW.summa>100 THEN

RAISE EXCEPTION 'Value cannot be greater than 100';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_discount\_trigger

BEFORE INSERT OR UPDATE ON discounts

FOR EACH ROW

EXECUTE FUNCTION check\_discount();

На рисунке 3.4.3 отображена работа триггера check\_discount\_trigger.

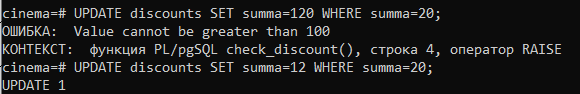


Рисунок 3.4.3 – Триггер check\_discount\_trigger

Для реализации четвертого ограничения создадим функцию check\_ preferences () и триггер check\_preferences\_trigger:

CREATE FUNCTION check\_preferences() RETURNS TRIGGER AS $$

BEGIN

IF NEW.price>100 THEN

RAISE EXCEPTION 'Value cannot be greater than 100';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_preferences\_trigger

BEFORE INSERT OR UPDATE ON preferential\_categories

FOR EACH ROW

EXECUTE FUNCTION check\_preferences();

На рисунке 3.4.4 отображена работа триггера check\_preferences \_trigger.

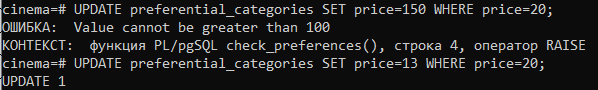


Рисунок 3.4.4 – Триггер check\_ preferences \_trigger

## **3.5 Безопасность и контроль**

С базой данных могут работать 3 типа пользователей: администратор, кассир и клиент.

Администраторы организуют работу всей базы данных. Они имеют доступ к любой информации и могут изменять структуры таблиц и данные в них.

Кассир имеет доступ к некоторым данным и может вводить необходимую для работы информацию в таблицы Табель заполнения, Билеты.

Клиент может только просматривать необходимую ему информацию и бронировать билет в таблице Табель заполнения.

* Создание роли Клиент и обозначение прав:

CREATE USER client;

GRANT SELECT ON films, actors, producers, report\_card, session\_schedule, prices, preferential\_categories, discounts TO client;

GRANT UPDATE ON report\_card TO client;

* Создание роли Кассир и обозначение прав:

CREATE USER cashier WITH PASSWORD 'cass123';

GRANT SELECT ON ALL TABLES IN SCHEMA public TO cashier;

GRANT UPDATE ON report\_card, tickets TO cashier;

* Создание роли Администратор и обозначение прав:

CREATE USER administrator WITH PASSWORD 'ad45usd220io';

GRANT ALL PRIVILEGES ON DATABASE cinema to administrator;

GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO administrator;

Теперь попробуем взаимодействовать с таблицами, используя разные роли.

Переходим в роль client при помощи команды SET ROLE client и пробуем изменить таблицу halls, к которой client не имеет доступа. Это изображено на рисунке 3.5.1.



Рисунок 3.5.1 – Запрет доступа пользователю client

Затем пробуем просмотреть расписание сеансов, на что у client есть доступ. Результат показан на рисунке 3.5.2.

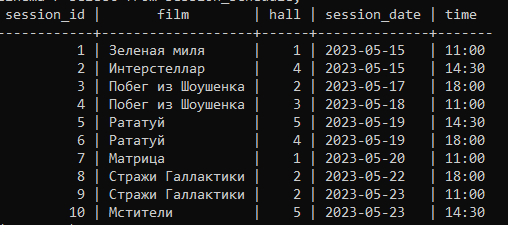


Рисунок 3.5.2 – Доступ к просмотру пользователю client

Далее переходим в роль cashier, используя команду SET ROLE cashier. При попытке изменить данные в таблице Расписание сеансов, пользователь cashier сталкивается с недостатком прав, сто изображено на рисунке 3.5.3.



Рисунок 3.5.3 – Запрет доступа пользователю cashier

Далее предпринимаем изменение табеля заполнения, на что есть права у пользователя cashier. Это изображено на рисунке 3.5.4.



Рисунок 3.5.4 – Доступ к изменению данных пользователю cashier

Теперь необходимо проверить права пользователя administrator. Он обладает всеми правами и может совершать все виды операций над данными. Один из примеров приведен на рисунке 3.5.5.

