МИНИСТЕРСТВО ВЫСШЕГО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ

ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«РЯЗАНСКИЙ ГОСУДАРСТВЕННЫЙ РАДИОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ В.Ф. УТКИНА»

Факультет вычислительной техники

Кафедра электронных вычислительных машин

К защите

Руководитель проекта

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ**

по дисциплине «Базы данных и клиент-серверные приложения»

по теме «Разработка информационной системы с клиент-серверной архитектурой»

Выполнил:

студент группы 845

Кулаков Д. А. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата сдачи на проверку,

подпись

Руководитель проекта:

ассистент кафедры ЭВМ

Хизриева Н.И. \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

оценка дата защиты, подпись

Рязань, 2021 г

ЗАДАНИЕ

на курсовой проект по дисциплине   
«Базы данных и клиент-серверные приложения»”

Исполнитель: Кулаков Д. А.

Тема: «Разработка информационной системы с клиент-серверной архитектурой»

Вариант задания: разработка системы управления объектами кинотеатра.

Требования к используемому программному обеспечению:

* ОС: MS Windows;
* СУБД: MS SQL Server;
* система программирования: MS Visual Studio (язык программирования C#);

Общие требования к проекту:

Разработать информационную систему (ИС) с клиент-серверной архитектурой, используя двухзвенную модель DBS (сервер баз данных).

Провести инфологическое и даталогическое проектирование базы данных (БД). Проверить все таблицы БД на соответствие нормальной форме Бойса-Кодда. Предусмотреть контроль целостности данных на уровне ограничений, триггеров, правил, умолчаний и хранимых процедур. Все действия по внесению изменений в БД оформить в виде хранимых процедур. В целом, в БД должно быть не менее 10 таблиц.

Клиентская часть ИС должна охватывать все информационные потребности пользователя. Предусмотреть возможность печати необходимых пользователю отчетов. Для доступа к БД использовать технологию ADO.NET.

С помощью инсталляционной программы разработать дистрибутив ИС, позволяющий установить клиентское приложение на локальной машине.

К защите представить:

- работоспособную программу и БД, установленные на компьютере в ауд.210 или ауд.122;

- пояснительную записку в распечатанном виде;

- пояснительную записку, дистрибутив ИС, исходные тексты программ, файлы БД, сценарий инсталляции ИС в электронном виде.

Требования к содержанию пояснительной записки:

ВВЕДЕНИЕ

1. Концептуальное описание предметной области
2. Характеристики ИС
3. Анализ существующих решений для предметной области
4. Разработка общей структуры ИС
5. Разработка серверной части информационной системы
   1. Инфологическое проектирование БД
   2. Даталогическое проектирование БД
   3. Программирование объектов БД
6. Разработка клиентского приложения
   1. Выбор программных компонентов клиентской части
   2. Разработка интерфейса пользователя
7. Разработка программной документации
8. Тестирование ИС

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

ПРИЛОЖЕНИЕ А: сценарий создания объектов БД.

ПРИЛОЖЕНИЕ Б: сценарий заполнения таблиц БД.

ПРИЛОЖЕНИЕ В: исходный текст клиентской программы.

Дата выдачи задания: 19 февраля 2021 г.

Срок сдачи проекта на проверку:

Подпись исполнителя:

Подпись руководителя:

**Содержание**

ВВЕДЕНИЕ………………………………………………………………………5

1 Концептуальное описание предметной области……………………….……6

2 Характеристики ИС…………………………………………………………...7

3 Анализ существующих решений для предметной области………………...9

4 Разработка общей структуры ИС……………………………………………10

5 Разработка серверной части информационной системы

5.1 Инфологическое проектирование БД……………………………………...11

5.2 Даталогическое проектирование БД……………………………………....18

5.3 Программирование объектов БД…………………………………………..33

6 Разработка клиентского приложения

6.1 Выбор программных компонентов клиентской части…………………...38

6.2 Разработка интерфейса пользователя……………………………………..39

7 Разработка программной документации……………………………………41

8 Тестирование ИС……………………………………………………………..42

ЗАКЛЮЧЕНИЕ………………………………………………………………...50

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ………………………………51

ПРИЛОЖЕНИЕ А: сценарий создания объектов БД………………………..52

ПРИЛОЖЕНИЕ Б: сценарий заполнения таблиц БД………………………..65

ПРИЛОЖЕНИЕ В: исходный текст клиентской программы……………….68

**Введение**

В современном мире с повсеместном развитием способов получения информации для людей острым встает вопрос затраты времени. Поэтому разрабатываются системы, позволяющие быстро получить необходимую информацию. Сейчас существуют крупные системы, обладающие большой скоростью, значительным объемом поисковых данных и их доступностью.

В данной курсовой работе, призванной ознакомить студентов с проектированием подобной системы, но куда более меньших размеров, была разработана система для кинотеатра, которая позволяет управлять основными объектами. Использование веб-приложения для кинотеатра позволяет узнавать о расписаниях и покупать билеты фильмы там, где удобно зрителю. То есть нет необходимости в посещении кинотеатра, кроме как для просмотра киноленты.

База данных разработана в среде Microsoft SQL Server 2018. Клиентское приложение (сайт) разработано в среде Microsoft Visual Studio. Использованные языки: Transact-SQL, C#, JS.

1. **Концептуальное описание предметной области**

Предметная область – кинотеатр, предоставляющий прокат различных фильмов.

Кинотеатр должен обладать определенным административным и оперативным штатом, в который входят: администраторы, менеджеры, киномеханики, кассиры (работники у залов), уборщики.

Основная деятельность – показ фильмов, поэтому у кинотеатра должна быть база кинолент, со временем меняющая свой состав. В состав так же входят кинозалы, кассы.

Кинотеатр имеет возможность предоставление проката реклам перед сеансами.

Покупка билетов может осуществляться через кассы, либо через интернет-сайт.

1. **Характеристики ИС**

Хранимые данные в базе:

* информация о кинозалах:
  + номер зала, наличие 3D, количество рядов и мест;
* информация о прокатываемым фильмам:
  + название фильма, год выхода, режиссер, длительность, жанр, рейтинг;
* информация о сотрудниках:
  + ФИО, должность, паспорт, опыт, номер телефона;
* информация о должностях:
  + название позиции, описание, ранг, зарплата;
* информация о сеансах:
  + номер фильма, номер зала, время показа, возрастной рейтинг, тип сеанса, стоимость билета;
* информация о рекламодателях:
  + ФИО, название компании, номер телефона;
* информация о прокатываемой рекламе:
  + номер сеанса, ответственный сотрудник, рекламодатель, название рекламы, длительность, стоимость;
* информация о кассах
  + номер кассы, сотрудник, время смены, время работы;
* информация о билетах
  + номер сеанса, номер кассы, тип билета, номер ряда и места;
* информация о типах сеансов, билетов
  + название типа, описание, скидка (для билетов);
* так же информация о том, к чему прикреплены сотрудники (кинозалы, кассы):
  + идентификационный номер сотрудника, номер объекта;

Бизнес-правила, накладываемые на базу:

* в одном сеансе может прокатываться только один кинофильм;
* билет на сеанс не стандартного типа может быть куплен только в кинотеатре;
* в одно и тоже время в одном зале не могут идти разные сеансы;
* сеансы могут выставляться только на будущие даты;
* рекламный ролик должен стоить не меньше 5 тыс. рублей;
* рекламный ролик должен длиться не более 180 секунд;
* сотрудник может занимать только одну должность;
* билет может быть куплен только на один сеанс, но сеанс может иметь множество билетов;
* один работник-кассир может обслуживать множество кинозалов, но кинозал может иметь множество работников;
* реклама может проигрываться на множестве сеансов, но сеанс может иметь множество рекламных роликов;
* принимать рекламу и назначать ее на сеансы могут только сотрудники, чей ранг равен трем;
* работать на кассах могут только работники с рангом равным двум;
* билеты, купленные через интернет-сайт, могут быть только стандартного типа;
* ФИО, паспорт, номер телефона для сотрудников должны быть уникальными;
* ФИО и номер телефона рекламодателя должны быть уникальными;

1. **Анализ существующих решений для предметной области**

Для курсовой предметной области уже существует множество решений, позволяющих получить обширный объем информации не только о фильме, но и дополнительную информацию, такую как рецензии, события, связанные с фильмом, сборы и другие.

Подобные системы сходны по функционалу, поэтому для рассмотрения был взят портал «КиноПоиск».

На сайте в список расположены фильмы, находящиеся в прокате. Для каждого фильма выводится название, изображение, режиссер и жанр. Также рейтинг, дата премьеры и кнопка для перехода к покупке билетов. Подобное представление позволяет пользователю быстро оценить текущую афишу и определиться с выбором.

Если этой информации недостаточно, то пользователь может перейти к расширенной информации о фильме.

Когда пользователь определится с выбором, то он может перейти к сеансам, выбрать подходящий, далее забронировать места.

На портале доступно большое количество функций, однако для курсового проекта выбор остановился на описанном выше наборе.

1. **Разработка общей структуры ИС**

В данной курсовой работе для построения информационной системы была взята «Клиент-Серверная» архитектура. Система на этой архитектуре делится на две подсистемы: клиентскую и серверную.

Клиентская часть представляет собой компьютер, на котором находится приложение (программа, сайт и т.п.), с помощью которого пользователь получает и запрашивает необходимую информацию, которая «обрисовывается» в приложении.

На серверной части расположены обработчик, СУБД и база данных. Они принимают запрос с клиентской части, обрабатывают его и возвращают результат. Сервер – это так же компьютер, который мощнее и производительнее. На нем располагается основная логика.

Подобное разделение позволяет снизить требования к клиентским компьютерам, облегчить управление и наладку сервера, обезопасить хранимую информацию баз данных, снизить нагрузку на сеть.

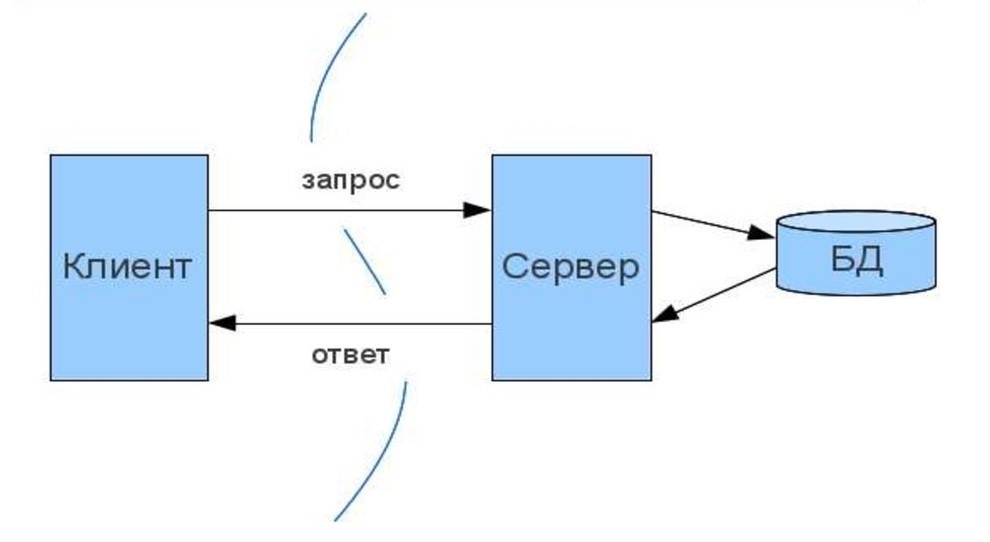


Рисунок 4.1 – Клиент-Серверная архитектура

1. **Разработка серверной части ИС**
   1. **Инфологическое проектирование БД**

Инфологическое проектирование – это процесс создания инфологической модели данных о предметной области, не зависящее от любых физических аспектов ее представления.

Подобная модель данных не подчиняется условностям какой-либо СУБД, вычислительной среды или языков программирования. Модель представляет собой структуру знаний об области: выделяются и классифицируются компоненты, связи между ними.

В данной курсовой работе для концептуального проектирования выбрана модель «Сущность-Связь» − ER-модель, позволяющая выделять ключевые сущности и связи между ними. Нотация П. Чена.

**Выявление сущностей и связей**

В выбранной предметной области можно выявить следующие сущности:

* Кинолента;
* Сотрудник;
* Рекламодатель;
* Кинозал;
* Сеанс;
* Билет;
* Касса;
* Реклама;
* Должность.

Между сущностями можно выделить следующие связи:

* Сеанс содержит кинофильм;
* Сеанс проигрывается в кинозале;
* Сеанс содержит рекламу;
* Билет используется для прохода на сеанс;
* Билет покупается на кассе;
* Рекламодатель покупает рекламу;
* Сотрудник продает рекламу;
* Сотрудник обладает должностью;
* Сотрудник работает на кассе;
* Сотрудник работает в кинозале;
* Сеанс и билет обладают типом;

**Построение ER-диаграмм**

На рисунках 5.1.1 – 5.1.15 приведены ER-диаграммы для связей.

1. Сеанс содержит кинофильм



Рисунок 5.1.1 – ER-диаграмма: Сеанс содержит кинофильм

Во время сеанса может проигрываться только один кинофильм. Один кинофильм может проигрываться в множестве сеансов.

Кинофильм не обязательно содержит сеанс. Сеанс обязательно содержит кинофильм.

1. Сеанс проигрывается в кинозале

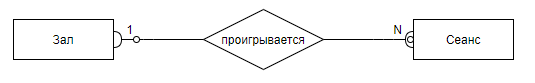


Рисунок 5.1.2 – ER-диаграмма: Сеанс проигрывается в кинозале

Сеанс может проигрываться только в одном кинозале. В кинозале может проигрываться множество сеансов.

Сеанс обязательно содержит кинозал. Зал не обязательно содержит сеанс.

1. Сеанс содержит рекламу

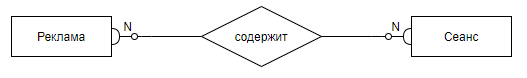


Рисунок 5.1.3 – ER-диаграмма: Сеанс содержит рекламу

В одном сеансе может быть много рекламы. Одна реклама может быть на множестве сеансов.

Реклама и сеанс не обязательно содержат друг друга

1. Билет используется для прохода на сеанс

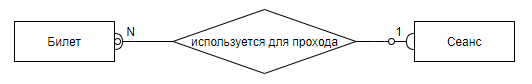


Рисунок 5.1.4 – ER-диаграмма: Билет используется для прохода на сеанс

Билет содержит только один сеанс. На сеанс может быть продано множество билетов.

Билет обязательно содержит сеанс. Сеанс не обязательно содержит билеты.

1. Билет покупается на кассе

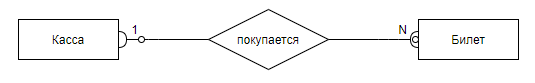


Рисунок 5.1.5 – ER-диаграмма: Билет покупается на кассе

На кассе можно купить множество билетов. Билет содержит только одну кассу.

Касса не обязательно содержит билет. Билет обязательно содержит кассу.

1. Рекламодатель покупает рекламу



Рисунок 5.1.6 – ER-диаграмма: Рекламодатель покупает рекламу

Рекламодатель может покупать множество рекламы. Реклама содержит только одного рекламодателя.

Рекламодатель не обязательно покупает рекламу. Реклама обязательно содержит рекламодателя.

1. Сотрудник продает рекламу



Рисунок 5.1.7 – ER-диаграмма: Сотрудник продает рекламу

Сотрудник может продавать множество рекламы. Реклама содержит множество сотрудников.

Сотрудник не обязательно продает рекламу. Реклама обязательно содержит сотрудника.

1. Сотрудник обладает должностью

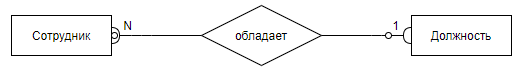


Рисунок 5.1.8 – ER-диаграмма: Сотрудник обладает должностью

Сотрудник может обладать только одной должностью. Должностью может обладать множество сотрудников.

Сотрудник обязательно содержит должность. Должность не обязательно содержит сотрудника.

1. Сотрудник работает на кассе;

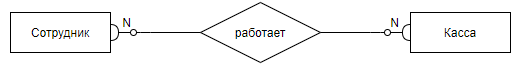


Рисунок 5.1.9 – ER-диаграмма: Сотрудник работает на кассе

Сотрудник может работать на множестве касс. На кассе может работать множество сотрудников.

Сотрудник не обязательно работает на кассе. Касса не обязательно содержит сотрудника.

1. Сотрудник работает в кинозале



Рисунок 5.1.10 – ER-диаграмма: Сотрудник работает в кинозале

Сотрудник может работать во множестве кинозалов. В зале может работать множество сотрудников.

Сотрудник не обязательно работает в зале. Кинозал не обязательно содержит сотрудника.

1. Сеанс обладает типом



Рисунок 5.1.11 – ER-диаграмма: Сеанс обладает типом

Сеанс может обладать только одним типом. Тип может быть у множества сеансов.

Сеанс обязательно обладает типом. Тип не обязательно содержит сеанс.

1. Билет обладает типом

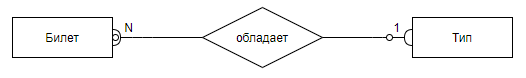


Рисунок 5.1.12 – ER-диаграмма: Билет обладает типом

Билет может обладать только одним типом. Тип может быть у множества билетов.

Билет обязательно обладает типом. Тип не обязательно содержит билет.

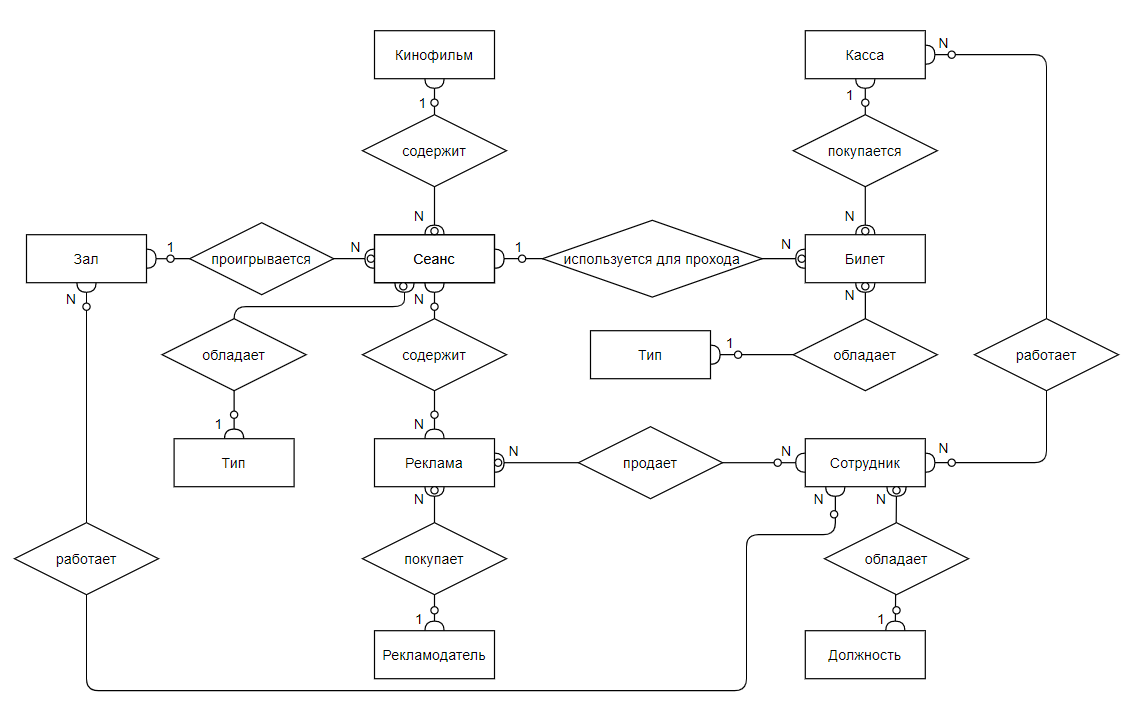


Рисунок 5.1.13 – Общая ER-диаграмма

* 1. **Даталогическое проектирование БД**

Даталогическое проектирование – это процесс создания даталогической модели, которая отражает логические связи между элементами данных независимо от их содержания и физической структуры. При этом модель должна быть представлена в терминах модели данных концептуального уровня для выбранной СУБД.

Существует несколько представлений структур данных. Для данной курсовой работы был выбран реляционный подход, где отношения представлены таблицами и связями между ними. Для таких баз применяется нормализация, целью которой является устранение недостатков структуры базы данных, приводящих к избыточности, которая, в свою очередь, потенциально приводит к различным аномалиям и нарушениям целостности данных.

При формировании отношений по ER-диаграммам используются определенные правила, описывающие отношения и зависимости между ними.

**Переход от ER-диаграмм к предварительным отношениям**

Проанализировав составленные ранее ER-диаграммы, можно сформировать следующие отношения:

1. Сеанс содержит кинофильм. По правилу 4 получаем:

* Сеанс (SeanceID, FilmID);
* Кинофильм (FilmID);

1. Сеанс проигрывается в кинозале. По правилу 4 получаем:

* Сеанс (SeanceID, HollID);
* Кинозал (HollID);

1. Сеанс содержит рекламу:

* Сеанс (SeanceID);
* Реклама (AdID);
* СеансРеклама (SeanceID, AdID);

1. Билет используется для прохода на сеанс. По правилу4 получаем:

* Сеанс (SeanceID);
* Билет (TicketID, SeanceID);

1. Билет покупается на кассе. По правилу 4 получаем:

* Билет (TicketID, CashboxID);
* Касса (CashboxID);

1. Рекламодатель покупает рекламу. По правилу 4 получаем:

* Рекламодатель (AdvertiserID, AdID);
* Реклама (AdID);

1. Сотрудник продает рекламу. По правилу 6 получаем:

* Сотрудник (EmployeeID);
* Реклама (AdID);
* РекламаСотрудник (EmployeeID, AdID);

1. Сотрудник обладает должностью. По правилу 4 получаем:

* Сотрудник (EmployeeID, PositionName);
* Должность (PositionName);

1. Сотрудник работает на кассе. По правилу 6 получаем:

* Сотрудник (EmployeeID);
* Касса (CashboxID);
* КассаСотрудник (CashboxID, EmployeeID);

1. Сотрудник работает в кинозале. По правилу 6 получаем:

* Сотрудник (EmployeeID);
* Кинозал (HollID);
* КинозалСотрудник (HollID, EmployeeID);

1. Сеанс обладает типом. По правилу 4 получаем:

* Сеанс (SeanceID, TypeName);
* Тип (TypeName);

1. Билет обладает типом. По правилу 4 получаем:

* Билет (TicketID, TypeName);
* Тип (TypeName).

**Заполнение предварительных отношений атрибутами**

1. Сеанс (SeanceID, FilmID, HollID, ShowTime, AgeRating, SeanceType, TicketCost);
2. Кинофильм (FilmID, FilmName, ReleaseYear, Director, Duration, Genre, Rating, FilmImage);
3. Кинозал (HollID, TdEnable, RowNumber, SeatNumber);
4. Реклама (AdID, Advertiser, AdvertisingName, AdvertisingDuration, AdvertisingCost);
5. Рекламодатель (AdvertiserID, AdvertiserName, CompanyName, AdvertiserPhone);
6. Билет (TicketID, TicketType, SeanceID, CashboxID, RowNumber, SeatNumber, Cost);
7. Касса (CashboxID, StaffChangeTime, WorkTime);
8. Сотрудник (EmployeeID, EmployeeName, Position, Passport, Experience, Phone);
9. Должность (PositionName, Responsibilities, EmployeeRank, Salary);
10. КассаСотрудник (EmployeeID, CashboxID);
11. КинозалСотрудник (HollID, EmployeeID, StaffChangeTime);
12. РекламаСотрудник (EmployeeID, AdID);
13. Тип для билета (TypeName, TypeDescription, Discount);
14. Тип для сеанса (TypeName, TypeDescription);
15. СеансРеклама (SeanceID, AdID);

**Проверка предварительных отношений на соответствие нормальным формам**

На рисунках 5.2.1 – 5.2.15 приведены схемы предварительных отношений.