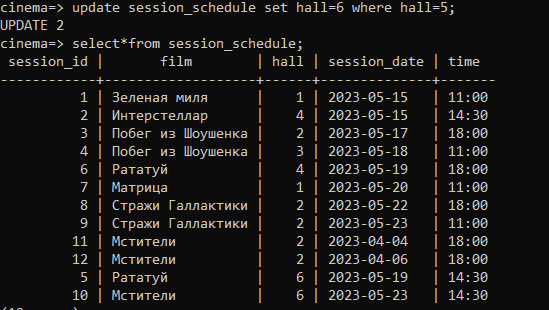


Рисунок 3.5.4 – Доступ к изменению данных пользователю administrator

## **Выводы**

В третьей главе был проведен анализ и выбор СУБД, создана база данных кинотеатра в выбранной СУБД. Также были разработаны представления, формы и отчеты, реализованы ограничения при помощи триггеров и обеспечена безопасность базы данных при использовании прав доступа.

# **ЗАКЛЮЧЕНИЕ**

В заключение курсовой работы по базе данных кинотеатра можно отметить, что создание и поддержка базы данных является важным аспектом для любого бизнеса, связанного с обработкой большого количества информации. Кинотеатры, несомненно, являются одним из таких бизнесов, и эффективная база данных может помочь им в управлении своими операциями и повышении удовлетворенности клиентов.

В рамках данной курсовой работы удалось ознакомиться с основными принципами построения реляционных баз данных при помощи СУБД postgreSQL. В первой главе был проведен анализ системной области, определены задачи пользователей базы данных кинотеатра. Также были рассмотрены основные СУБД и модели данных, сформулированы требования к разрабатываемой базе данных.

Во второй главе была разработана инфологическая модель, включающая в себя 10 взаимосвязанных таблиц. В качестве модели данных была выбрана реляционная модель и приведены аргументы, вследствие которых было принято такое решение. Также был описан переход от инфологической модели к даталогическому проектированию. После этого была описана необходимость перехода к 3 нормальной форме и осуществлен этот переход.

В третьей главе были приведены аргументы в пользу выбора PostgreSQL в качестве СУБД и создана база данных кинотеатра в СУБД PostgreSQL. Также были разработаны представления для создания форм, формы и отчеты для более удобного ввода информации, реализованы ограничения и автоматизация обработки данных в базе данных при помощи триггеров. Помимо этого, для обеспечения безопасности и контроля были созданы роли и определены права пользователей.

Таким образом, создание базы данных кинотеатра является сложным и многоэтапным процессом, который требует глубоких знаний в области баз данных и SQL-запросов. Однако, результатом является эффективная и надежная система, которая помогает управлять операциями кинотеатра, повышать эффективность работы персонала и повышать удовлетворенность клиентов.