1. Сеанс (SeanceID, FilmID, HollID, ShowTime, AgeRating, SeanceType, TicketCost):

* Отношение находится в 2НФ, так как оно находится в 1НФ и каждый не ключевой атрибут функционально полно зависит от потенциального ключа;
* Отношение находится в 3НФ, так как оно находится в 2НФ и отсутствуют транзитивные зависимости не ключевых атрибутов от ключевых;
* Отношение находится в БКНФ, так как каждая функциональная зависимость имеет в качестве своего детерминанта потенциальный ключ;

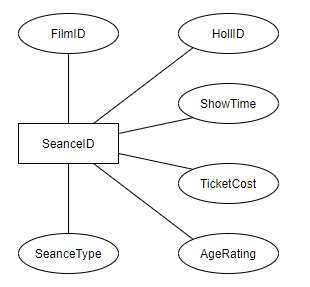


Рисунок 5.2.1 – Отношение Сеанс

1. Кинофильм (FilmID, FilmName, ReleaseYear, Director, Duration, Genre, Rating, FilmImage):

* Отношение находится в 2НФ, так как оно находится в 1НФ и каждый не ключевой атрибут функционально полно зависит от потенциального ключа;
* Отношение находится в 3НФ, так как оно находится в 2НФ и отсутствуют транзитивные зависимости не ключевых атрибутов от ключевых;
* Отношение находится в БКНФ, так как каждая функциональная зависимость имеет в качестве своего детерминанта потенциальный ключ;

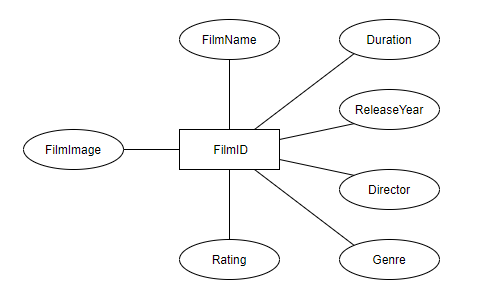


Рисунок 5.2.2 – Отношение Кинофильм

1. Кинозал (HollID, TdEnable, RowNumber, SeatNumber):

* Отношение находится в 2НФ, так как оно находится в 1НФ и каждый не ключевой атрибут функционально полно зависит от потенциального ключа;
* Отношение находится в 3НФ, так как оно находится в 2НФ и отсутствуют транзитивные зависимости не ключевых атрибутов от ключевых;
* Отношение находится в БКНФ, так как каждая функциональная зависимость имеет в качестве своего детерминанта потенциальный ключ;

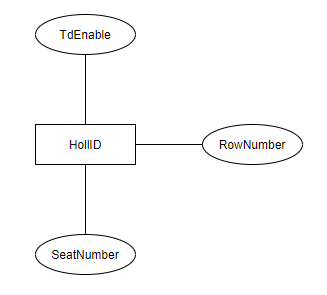


Рисунок 5.2.3 – Отношение Кинозал

1. Реклама (AdID, Advertiser, AdvertisingName, AdvertisingDuration, AdvertisingCost):

* Отношение находится в 2НФ, так как оно находится в 1НФ и каждый не ключевой атрибут функционально полно зависит от потенциального ключа;
* Отношение находится в 3НФ, так как оно находится в 2НФ и отсутствуют транзитивные зависимости не ключевых атрибутов от ключевых;
* Отношение находится в БКНФ, так как каждая функциональная зависимость имеет в качестве своего детерминанта потенциальный ключ;

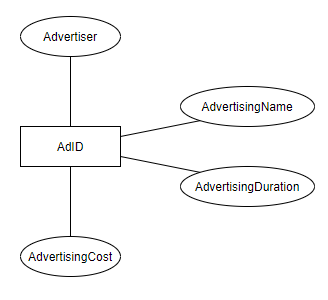


Рисунок 5.2.4 – Отношение Реклама

1. Рекламодатель (AdvertiserID, AdvertiserName, CompanyName, AdvertiserPhone) :

* Отношение находится в 2НФ, так как оно находится в 1НФ и каждый не ключевой атрибут функционально полно зависит от потенциального ключа;
* Отношение находится в 3НФ, так как оно находится в 2НФ и отсутствуют транзитивные зависимости не ключевых атрибутов от ключевых;
* Отношение находится в БКНФ, так как каждая функциональная зависимость имеет в качестве своего детерминанта потенциальный ключ;

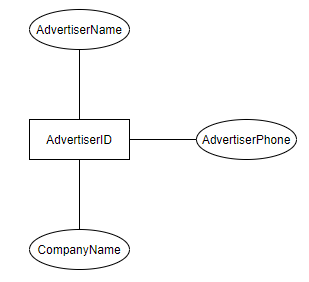


Рисунок 5.2.5 – Отношение Рекламодатель

1. Билет (TicketID, TicketType, SeanceID, CashboxID, RowNumber, SeatNumber, Cost):

* Отношение находится в 2НФ, так как оно находится в 1НФ и каждый не ключевой атрибут функционально полно зависит от потенциального ключа;
* Отношение находится в 3НФ, так как оно находится в 2НФ и отсутствуют транзитивные зависимости не ключевых атрибутов от ключевых;
* Отношение находится в БКНФ, так как каждая функциональная зависимость имеет в качестве своего детерминанта потенциальный ключ;

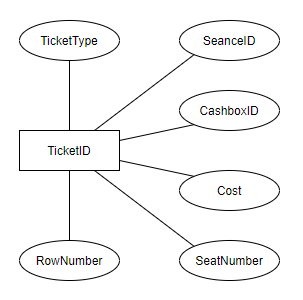


Рисунок 5.2.6 – Отношение Билет

1. Касса (CashboxID, StaffChangeTime, WorkTime):

* Отношение находится в 2НФ, так как оно находится в 1НФ и каждый не ключевой атрибут функционально полно зависит от потенциального ключа;
* Отношение находится в 3НФ, так как оно находится в 2НФ и отсутствуют транзитивные зависимости не ключевых атрибутов от ключевых;
* Отношение находится в БКНФ, так как каждая функциональная зависимость имеет в качестве своего детерминанта потенциальный ключ;

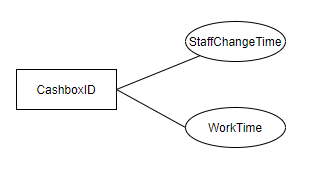


Рисунок 5.2.7 – Отношение Касса

1. Сотрудник (EmployeeID, EmployeeName, Position, Passport, Experience, Phone):

* Отношение находится в 2НФ, так как оно находится в 1НФ и каждый не ключевой атрибут функционально полно зависит от потенциального ключа;
* Отношение находится в 3НФ, так как оно находится в 2НФ и отсутствуют транзитивные зависимости не ключевых атрибутов от ключевых;
* Отношение находится в БКНФ, так как каждая функциональная зависимость имеет в качестве своего детерминанта потенциальный ключ;

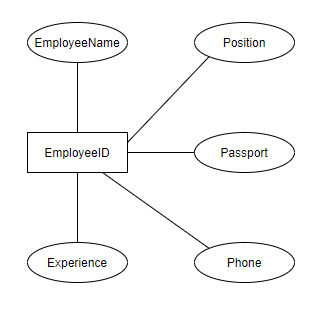


Рисунок 5.2.8 – Отношение Сотрудник

1. Должность (PositionName, Responsibilities, EmployeeRank, Salary):

* Отношение находится в 2НФ, так как оно находится в 1НФ и каждый не ключевой атрибут функционально полно зависит от потенциального ключа;
* Отношение находится в 3НФ, так как оно находится в 2НФ и отсутствуют транзитивные зависимости не ключевых атрибутов от ключевых;
* Отношение находится в БКНФ, так как каждая функциональная зависимость имеет в качестве своего детерминанта потенциальный ключ;

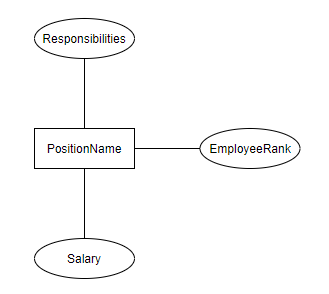


Рисунок 5.2.9 – Отношение Должность

1. КассаСотрудник (EmployeeID, CashboxID):

* Отношение находится в 2НФ, так как оно находится в 1НФ и каждый не ключевой атрибут функционально полно зависит от потенциального ключа;
* Отношение находится в 3НФ, так как оно находится в 2НФ и отсутствуют транзитивные зависимости не ключевых атрибутов от ключевых;
* Отношение находится в БКНФ, так как каждая функциональная зависимость имеет в качестве своего детерминанта потенциальный ключ;

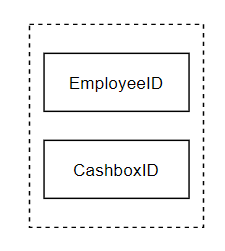


Рисунок 5.2.10 – Отношение КассаСотрудник

1. КинозалСотрудник (HollID, EmployeeID, StaffChangeTime):

* Отношение находится в 2НФ, так как оно находится в 1НФ и каждый не ключевой атрибут функционально полно зависит от потенциального ключа;
* Отношение находится в 3НФ, так как оно находится в 2НФ и отсутствуют транзитивные зависимости не ключевых атрибутов от ключевых;
* Отношение находится в БКНФ, так как каждая функциональная зависимость имеет в качестве своего детерминанта потенциальный ключ;

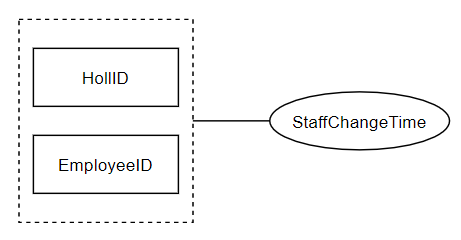


Рисунок 5.2.11 – Отношение КинозалСотрудник

1. РекламаСотрудник (EmployeeID, AdID):

* Отношение находится в 2НФ, так как оно находится в 1НФ и каждый не ключевой атрибут функционально полно зависит от потенциального ключа;
* Отношение находится в 3НФ, так как оно находится в 2НФ и отсутствуют транзитивные зависимости не ключевых атрибутов от ключевых;
* Отношение находится в БКНФ, так как каждая функциональная зависимость имеет в качестве своего детерминанта потенциальный ключ;

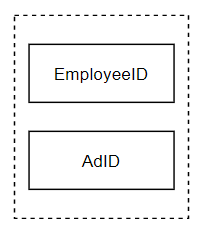


Рисунок 5.2.12 – Отношение РекламаСотрудник

1. Тип для билета (TypeName, TypeDescription, Discount):

* Отношение находится в 2НФ, так как оно находится в 1НФ и каждый не ключевой атрибут функционально полно зависит от потенциального ключа;
* Отношение находится в 3НФ, так как оно находится в 2НФ и отсутствуют транзитивные зависимости не ключевых атрибутов от ключевых;
* Отношение находится в БКНФ, так как каждая функциональная зависимость имеет в качестве своего детерминанта потенциальный ключ;