# **СПИСОК ИСТОЧНИКОВ И ЛИТЕРАТУРЫ**

1. Вендров А.М. Практикум по проектированию программного обеспечения экономических информационных систем: Учеб. пособие / А.М. Вендров. М.: Финансы и статистика, 2010. 192 с., ил. ISBN 5-279-02440-6.
2. Информационные системы и технологии в экономике: Учебник. 2-е изд., доп. и перераб. / Т.П. Барановская, В.И. Лойко, М.И. Семенов, А.И. Трубилин; Под ред. В.И. Лойко. М.: Финансы и статистика, 2010. 416 с., ил. ISBN 5-279-02605-0.
3. Карпова Т.С. Базы данных: модели, разработка, реализация / Т.С. Карпова. СПб: Питер, 2010. 304 с. ISBN 5-272-00278-4.
4. Конгаловский М.Р. Энциклопедия технологий баз данных. М.: Финансы и статистика, 2010. 800 с.: ил. ISBN 5-279-02276-4.
5. Корнеев В.В. Базы данных. Интеллектуальная обработка информации / В.В. Корнеев, А.Ф. Гареев, С.В. Васютин, В.В. Райх. М.: Издатель Молгачева С.В., Издательство Нолидж, 2010, - 496 с.: ил. ISBN 5-89251-100-6.
6. Марков А.С. Базы данных. Введение в теорию и методологию: учебник / А.С. Марков, К.Ю. Лисовкий. М.: Финансы и статистика, 2010. 512 с. ISBN 5-279-02298-5.
7. Петров В.Н. Информационные системы / В.Н. Петров. СПб: Питер, 2002. 688 с. ISBN 5-318-00561-6.
8. Риккарди Г. Системы баз данных. Теория и практика использования в Interner и среде Java. / Грег Риккарди; пер. с англ. М.: Издательский дом «Вильямс», 2001. 480 с. ISBN 5-8459-0208-8 (рус.).
9. Хомоненко А.Д. Базы данных: учебник для высших учебных заведений / А.Д. Хомоненко, В.М. Цыганков, М.Г. Мальцев. 4-е изд., доп. и перераб. СПб: КОРОНА принт, 2004. 736 с. ISBN 5-7931-0284-1.
10. Домбровская Г., Новиков Б., Бейликова А. Оптимизация запросов в PostgreSQL / пер. с англ. Д. А. Беликова. – М.: ДМК Пресс, 2022. – 278 с.: ил
11. Новиков Б. А. Основы технологий баз данных: учебное пособие / Б. А. Новиков, Е. А. Горшкова, Н. Г. Графеева; под ред. Е. В. Рогова. — 2-е изд. — М.: ДМК Пресс, 2020. — 582 с
12. Моргунов, Е. П. PostgreSQL. Основы языка SQL: учеб. пособие / Е. П. Моргунов; под ред. Е. В. Рогова, П. В. Лузанова. — СПб.: БХВ-Петербург, 2018. — 336 с.: ил.
13. Евдокимов В. В. Экономическая информатика. Учебник для ВУЗов. СПб. Питер. 1997
14. Айвазян С.А., Мхитарян В.С. Прикладная статистика и основы эконометрики: Учебник для вузов. – М.: ЮНИТИ, 1998. – 1022 с.
15. Валеев С.Г. Оптимизация математических моделей обработки данных и информационные технологии // Вестник Ульяновского гос.техн.ун-та. Юбил.вып. – Ульяновск: УлГТУ, 1997. С. 119-129

# **ПРИЛОЖЕНИЯ**

**Приложение 1**

Описание даталогической модели

Опишем каждую сущность даталогической модели.

1. Билеты

* № билета – INTEGER NOT NULL PK
* Фильм – INTEGER NOT NULL FK
* Цена – INTEGER NOT NULL FK
* Сеанс – INTEGER NOT NULL FK
* Место – INTEGER NOT NULL FK
* Льготная категория – VARCHAR(25) NOT NULL FK
* Скидка – VARCHAR(25) NOT NULL FK

1. Фильмы:

* ID фильма – INT NOT NULL PK
* Название – VARCHAR(50) NOT NULL
* Описание – VARCHAR(1500) NOT NULL
* Страна производства – VARCHAR(15) NOT NULL
* Дата выпуска – DATE NOT NULL
* Жанр – VARCHAR(15) NOT NULL
* Актеры – INT NOT NULL FK
* Режиссеры – INT NOT NULL FK
* Длительность – VARCHAR(9) NOT NULL

1. Актеры:

* ID актера – INT NOT NULL PK
* Имя – VARCHAR(15) NOT NULL
* Фамилия – VARCHAR(20) NOT NULL
* Год рождения – DATE NOT NULL
* Фильмы(снимались) – VARCHAR(50) NOT NULL

1. Режиссеры:

* ID режиссера – INT NOT NULL
* Имя – VARCHAR(15) NOT NULL
* Фамилия – VARCHAR(20) NOT NULL
* Фильмы(снимали) – VARCHAR(50) NOT NULL

5. Табель заполнения:

* ID места – INT NOT NULL PK
* Зал – INT NOT NULL FK
* Ряд – INT NOT NULL
* Место – INT NOT NULL
* Статус – VARCHAR(15) NOT NULL

6. Расписание сеансов:

* ID сеанса – INT NOT NULL PK
* Фильм – VARCHAR(50) NOT NULL
* Зал – INT NOT NULL FK
* Дата – DATE NOT NULL
* Время – VARCHAR(5) NOT NULL

7. Льготные категории:

* Наименование – VARCHAR(25) NOT NULL PK
* Цена – INT NOT NULL

8. Скидки:

* Наименование скидки – VARCHAR(25) NOT NULL PK
* Сумма – INT NOT NULL

9. Цены:

* ID цены – INT NOT NULL PK
* Время – VARCHAR(5) NOT NULL
* День недели – VARCHAR(11) NOT NULL
* Цена – INT NOT NULL

10. Залы

* Зал – INT NOT NULL PK
* Кол-во мест – INT NOT NULL

**Приложение 2**

Создание таблиц

Реализацию базы данных в PostgreSQL.

1) Залы

SQL-запрос создания таблицы:

CREATE TABLE halls (

hall integer PRIMARY KEY,

places\_count integer NOT NULL

);

2) Цены

SQL-запрос создания таблицы:

CREATE TABLE prices (

price\_ID integer PRIMARY KEY,

time varchar(5) NOT NULL,

week\_day varchar(11) NOT NULL,

price integer NOT NULL

);

3) Скидки

SQL-запрос создания таблицы:

CREATE TABLE discounts (

discount\_name varchar(25) PRIMARY KEY,

summa integer NOT NULL

);

4) Льготные категории

SQL-запрос создания таблицы:

CREATE TABLE preferential\_categories (

name varchar(25) PRIMARY KEY,

price integer NOT NULL

);

5) Расписание сеансов

SQL-запрос создания таблицы:

CREATE TABLE session\_schedule (

session\_ID integer PRIMARY KEY,

film varchar(50) NOT NULL,

hall integer REFERENCES halls,

session\_date date NOT NULL,

time varchar(5) NOT NULL

);

6) Табель заполнения

SQL-запрос создания таблицы:

CREATE TABLE report\_card (

place\_ID integer PRIMARY KEY,

hall integer REFERENCES halls,

row integer NOT NULL,

plase integer NOT NULL,

status varchar(15) NOT NULL

);

7) Режиссеры

SQL-запрос создания таблицы:

CREATE TABLE producers (

producer\_ID integer PRIMARY KEY,

name varchar(15) NOT NULL,

surname varchar(20) NOT NULL,

made\_films varchar(50) NOT NULL

);

8) Актеры

SQL-запрос создания таблицы:

CREATE TABLE actors (

actor\_ID integer PRIMARY KEY,

name varchar(15) NOT NULL,

surname varchar(20) NOT NULL,

birthday date NOT NULL,

starred\_films varchar(50) NOT NULL

);

9) Фильмы

SQL-запрос создания таблицы:

CREATE TABLE films (

film\_ID integer PRIMARY KEY,

title varchar(50) NOT NULL,

description varchar(1500) NOT NULL,

production\_country varchar(15) NOT NULL,

release\_date date NOT NULL,

genre varchar(15) NOT NULL,

actors integer REFERENCES actors,

producers integer REFERENCES producers,

duration varchar(9) NOT NULL

);

10) Билеты

SQL-запрос создания таблицы:

CREATE TABLE tickets (

ticket\_no integer PRIMARY KEY,

film integer REFERENCES films,

price integer REFERENCES prices,

session integer REFERENCES session\_schedule,

place integer REFERENCES report\_card,

preferential\_category varchar(25) REFERENCES

preferential\_categories,

discount varchar(25) REFERENCES discounts

);

**Приложение 3**

Представления

Первое представление показывает информацию о названии, жанре, стране и дате выпуска, длительности фильмов, на которые ходят студенты.

CREATE OR REP LACE VIEW movie\_student\_example1 AS

SELECT DISTINCT ON (title) title, genre, production\_country,

release\_date, duration

FROM films, tickets

WHERE tickets.film=films.film\_ID AND

tickets.preferential\_category=’Студент’

ORDER BY title;

Второе представление показывает информацию о количестве посетителях, купивших или забронировавших место на фильм Интерстеллар.

CREATE OR REPLACE VIEW count\_customers\_on\_Interstellar AS

SELECT COUNT(place\_ID)

FROM report\_card, session\_schedule

WHERE session\_schedule.film = ’Интерстеллар’ AND session\_schedule.hall = report\_card.hall AND (report\_card.status=’Занято’ OR report\_card.status=’Забронировано’);

Третье представление показывает информацию о посещениях клиентов (фильм, стоимость, скидка, льготы) с итоговой стоимостью выше 300 рублей.