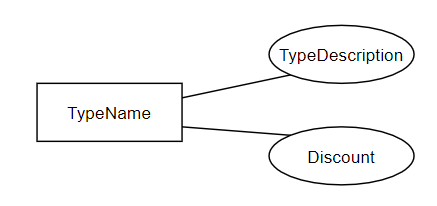


Рисунок 5.2.13 – Отношение Тип

1. Тип для сеанса (TypeName, TypeDescription):

* Отношение находится в 2НФ, так как оно находится в 1НФ и каждый не ключевой атрибут функционально полно зависит от потенциального ключа;
* Отношение находится в 3НФ, так как оно находится в 2НФ и отсутствуют транзитивные зависимости не ключевых атрибутов от ключевых;
* Отношение находится в БКНФ, так как каждая функциональная зависимость имеет в качестве своего детерминанта потенциальный ключ;

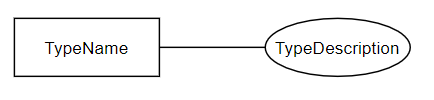


Рисунок 5.2.14 – Отношение Тип

1. СеансРеклама (SeanceID, AdID):

* Отношение находится в 2НФ, так как оно находится в 1НФ и каждый не ключевой атрибут функционально полно зависит от потенциального ключа;
* Отношение находится в 3НФ, так как оно находится в 2НФ и отсутствуют транзитивные зависимости не ключевых атрибутов от ключевых;
* Отношение находится в БКНФ, так как каждая функциональная зависимость имеет в качестве своего детерминанта потенциальный ключ;

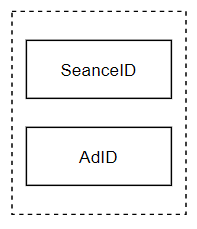


Рисунок 5.2.15 – Отношение СеансРеклама

**Построение схемы базы данных**

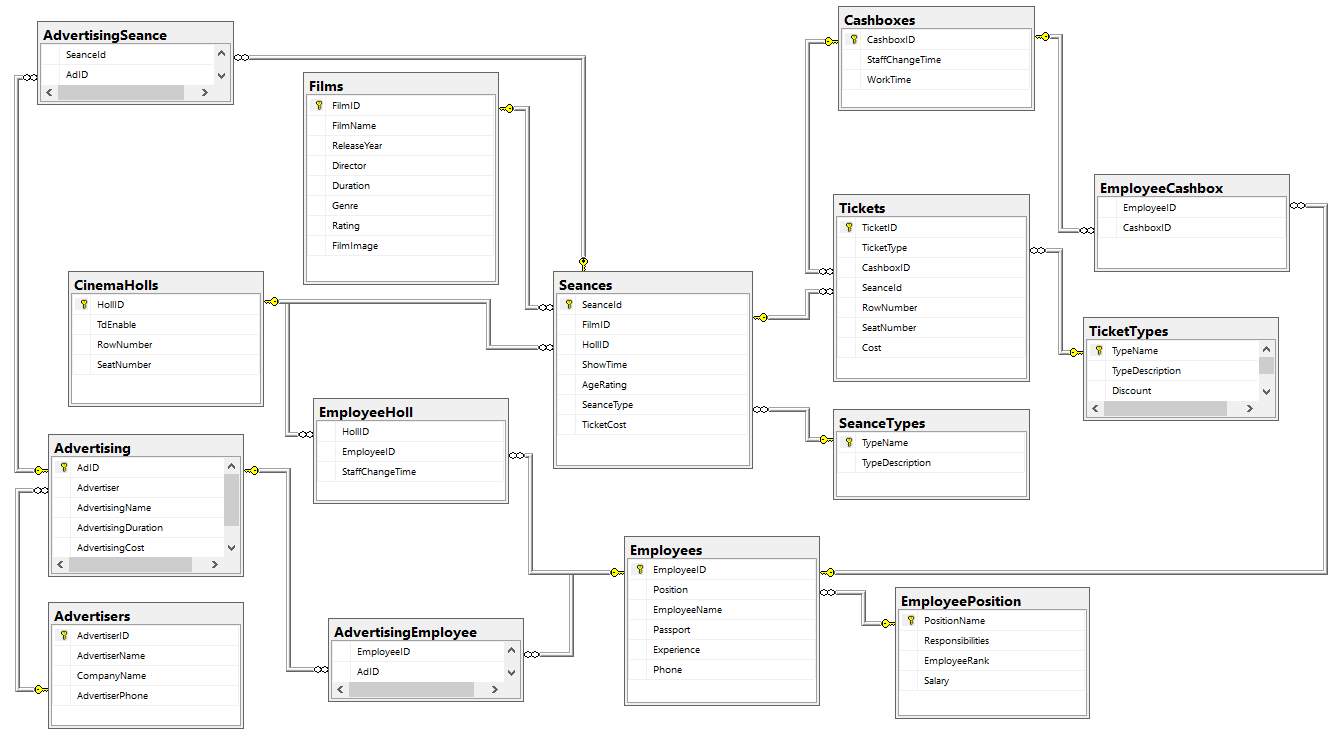


Рисунок 5.2.16- Схема базы данных

* 1. **Программирование объектов БД**

**Создание ограничений**

Ограничения – это специальные объекты в СУБД, которые позволяют задать допустимые значения для определенных столбцов с целью обеспечения автоматической безопасности базы данных.

В Microsoft SQL Server реализовано несколько типов ограничений, при этом пользователю доступно создание собственных правил.

Основные ограничения:

* [NOT] NULL – запрет или разрешение NULL значений в столбце;
* PRIMARY KEY – ограничение первичного ключа;
* FOREIGN KEY – ограничение внешнего ключа;
* UNIQUE – ограничение уникальности значения столбца;
* CHECK – проверка значения столбца на соответствие условию;
* DEFAULT – задание стандартного значения для столбца;

Задание пользовательских правил производится посредством команды CREATE RULE «name» AS («condition»). Чтобы привязать правило к столбцу таблицы нужно исполнить процедуру sp\_bindrule.

Чтобы создать пользовательские умолчания, необходимо использовать команду CREATE DEFAULT «» AS («»). Для привязки – sp\_bindefault.

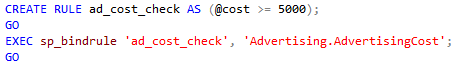
Данная комбинация команд будет использована для задания ограничений в некоторых таблицах.

**Ограничения, реализованные в базе данных:**

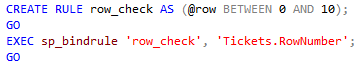
* Правило, ограничивающее рейтинг фильма:



* Правило, ограничивающее минимальную стоимость рекламы:



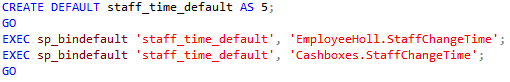
* Правило, ограничивающее номер ряда в кинозале для билета:



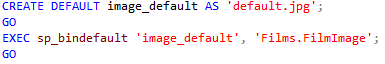
* Умолчание, задающее опыт сотрудника:



* Умолчание, задающее стандартное время смены персонала:



* Умолчание, задающее путь к стандартному изображению фильма:

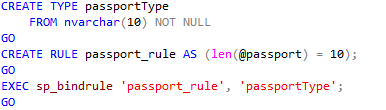


**Создание пользовательских типов**

Данные типы позволяют реализовывать пользовательскую логику на основе стандартных типов базы. Создание типа производится посредством команды CREATE TYPE «name» FROM \*type\* [NULL | NOT NULL]. На столбцы, обладающие пользовательским типом, так же можно навешивать ограничения.

Пользовательские типы, реализованные в базе данных:

* Паспорт, основанный на символьном типе:



* Номер телефона, основанный на символьном типе:



* Путь к изображению фильма, основанный на символьном типе:



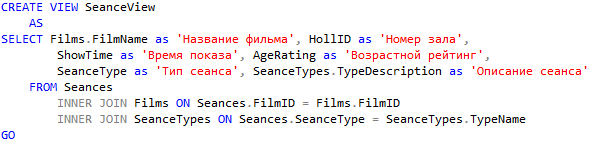
**Разработка представлений**

Представление — это виртуальная таблица, содержимое которой определяется запросом. Как и таблица, представление состоит из ряда именованных столбцов и строк данных. Пока представление не будет проиндексировано, оно не существует в базе данных как хранимая совокупность значений.

Для создания представления используется команда   
CREATE VIEW «name» AS SELECT…

Представления, разработанные для базы данных:

* Информация о сеансе:



Результаты приведены на рисунке 5.3.1.

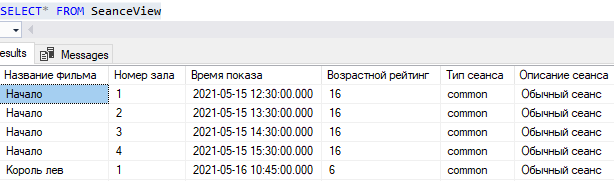
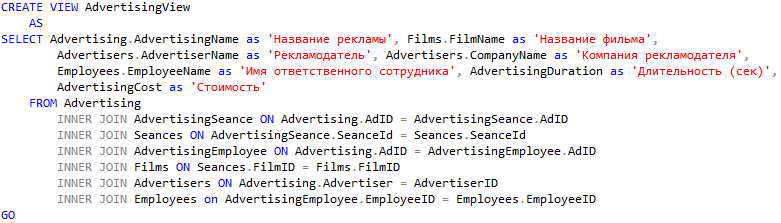


Рисунок 5.3.1 – Представление SeanceView

* Информация о рекламе:



Результаты приведены на рисунке 5.3.2.

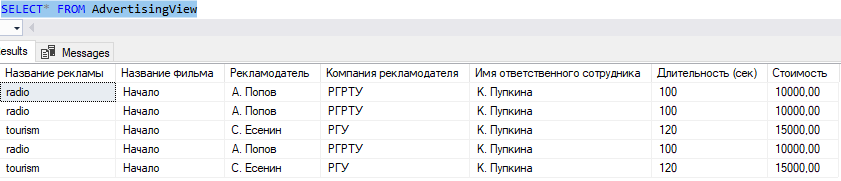
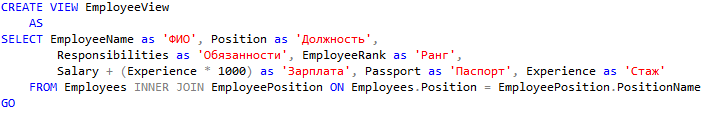


Рисунок 5.3.2 – Представление AdvertisingView

* Информация о сотрудниках:



Результаты приведены на рисунке 5.3.3.

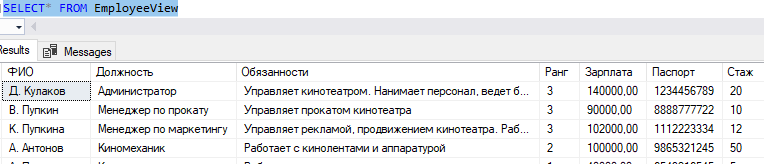


Рисунок 5.3.3 – Представление EmployeeView

**Разработка сценария создания БД**

Скрипт создания новой базы данных находится в Приложении 1.

**Разработка ХП, триггеров**

Хранимая процедура – это модуль, состоящий из SQL-операторов и хранящийся в БД на SQL-сервере. Вызывается с помощью команды EXEC.

Триггеры – это хранимая процедура, которая исполняется автоматически при возникновении определенных событий на сервере БД. Такими событиями могут являются попытки изменения данных в базе: INSERT, UPDATE, DELETE, которые применяются к таблицам или представлениям. В данной работе реализованы триггеры только для DML.  
 Есть два типа триггеров - INSERT OF и AFTER. Первый вызывается каждый раз заместо операций DML. Второй – после успешного исполнения операций DML.

Скрипты создания ХП и триггеров находятся в Приложении 2.

1. **Разработка клиентского приложения**

**6.1 Выбор программных компонентов клиентской части**

Клиентское приложение для данной курсовой работы было разработано в среде разработки Microsoft Visual Studio на языке С# с использованием фреймворков ASP.NET и ADO.NET.

Фреймворк ASP.NET обладает несколькими модулями для разработки веб-приложений. В работе был использован ASP.NET MVC, который предоставляет создавать веб-страницы с использованием шаблона проектирования MVC.

Платформа ASP.NET MVC базируется на взаимодействии трех компонентов: контроллера, модели и представления.

Контроллер принимает запросы, обрабатывает пользовательский ввод, взаимодействует с моделью и представлением и возвращает пользователю результат обработки запроса.

Модель представляет слой, описывающий логику организации данных в приложении. В модель могут быть включены несколько экземпляров разных классов, что позволяет использовать лишь одну модель для получения данных.

Представление получает данные из контроллера и генерирует элементы пользовательского интерфейса для отображения информации. Для этого применяется смесь языка C# и HTML – cshtml. Это позволяет получать и обрабатывать данные посредством как языка C#, так и JavaScript, а также отрисовывать страницы с помощью CSS.

Фреймворк ADO.NET - технология, предоставляющая доступ и управление данным, хранящимся в базе данных. В данной курсовой используется доступ к Microsoft SQL Server.

**6.2 Разработка интерфейса пользователя**

Для данной курсовой работы был разработан веб-сайт на фреймворке ASP.NET с использованием языков C#, JS, HTML, а также CSS.

Подключение к базе данных производится автоматически при запуске сайта.

После запуска сайта пользователю выводится информация о текущих фильмах в кинотеатре: изображение, название, режиссер и кнопка для перехода к сеансам на соответственный фильм. Стартовая страница приведена на рисунке 6.2.1

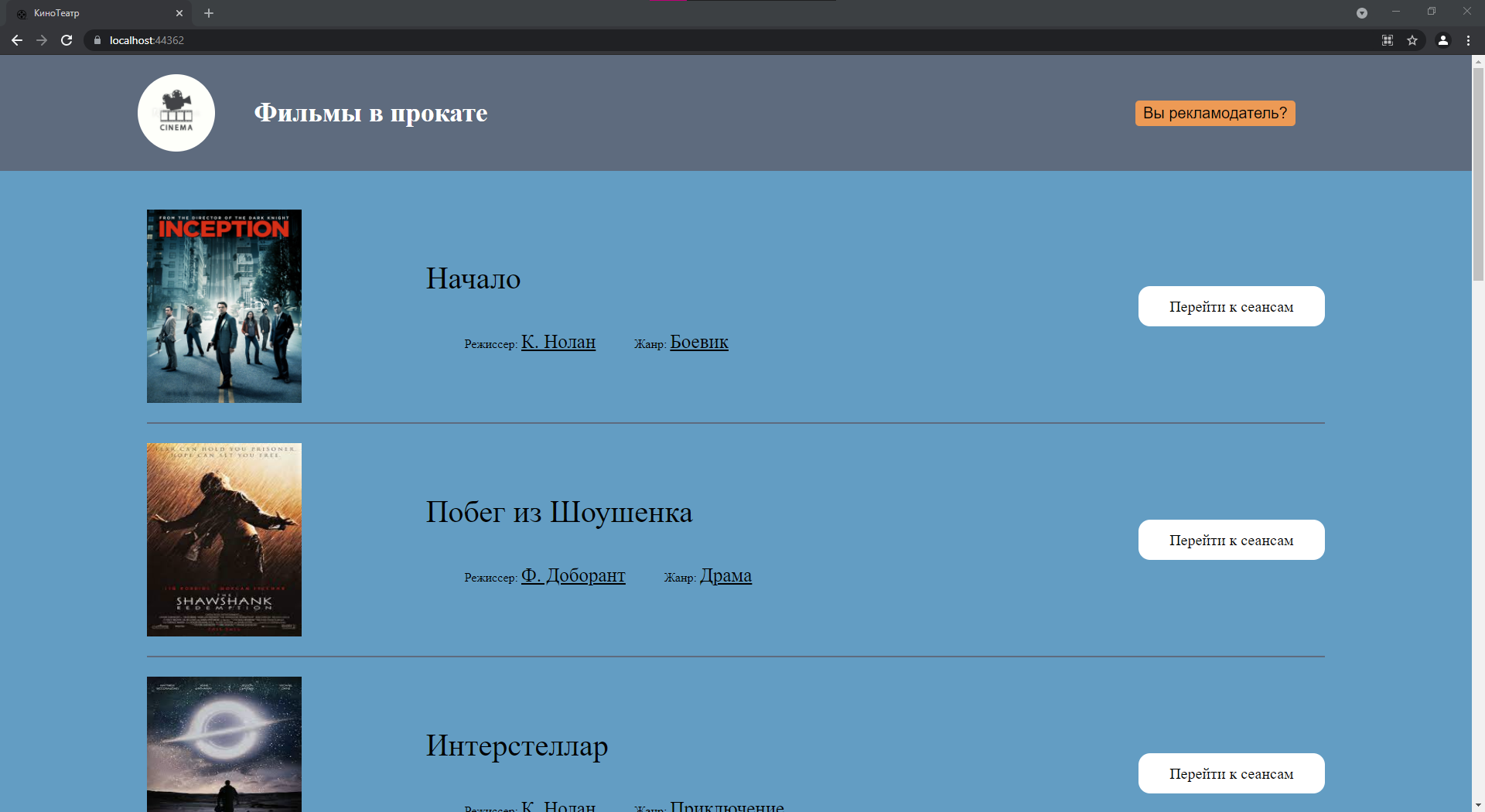


Рисунок 6.2.1 – Стартовая страница

После перехода к сеансам пользователь получает информацию о доступных сеансах, а также развернутую информацию о фильмах. Данная страница приведена на рисунке 6.2.2.

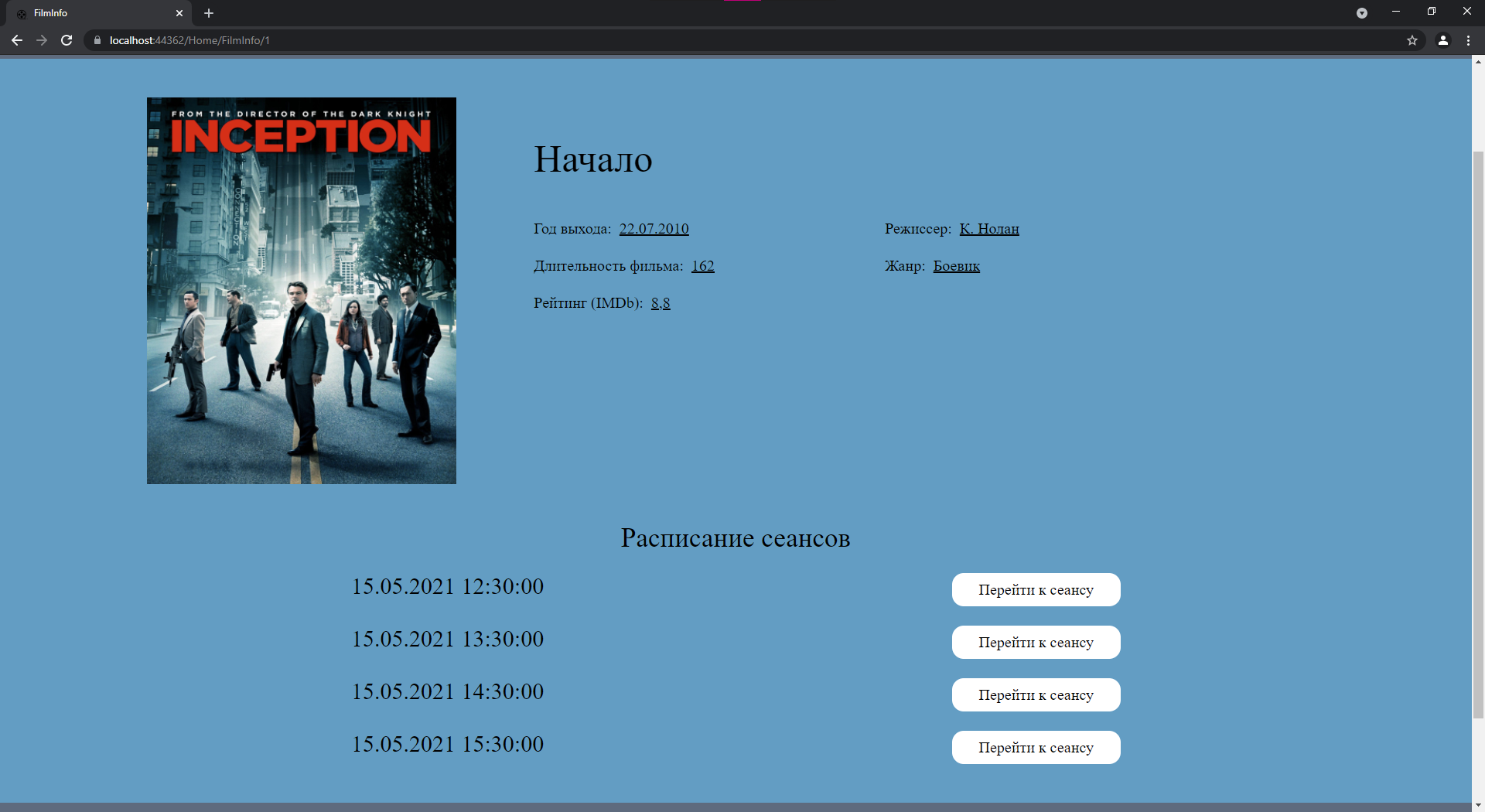


Рисунок 6.2.2 – Страница фильма

Далее, при выборе сеанса будет выведены данные о нем: доступные (полупрозрачные), выбранные (обведенные) и занятые (непрозрачные) места, информация о сеансе и кнопка покупки билета. Страница приведена на рисунке 6.2.3.