CREATE OR REPLACE VIEW prices\_more\_300 AS

SELECT title, prices.price \* (preferential\_categories.price + discounts.summa)\*0.01 AS price, preferential\_categories.name AS preferential\_category, discounts.discount\_name AS discount

FROM tickets, preferential\_categories, discounts, films, prices

WHERE tickets.preferential\_category = preferential\_categories.name AND tickets.discount = discounts.discount\_name AND tickets.film = films.film\_ID AND (prices.price \* (preferential\_categories.price + discounts.summa) \* 0.01) > 300;

**Приложение 4**

Отчеты

Для вывода отчета о билетах я создала функцию insert\_tickets:

CREATE OR REPLACE FUNCTION insert\_tickets (ticket\_no integer, film integer, price integer, session integer, place integer, preferential\_category varchar(25), discount varchar(25))

RETURNS void AS $$

BEGIN

INSERT INTO public.tickets (ticket\_no, film, price, session, place, preferential\_category, discount)

VALUES (ticket\_no, film, price, session, place, preferential\_category, discount);

END;

$$ LANGUAGE plpgsql;

Запись добавляется командой select\*from insert\_tickets(15, 8, 16, 7, 1, 'Ничего', 'Ничего');

Для вывода отчета о фильмах я создала функцию insert\_films:

CREATE OR REPLACE FUNCTION insert\_films (film\_ID integer, title varchar(50), description varchar(1500), production\_country varchar(15), release\_date date, genre varchar(15), actors integer, producers integer, duration varchar(9))

RETURNS void AS $$

BEGIN

INSERT INTO public.films (film\_ID, title, description, production\_country, release\_date, genre, actors, producers, duration)

VALUES (film\_ID, title, description, production\_country, release\_date, genre, actors, producers, duration);

END;

$$ LANGUAGE plpgsql;

Запись добавляется командой select\*from insert\_films (15, 'Рассвет', 'Прекрасное весеннее утро. Тишина, спокойствие, а где-то вдали лежат части человеческого тела, гниющие больше недели...', 'Россия', '2003-10-07', 'Апокалипсис', 1, 1, '02:43');

Для вывода отчета об актерах я создала функцию insert\_actors:

CREATE OR REPLACE FUNCTION insert\_actors (actor\_ID integer, name varchar(15), surname varchar(20), birthday date, starred\_films varchar(50))

RETURNS void AS $$

BEGIN

INSERT INTO public.actors (actor\_ID, name, surname, birthday, starred\_films)

VALUES(actor\_ID, name, surname, birthday, starred\_films);

END;

$$ LANGUAGE plpgsql;

Запись добавляется командой select\*from insert\_actors (14, 'Крис', 'Хемсворт', '1982-02-04', 'Мстители');

Для вывода отчета о режиссерах я создала функцию insert\_producers:

CREATE OR REPLACE FUNCTION insert\_producers (producer\_ID integer, name varchar(15), surname varchar(20), made\_films varchar(50))

RETURNS void AS $$

BEGIN

INSERT INTO public.producers (producer\_ID, name, surname, made\_films)

VALUES(producer\_ID, name, surname, made\_films);

END;

$$ LANGUAGE plpgsql;

Запись добавляется командой select\*from insert\_producers(12, 'Джеймс', 'Ганн', 'Мстители');

Для вывода отчета о табели заполнения создала функцию insert\_report\_card:

CREATE OR REPLACE FUNCTION insert\_report\_card (place\_ID integer, hall integer, rows integer, plase integer, status varchar(15))

RETURNS void AS $$

BEGIN

INSERT INTO public.report\_card (place\_ID, hall, row, plase, status)

VALUES(place\_ID, hall, rows, plase, status);

END;

$$ LANGUAGE plpgsql;

Запись добавляется командой select\*from insert\_report\_card(42, 5, 4, 1, 'Занято');

Для вывода отчета о расписании сеансов создала функцию insert\_session\_schedule:

CREATE OR REPLACE FUNCTION insert\_session\_schedule (session\_ID integer, film varchar(50), hall integer, session\_date date, session\_time varchar(5))

RETURNS void AS $$

BEGIN

INSERT INTO public.session\_schedule (session\_ID, film, hall, session\_date, time)

VALUES(session\_ID, film, hall, session\_date, session\_time);

END;

$$ LANGUAGE plpgsql;

Запись добавляется командой select\*from insert\_session\_schedule (10, 'Мстители', 5, '2023-05-23', '14:30');

Для вывода отчета о льготных категориях я создала функцию insert\_ preferential\_categories:

CREATE OR REPLACE FUNCTION insert\_preferential\_categories (name varchar(25), price integer)

RETURNS void AS $$

BEGIN

INSERT INTO public.preferential\_categories (name, price)