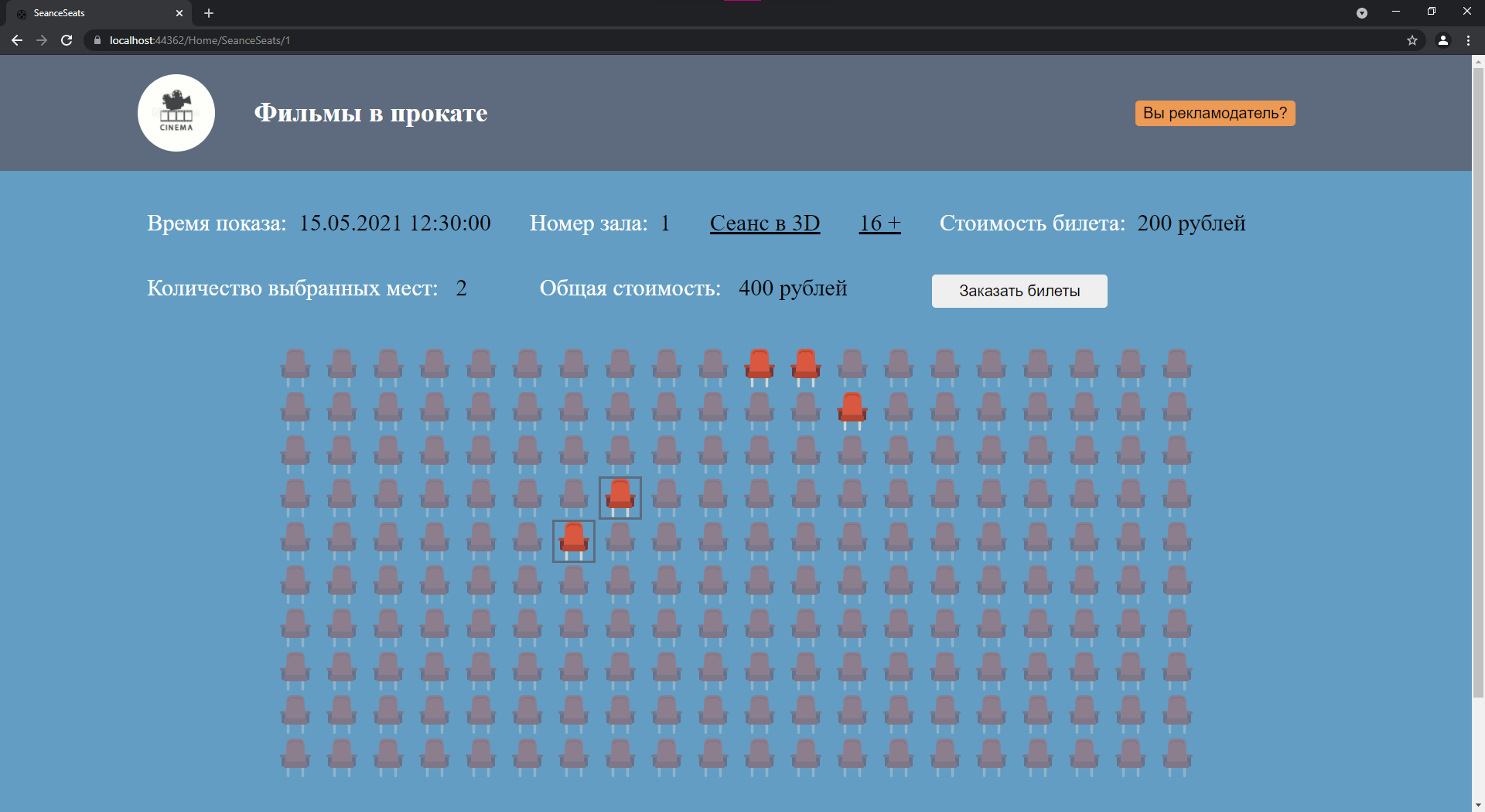


Рисунок 6.2.3 – Страница сеанса фильма

1. **Разработка программной документации**

**Назначение программы**

Программа представляет собой веб-приложение кинотеатра, позволяющее получать информацию о фильмах, сеансах, а также заказывать (бронировать) билеты на выбранные сеансы.

**Программные и аппаратные средства**

Для корректной работы веб-сайта в качестве аппаратного средства нужен ПК, обладающий следующими минимальными характеристиками:

* ОС: любая ОС;
* ОЗУ: не менее 1 Гб;
* Наличие интернет-соединения.

В качестве программного средства необходим любой современный браузер.

**Выполнение программы**

Для данной курсовой работы сайт не был помещен на хостинг в связи с проблемами установления работы базы данных. Поэтому запуск сайта может быть осуществлен только через среду разработки Microsoft Visual Studio.

Вся работа производится в браузере.

Для завершения работы можно либо закрыть браузер, либо остановить отладку в Microsoft Visual Studio.

1. **Тестирование ИС**

**Тестирование ограничений**

* Правило, ограничивающее минимальную стоимость рекламы:

Тестирование проводится методом черного ящика с помощью классов эквивалентности. Для проверки используется два класса:

1. Стоимость заходится за разрешенным пределом (меньше 5000):

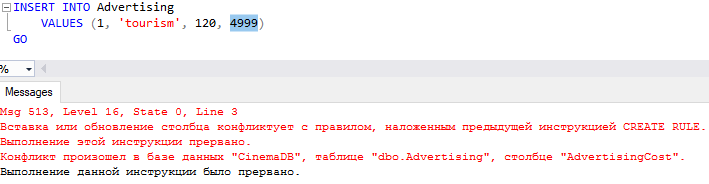


Рисунок 8.3 – Ошибка выполнения вставки

1. Стоимость находится в разрешенных пределах (больше 5000):

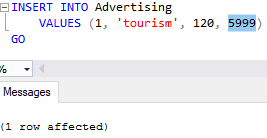


Рисунок 8.1 – Успешное выполнение

* Умолчание, задающее путь к стандартному изображению файла:

Пропущено значение изображения пути для фильма. Ожидается ‘default.jpg’.

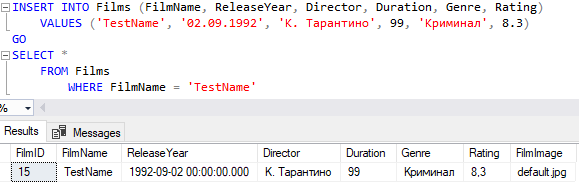
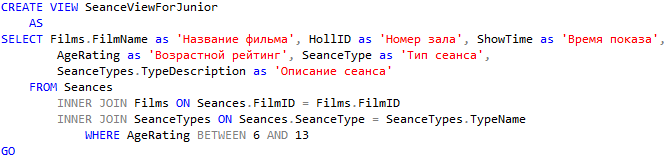


Рисунок 8.2 – Результат вставки данных

**Тестирование представлений**

* Тестирование представления, выводящего информацию о сеансах с возрастным рейтингом от 6 до 13:



Тестирование будет проводиться методом черного ящика с помощью классов эквивалентности.

Для проверки будет использоваться два класса эквивалентности:

1. Значение возрастного рейтинга находится в заданных границах:

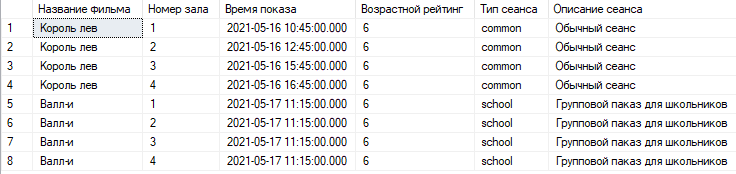


Рисунок 8.3 – Результат выборки

1. Значение возрастного рейтинга находится за пределами границ:

В этом случае данные о сеансах не попадут в выборку.

**Тестирование ХП и триггеров**

* Тестирование процедуры вставки сеанса:

Тестирование будет проводиться методом белого ящика.

1. Формирование потокового графа:

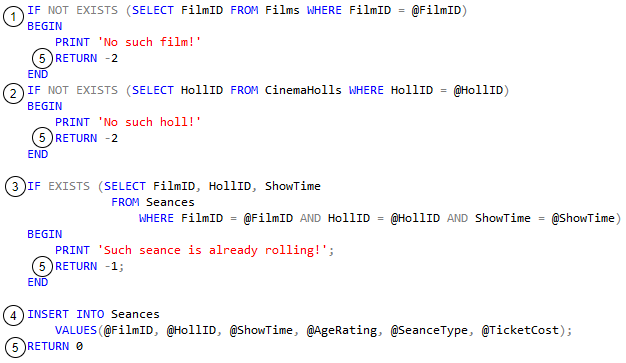


Рисунок 8.4 – Пронумерованные операторы текста

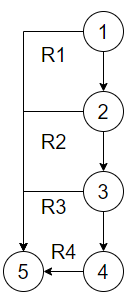


Рисунок 8.5 – Потоковый граф

1. Определение цикломатической сложности потокового графа:
2. По числу регионов – 4;
3. По формуле = 15 – 13 + 2 = 4;
4. По выражению ;
5. Определение базового множества независимых путей в графе:
6. 1 – 5;
7. 1 – 2 – 5;
8. 1 – 2 – 3 – 5;
9. 1 – 2 – 3 – 4 – 5;
10. Подготовление тестовых вариантов:
11. 1 Путь. Выбранного фильма нет на складе:

ИД1 - Будет введен несуществующий в базе номер фильма.

ОР1 - Процедура вернет значение -2. Данные не будут вставлены.

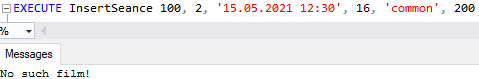


Рисунок 8.6 – Результат исполнения процедуры

1. 2 Путь. Некорректное значение выбранного зала:

ИД2 – Будет введен несуществующий в базе номер кинозала.

ОР2 – Процедура вернет значение -2. Данные не будут вставлены.

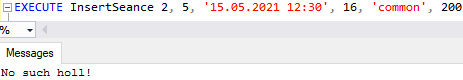


Рисунок 8.7 – Результат исполнения процедуры

1. 3 Путь. Сеанс уже прокатывается:

ИД3 – Будут введены значения сеанса, который уже прокатывается в кинотеатре.

ОР3 – Процедура вернет значение -1. Данные не будут вставлены.

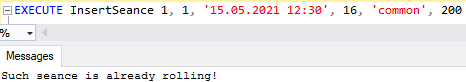


Рисунок 8.8 – Результат исполнения процедуры

1. 4 Путь. Новые корректные данные:

ИД4 – Будут введены корректные значения для не прокатываемого фильма (на текущем времени).

ОР4 – Процедура вернет 0. Данные будут вставлены.

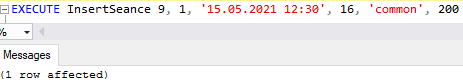


Рисунок 8.9 – Результат исполнения процедуры

1. Определение степени покрытия логики программы тестовыми вариантами:

* Тестирование триггера, проверяющего корректность значений занятых мест в билете в соответствии с кинозалом:

Тестирование будет проводиться методом белого ящика.

1. Формирование потокового графа:

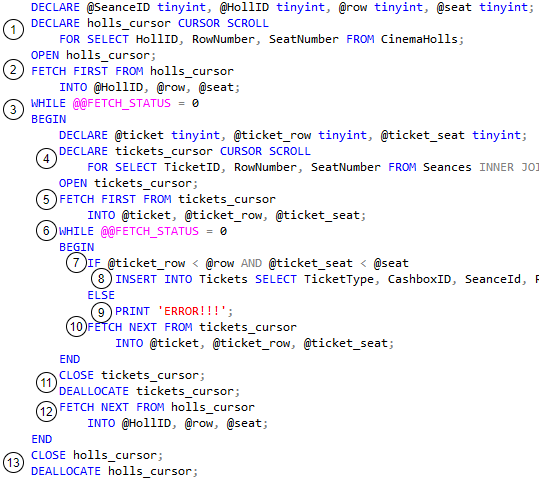


Рисунок 8.10 – Пронумерованные операторы текста

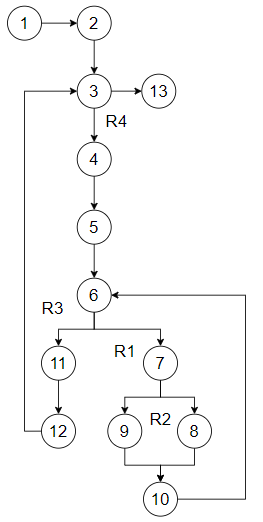


Рисунок 8.11– Потоковый граф

1. Определение цикломатической сложности потокового графа:
2. По числу регионов – 4;
3. По формуле = 15 – 13 + 2 = 4;
4. По выражению ;
5. Определение базового множества независимых путей в графе:
6. 1 – 2 – 3 – 13;
7. 1 – 2 – 3 – 4 – 5 – 6 – 11 – 12 – 3 – 13;
8. 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 10 – 6 – 11 – 12 – 3 – 13;
9. 1 – 2 – 3 – 4 – 5 – 6 – 7 – 9 – 10 – 6 – 11 – 12 – 3 – 13;
10. Подготовление тестовых вариантов:

Путь 1: ИД1 – В базе нет данных о кинозалах; ОР1 – Корректное завершение, данные о билетах в базу не вставлены.

Путь 2: ИД2 – Нет билетов для текущего зала; ОР2 – Корректное завершение, данные о билетах для зала не вставлены в базу.

Путь 3: ИД3 – Есть билеты для текущего зала, данные корректны;   
ОР3 – Корректное завершение, данные о билетах вставлены в базу.

Путь 4: ИД4 – Есть билеты для текущего зала, данные некорректны;   
ОР4 – Корректное завершение, выдача сообщения, данные не вставлены в базу.