VALUES(name, price);

END;

$$ LANGUAGE plpgsql;

Запись добавляется командой select\*from insert\_preferential\_categories ('Льготный', 5);

Для вывода отчета о скидках я создала функцию insert\_ discounts:

CREATE OR REPLACE FUNCTION insert\_discounts (discount\_name varchar(25), summa integer)

RETURNS void AS $$

BEGIN

INSERT INTO public.discounts(discount\_name, summa)

VALUES(discount\_name, summa);

END;

$$ LANGUAGE plpgsql;

Запись добавляется командой select\*from insert\_discounts('Группа', 15);

Для вывода отчета о ценах создала функцию insert\_ prices:

CREATE OR REPLACE FUNCTION insert\_prices (price\_ID integer, day\_time varchar(5), week\_day varchar(11), price integer)

RETURNS void AS $$

BEGIN

INSERT INTO public.prices(price\_ID, time, week\_day, price)

VALUES(price\_ID, day\_time, week\_day, price);

END;

$$ LANGUAGE plpgsql;

Запись добавляется командой select\*from insert\_prices (22, '20:30', 'Воскресенье', 700);

Для вывода отчета о залах я создала функцию insert\_session\_schedule:

CREATE OR REPLACE FUNCTION insert\_halls (hall integer, places\_count integer)

RETURNS void AS $$

BEGIN

INSERT INTO public.halls(hall, places\_count)

VALUES(hall, places\_count);

END;

$$ LANGUAGE plpgsql;

Запись добавляется командой select\*from insert\_halls (6, 10);

**Приложение 5**

Триггеры

Для данной базы данных требуется предусмотреть следующие ограничения:

1. Нельзя в одном зале одновременно показывать более 1 фильма;

CREATE FUNCTION no\_duplicate\_sessions() RETURNS TRIGGER AS $$

BEGIN

IF EXISTS (SELECT \* FROM session\_schedule WHERE hall = NEW.hall AND time = NEW.time AND session\_date=NEW.session\_date) THEN

RAISE EXCEPTION 'This session already exists';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER no\_duplicate\_sessions\_trigger

BEFORE INSERT ON session\_schedule

FOR EACH ROW

EXECUTE FUNCTION no\_duplicate\_sessions();

1. Нельзя бронировать или покупать билет на купленное место;

CREATE FUNCTION no\_booking\_when\_occupied() RETURNS TRIGGER AS $$

BEGIN

IF EXISTS (SELECT \* FROM report\_card WHERE place\_id = NEW.place\_id AND status = 'Занято') THEN

RAISE EXCEPTION 'This seat is already occupied';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER no\_booking\_when\_occupied\_trigger

BEFORE UPDATE ON report\_card

FOR EACH ROW

EXECUTE FUNCTION no\_booking\_when\_occupied();

1. Величина скидки не может быть больше 100%;

CREATE FUNCTION check\_discount() RETURNS TRIGGER AS $$

BEGIN

IF NEW.summa>100 THEN

RAISE EXCEPTION 'Value cannot be greater than 100';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_discount\_trigger

BEFORE INSERT OR UPDATE ON discounts

FOR EACH ROW

EXECUTE FUNCTION check\_discount();

1. Величина снижения цены по льготам не может быть больше 100%.

CREATE FUNCTION check\_preferences() RETURNS TRIGGER AS $$

BEGIN

IF NEW.price>100 THEN

RAISE EXCEPTION 'Value cannot be greater than 100';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_preferences\_trigger

BEFORE INSERT OR UPDATE ON preferential\_categories

FOR EACH ROW

EXECUTE FUNCTION check\_preferences();

**Приложение 6**

Пользователи

* Создание роли Клиент и обозначение прав:

CREATE USER client;

GRANT SELECT ON ALL TABLES IN SCHEMA public TO client;

GRANT UPDATE ON report\_card TO client;

* Создание роли Кассир и обозначение прав:

CREATE USER cashier WITH PASSWORD 'cass123';

GRANT SELECT ON ALL TABLES IN SCHEMA public TO cashier;

GRANT UPDATE ON report\_card, tickets TO cashier;

* Создание роли Администратор и обозначение прав:

CREATE USER administrator WITH PASSWORD 'ad45usd220io';

GRANT ALL PRIVILEGES ON DATABASE cinema to administrator;

GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO administrator;