1. Определение степени покрытия логики программы тестовыми вариантами:

**Заключение**

По итогу данного курсового проекта было произведено проектирование системы кинотеатра. Была составлена база данных, содержащая информацию об основных объектах кинотеатра, таких как фильмы, сеансы, сотрудники и др. А также разработано веб-приложение, которое, в ограниченном формате, позволяет производить действия над базой данных, а именно выборку данных и модификацию таблиц(ы). Кроме того, было произведено тестирование созданных объектов базы данных.

Данная работа позволила получить новые и улучшить уже имеющиеся навыки работы с базами данных, анализом исследуемой системы, проектированием и реализацией программной части проекта, проведением тестирования и составления программной документации.

**Список использованной литературы**

1. Гринченко Н.Н. Конспект лекций по дисциплине «Базы данных и клиент-серверные приложения», 2021.
2. Райордан Р. Основы реляционных баз данных. - Русская Редакция, 2001. - 390 с.
3. ГОСТ 19.503-79 ЕСПД. Руководство системного программиста. Требования к содержанию и оформлению.

**ПРИЛОЖЕНИЕ А: сценарий создания объектов БД.**

--creating db

USE master

CREATE DATABASE CinemaDB

ON PRIMARY (

NAME = N'Cinema',

FILENAME = N'D:\CinemaDB\Cinema.mdf' ,

SIZE = 8192KB ,

FILEGROWTH = 65536KB

)

LOG ON (

NAME = N'Cinema\_log',

FILENAME = N'D:\CinemaDB\Cinema\_log.ldf' ,

SIZE = 8192KB ,

FILEGROWTH = 65536KB

)

GO

--switch to CinemaDB

USE CinemaDB

GO

--user-defined data types

CREATE TYPE passportType

FROM nvarchar(10) NOT NULL

GO

CREATE RULE passport\_rule AS (len(@passport) = 10);

GO

EXEC sp\_bindrule 'passport\_rule', 'passportType';

GO

CREATE TYPE phoneType

FROM nvarchar(10) NOT NULL

GO

CREATE TYPE imagePath

FROM nvarchar(255) NOT NULL

GO

--starting creating tables

--collection of films

CREATE TABLE dbo.Films (

FilmID tinyint PRIMARY KEY IDENTITY(1, 1), -- max 50 films at one time

FilmName nvarchar(30) NOT NULL,

ReleaseYear datetime NOT NULL,

Director nvarchar(50) NOT NULL,

Duration smallint,

Genre nvarchar(20),

Rating float,

FilmImage imagePath

)

GO

--employee's position

CREATE TABLE dbo.EmployeePosition (

PositionName nvarchar(30) PRIMARY KEY,

Responsibilities nvarchar(255) NOT NULL,

EmployeeRank tinyint NOT NULL, -- 1 - service staff ,2 - common employee, 3 - managment

Salary money NOT NULL

)

GO

--employees

CREATE TABLE dbo.Employees (

EmployeeID tinyint PRIMARY KEY IDENTITY(1, 1),

Position nvarchar(30) FOREIGN KEY REFERENCES EmployeePosition(PositionName),

EmployeeName nvarchar(50) NOT NULL,

Passport passportType UNIQUE NOT NULL, -- create user type

Experience tinyint,

Phone phoneType

)

GO

--advertisers

CREATE TABLE dbo.Advertisers (

AdvertiserID tinyint PRIMARY KEY IDENTITY(1, 1),

AdvertiserName nvarchar(50) NOT NULL,

CompanyName nvarchar(50) NOT NULL,

AdvertiserPhone phoneType

)

GO

--cinemaholl

CREATE TABLE dbo.CinemaHolls (

HollID tinyint PRIMARY KEY IDENTITY(1, 1),

TdEnable bit DEFAULT 0,

RowNumber tinyint NOT NULL,

SeatNumber tinyint NOT NULL--max(10x20) - avr(10x15)

)

GO

--seance types

CREATE TABLE dbo.SeanceTypes (

TypeName nvarchar(20) PRIMARY KEY,

TypeDescription nvarchar(100) NOT NULL

)

GO

--seance

CREATE TABLE dbo.Seances (

SeanceId tinyint PRIMARY KEY IDENTITY(1, 1),

FilmID tinyint FOREIGN KEY REFERENCES Films(FilmID),

HollID tinyint FOREIGN KEY REFERENCES CinemaHolls(HollID),

ShowTime datetime NOT NULL, -- starting

AgeRating tinyint,

SeanceType nvarchar(20) FOREIGN KEY REFERENCES SeanceTypes(TypeName),

TicketCost money NOT NULL

)

GO

--advertising

CREATE TABLE dbo.Advertising (

AdID tinyint PRIMARY KEY IDENTITY(1, 1),

Advertiser tinyint FOREIGN KEY REFERENCES Advertisers(AdvertiserID),

AdvertisingName nvarchar(20) NOT NULL,

AdvertisingDuration tinyint CHECK (AdvertisingDuration <= 180) NOT NULL,

AdvertisingCost money NOT NULL

)

GO

CREATE TABLE dbo.AdvertisingEmployee (

EmployeeID tinyint FOREIGN KEY REFERENCES Employees(EmployeeID),

AdID tinyint FOREIGN KEY REFERENCES Advertising(AdID)

)

GO

CREATE TABLE dbo.AdvertisingSeance (

SeanceId tinyint FOREIGN KEY REFERENCES Seances(SeanceId),

AdID tinyint FOREIGN KEY REFERENCES Advertising(AdID)

)

--EmployeeHoll

CREATE TABLE dbo.EmployeeHoll (

HollID tinyint FOREIGN KEY REFERENCES CinemaHolls(HollID),

EmployeeID tinyint FOREIGN KEY REFERENCES Employees(EmployeeID),

StaffChangeTime tinyint

)

GO

--ticket type

CREATE TABLE dbo.TicketTypes (

TypeName nvarchar(20) PRIMARY KEY,

TypeDescription nvarchar(100) NOT NULL,

Discount tinyint DEFAULT 0

)

GO

--cashbox

CREATE TABLE dbo.Cashboxes (

CashboxID tinyint PRIMARY KEY IDENTITY(0, 1),

StaffChangeTime tinyint,

WorkTime tinyint DEFAULT 12

)

GO

CREATE TABLE dbo.EmployeeCashbox (

EmployeeID tinyint FOREIGN KEY REFERENCES Employees(EmployeeID),

CashboxID tinyint FOREIGN KEY REFERENCES Cashboxes(CashboxID),

)

--ticket

CREATE TABLE dbo.Tickets (

TicketID smallint PRIMARY KEY IDENTITY(1, 1),

TicketType nvarchar(20) FOREIGN KEY REFERENCES TicketTypes(TypeName),

CashboxID tinyint FOREIGN KEY REFERENCES Cashboxes(CashboxID),

SeanceId tinyint FOREIGN KEY REFERENCES Seances(SeanceId),

RowNumber tinyint NOT NULL, --constraints!

SeatNumber tinyint CHECK(SeatNumber BETWEEN 0 AND 20) NOT NULL,

Cost money NOT NULL

)

GO

--constraints

-- rules

CREATE RULE rating\_check AS (@rating BETWEEN 0 AND 10);

GO

EXEC sp\_bindrule 'rating\_check', 'Films.Rating';

GO

CREATE RULE ad\_cost\_check AS (@cost >= 5000);

GO

EXEC sp\_bindrule 'ad\_cost\_check', 'Advertising.AdvertisingCost';

GO

CREATE RULE row\_check AS (@row BETWEEN 0 AND 10);

GO

EXEC sp\_bindrule 'row\_check', 'Tickets.RowNumber';

GO

-- default

CREATE DEFAULT experience\_default AS 0;

GO

EXEC sp\_bindefault 'experience\_default', 'Employees.Experience';

GO

CREATE DEFAULT staff\_time\_default AS 5;

GO

EXEC sp\_bindefault 'staff\_time\_default', 'EmployeeHoll.StaffChangeTime';

EXEC sp\_bindefault 'staff\_time\_default', 'Cashboxes.StaffChangeTime';

GO

CREATE DEFAULT image\_default AS 'default.jpg';

GO

EXEC sp\_bindefault 'image\_default', 'Films.FilmImage';

GO

--triggers

-- when delete seance, remove all fit tickets ADD CURSOR (cancellation of the session)

CREATE TRIGGER dbo.RemoveTickets ON Seances

AFTER delete

AS

DECLARE @deletingSeance tinyint;

DECLARE seances\_cursor CURSOR SCROLL

FOR SELECT SeanceId FROM deleted;

OPEN seances\_cursor;

FETCH FIRST FROM seances\_cursor

INTO @deletingSeance;

WHILE @@FETCH\_STATUS = 0

BEGIN

DELETE FROM Tickets WHERE SeanceId = @deletingSeance;

FETCH NEXT FROM seances\_cursor

INTO @deletingSeance;

END

CLOSE seances\_cursor;

DEALLOCATE seances\_cursor;

GO

-- checking the seating in the hall

-- how it works: reading all holls (getting id, rows and seat). then read all tickets for this holl and read the same. check values.

CREATE TRIGGER dbo.CheckingCorrectSeatsInTicket ON Tickets

INSTEAD OF insert

AS

DECLARE @SeanceID tinyint, @HollID tinyint, @row tinyint, @seat tinyint;

DECLARE holls\_cursor CURSOR SCROLL

FOR SELECT HollID, RowNumber, SeatNumber FROM CinemaHolls;

OPEN holls\_cursor;

FETCH FIRST FROM holls\_cursor

INTO @HollID, @row, @seat;

WHILE @@FETCH\_STATUS = 0

BEGIN

DECLARE @ticket tinyint, @ticket\_row tinyint, @ticket\_seat tinyint;

DECLARE tickets\_cursor CURSOR SCROLL

FOR SELECT TicketID, RowNumber, SeatNumber FROM Seances INNER JOIN inserted ON Seances.SeanceId = inserted.SeanceId WHERE Seances.HollID = @HollID;

OPEN tickets\_cursor;

FETCH FIRST FROM tickets\_cursor

INTO @ticket, @ticket\_row, @ticket\_seat;

WHILE @@FETCH\_STATUS = 0

BEGIN

IF @ticket\_row < @row AND @ticket\_seat < @seat

INSERT INTO Tickets SELECT TicketType, CashboxID, SeanceId, RowNumber, SeatNumber, Cost FROM inserted WHERE TicketID = @ticket;

ELSE

PRINT 'ERROR!!!';

FETCH NEXT FROM tickets\_cursor

INTO @ticket, @ticket\_row, @ticket\_seat;

END

CLOSE tickets\_cursor;

DEALLOCATE tickets\_cursor;

FETCH NEXT FROM holls\_cursor

INTO @HollID, @row, @seat;

END

CLOSE holls\_cursor;

DEALLOCATE holls\_cursor;

GO

-- advesting check

--how it works: if general duration of inserting and existing ad for seance more 600, it is skipping

CREATE TRIGGER dbo.ChecingDurationForSeance ON AdvertisingSeance

INSTEAD OF insert

AS

DECLARE @id tinyint, @SeanceId tinyint, @Duration tinyint;

DECLARE advertising\_cursor CURSOR SCROLL

FOR SELECT AdID, SeanceId FROM inserted;

OPEN advertising\_cursor;

FETCH FIRST FROM advertising\_cursor

INTO @id, @SeanceId;

WHILE @@FETCH\_STATUS = 0

BEGIN

SELECT @Duration = AdvertisingDuration FROM Advertising WHERE AdID = @id;

DECLARE @checkingDuration smallint;

IF NOT EXISTS(SELECT\* FROM AdvertisingSeance)

SET @checkingDuration = 0;

ELSE

SELECT @checkingDuration = SUM(AdvertisingDuration) FROM Advertising INNER JOIN AdvertisingSeance on Advertising.AdID = AdvertisingSeance.AdID GROUP BY SeanceId HAVING SeanceId = 1;

IF @Duration + @checkingDuration < 600

INSERT INTO AdvertisingSeance VALUES (@id, @SeanceId);

ELSE

PRINT 'error with duration. too long!';

FETCH NEXT FROM advertising\_cursor

INTO @id, @SeanceId;

END

CLOSE advertising\_cursor;

DEALLOCATE advertising\_cursor;

GO

--checking seance show time (comparing show time and film duration)

--how is works: get last time in holl and check inserting and its time

CREATE TRIGGER dbo.CheckingSeanceShowTime ON Seances

AFTER insert

AS

DECLARE @seanceID tinyint, @HollID tinyint, @FilmID tinyint, @insert\_time datetime, @closest\_time datetime;

DECLARE showtime\_cursor CURSOR SCROLL

FOR SELECT SeanceId, HollID, FilmID, ShowTime FROM inserted;

OPEN showtime\_cursor;

FETCH FIRST FROM showtime\_cursor

INTO @seanceID, @HollID, @FilmID, @insert\_time;

WHILE @@FETCH\_STATUS = 0

BEGIN

DECLARE @film\_duration smallint;

SET @closest\_time = (SELECT TOP(1) ShowTime FROM Seances WHERE HollID = @HollID ORDER BY ShowTime DESC);

SELECT @film\_duration = Duration FROM Films WHERE FilmID = @FilmID;

IF CONVERT(date, @closest\_time) < CONVERT(date, @insert\_time) OR (CONVERT(date, @closest\_time) = CONVERT(date, @insert\_time) AND DATEDIFF(MINUTE, DATEADD(DAY, DATEDIFF(DAY, 0, @closest\_time), 0), @closest\_time) < DATEDIFF(MINUTE, DATEADD(DAY, DATEDIFF(DAY, 0, @insert\_time), 0), @insert\_time))

BEGIN

delete from Seances WHERE SeanceId = @seanceID;

PRINT 'ERROR';

END

FETCH NEXT FROM showtime\_cursor

INTO @seanceID, @HollID, @FilmID, @insert\_time;

END

CLOSE showtime\_cursor;

DEALLOCATE showtime\_cursor;

GO

-- insert procedures

-- films

CREATE PROCEDURE InsertFilm (@FilmName nvarchar(30), @ReleaseYear datetime, @Director nvarchar(50), @Duration smallint, @Genre nvarchar(20), @Rating tinyint, @image nvarchar(255))

AS

BEGIN

IF EXISTS(SELECT FilmName, ReleaseYear, Director

FROM Films

WHERE FilmName = @FilmName and ReleaseYear = @ReleaseYear and Director = @Director)

BEGIN

PRINT 'Such a film is already in the collection!';

RETURN -1;

END

INSERT INTO Films

VALUES(@FilmName, @ReleaseYear, @Director, @Duration, @Genre, @Rating, @image);

RETURN 0

END

GO

-- seances

CREATE PROCEDURE InsertSeance (@FilmID tinyint, @HollID tinyint, @ShowTime datetime, @AgeRating tinyint, @SeanceType nvarchar(20), @TicketCost tinyint)

AS

BEGIN

IF NOT EXISTS (SELECT FilmID FROM Films WHERE FilmID = @FilmID)

BEGIN

PRINT 'No such film!'

RETURN -2

END

IF NOT EXISTS (SELECT HollID FROM CinemaHolls WHERE HollID = @HollID)

BEGIN

PRINT 'No such holl!'

RETURN -2

END

IF EXISTS (SELECT FilmID, HollID, ShowTime

FROM Seances

WHERE FilmID = @FilmID AND HollID = @HollID AND ShowTime = @ShowTime)

BEGIN

PRINT 'Such seance is already rolling!';

RETURN -1;

END

INSERT INTO Seances

VALUES(@FilmID, @HollID, @ShowTime, @AgeRating, @SeanceType, @TicketCost);

RETURN 0

END

GO

-- employees

CREATE PROCEDURE InsertEmployee (@Position nvarchar(20), @EmployeeName nvarchar(50), @Passport passportType, @Expirience tinyint, @Phone phoneType)

AS

BEGIN

IF NOT EXISTS(SELECT PositionName FROM EmployeePosition WHERE PositionName = @Position)

BEGIN

PRINT 'No such position!'

RETURN -2

END

IF EXISTS(SELECT Passport

FROM Employees

WHERE Passport = @Passport)

BEGIN

PRINT 'The employee already exists in the database!';

RETURN -1;

END

INSERT INTO Employees (Position, EmployeeName, Passport, Experience, Phone)

VALUES(@Position, @EmployeeName, @Passport, @Expirience, @Phone);

RETURN 0

END

GO

-- exec InsertEmployee 'Кассир', 'Martin', '1234567890', 0

-- advertisers

CREATE PROCEDURE InsertAdvertiser (@AdvertiserName nvarchar(50), @CompanyName nvarchar(50), @Phone phoneType)

AS

BEGIN

IF EXISTS(SELECT AdvertiserName, CompanyName, AdvertiserPhone

FROM Advertisers

WHERE AdvertiserName = @AdvertiserName and CompanyName = @CompanyName and AdvertiserPhone = @Phone)

BEGIN

PRINT 'This advertiser is already buying services from us!';

RETURN -1;

END

INSERT INTO Advertisers

VALUES(@AdvertiserName, @CompanyName, @Phone);

RETURN 0

END

GO

-- exec InsertAdvertiser 'Р. Кирилин', 'ПоЛИтех'

-- advertising || just adding advertising, setting on seance in another procedure

CREATE PROCEDURE InsertAdvertising (@Employee tinyint, @Advertiser tinyint, @AdvertisingName nvarchar(20), @AdvertisingDuration tinyint, @AdvertisingCost smallint)

AS

BEGIN

IF NOT EXISTS(SELECT EmployeeID, EmployeeRank FROM Employees INNER JOIN EmployeePosition ON Employees.Position = EmployeePosition.PositionName WHERE EmployeeID = @Employee AND EmployeeRank = 3)

BEGIN

PRINT 'No such employee or his rank is not suitable!'

RETURN -2

END

IF NOT EXISTS(SELECT AdvertiserID FROM Advertisers WHERE AdvertiserID = @Advertiser)

BEGIN

PRINT 'Unknown advertiser!'

RETURN -2

END

IF EXISTS(SELECT Advertiser, AdvertisingName

FROM Advertising

WHERE Advertiser = @Advertiser and AdvertisingName = @AdvertisingName)

BEGIN

PRINT 'Such advertising is already rolling!';

RETURN -1;

END

INSERT INTO Advertising

VALUES(@Advertiser, @AdvertisingName, @AdvertisingDuration, @AdvertisingCost);

INSERT INTO AdvertisingEmployee

VALUES(@Employee, (SELECT AdID FROM Advertising WHERE Advertiser = @Advertiser and AdvertisingName = @AdvertisingName))

RETURN 0

END

GO

--exec InsertAdvertising 3, 1, 'some', 180, 5000

-- ticket

CREATE PROCEDURE CreateTicket (@SeanceId tinyint, @TypeName nvarchar(20), @CashboxID tinyint, @RowNumber tinyint, @SeatNumber tinyint, @Cost smallint)

AS

BEGIN

IF NOT EXISTS(SELECT SeanceId FROM Seances WHERE SeanceId = @SeanceId)

BEGIN

PRINT 'No such seance!'

RETURN -2

END

IF NOT EXISTS(SELECT TypeName FROM TicketTypes WHERE TypeName = @TypeName)

BEGIN

PRINT 'No such ticket type!'

RETURN -2

END

IF NOT EXISTS(SELECT CashboxID FROM Cashboxes WHERE CashboxID = @CashboxID)

BEGIN

PRINT 'No such cashbox!'

RETURN -2

END

IF EXISTS(SELECT SeanceId, RowNumber, SeatNumber

FROM Tickets

WHERE SeanceId = @SeanceId and RowNumber = @RowNumber and SeatNumber = @SeatNumber)

BEGIN

PRINT 'This seat is already taken!';

RETURN -1;

END

INSERT INTO Tickets

VALUES(@TypeName, @CashboxID, @SeanceId, @RowNumber, @SeatNumber, @Cost);

RETURN 0

END

GO

-- exec CreateTicket 1, 'standart', 0, 9, 15, 200

-- IF NOT EXISTS(SELECT FROM WHERE = @)

-- BEGIN

-- PRINT '!'

-- RETURN -2

-- END

CREATE PROCEDURE GetAllFilms

AS

BEGIN

SELECT\* FROM Films

END

GO

CREATE PROCEDURE GetTicketsForSeance (@seance tinyint)

AS

BEGIN

SELECT\* FROM Tickets WHERE SeanceID = @seance

END

GO

CREATE PROCEDURE GetSeanceInfo (@seance tinyint)

AS

BEGIN

SELECT\* FROM Seances WHERE SeanceID = @seance

END

GO

CREATE PROCEDURE GetFilmById (@film tinyint)

AS

BEGIN

SELECT\* FROM Films WHERE FilmID = @film

END

GO

-- select\* from Tickets

-- insert into Tickets values('standart', 0, 1, 5, 6, 200)

CREATE PROCEDURE GetHollInfoBySeance (@seance tinyint)

AS

BEGIN

SELECT CinemaHolls.HollID, TdEnable, RowNumber, SeatNumber

FROM CinemaHolls INNER JOIN Seances ON CinemaHolls.HollID = Seances.HollID

WHERE Seances.SeanceId = @seance

END

GO

--view with seance info

CREATE VIEW SeanceViewForJunior

AS

SELECT Films.FilmName as 'Название фильма', HollID as 'Номер зала', ShowTime as 'Время показа', AgeRating as 'Возрастной рейтинг', SeanceType as 'Тип сеанса', SeanceTypes.TypeDescription as 'Описание сеанса'

FROM Seances

INNER JOIN Films ON Seances.FilmID = Films.FilmID

INNER JOIN SeanceTypes ON Seances.SeanceType = SeanceTypes.TypeName

WHERE AgeRating BETWEEN 6 AND 13

GO

--view about advertising

CREATE VIEW AdvertisingView

AS

SELECT Advertising.AdvertisingName as 'Название рекламы', Films.FilmName as 'Название фильма', Advertisers.AdvertiserName as 'Рекламодатель', Advertisers.CompanyName as 'Компания рекламодателя', Employees.EmployeeName as 'Имя ответственного сотрудника', AdvertisingDuration as 'Длительность (сек)', AdvertisingCost as 'Стоимость'

FROM Advertising

INNER JOIN AdvertisingSeance ON Advertising.AdID = AdvertisingSeance.AdID

INNER JOIN Seances ON AdvertisingSeance.SeanceId = Seances.SeanceId

INNER JOIN AdvertisingEmployee ON Advertising.AdID = AdvertisingEmployee.AdID

INNER JOIN Films ON Seances.FilmID = Films.FilmID

INNER JOIN Advertisers ON Advertising.Advertiser = AdvertiserID

INNER JOIN Employees on AdvertisingEmployee.EmployeeID = Employees.EmployeeID

GO

--view about employees

CREATE VIEW EmployeeView

AS

SELECT EmployeeName as 'ФИО', Position as 'Должность', Responsibilities as 'Обязанности', EmployeeRank as 'Ранг', Salary + (Experience \* 1000) as 'Зарплата', Passport as 'Паспорт', Experience as 'Стаж'

FROM Employees INNER JOIN EmployeePosition ON Employees.Position = EmployeePosition.PositionName

GO

ПРИЛОЖЕНИЕ Б: сценарий заполнения таблиц БД.

USE CinemaDB

GO

-- filling in start tables

--films

INSERT INTO Films VALUES

('Начало', '22.07.2010', 'К. Нолан', 162, 'Боевик', 8.8, 'inception.jpg'),

('Побег из Шоушенка', '14.10.1994', 'Ф. Доборант', 144, 'Драма', 9.3, 'shawshank.jpg'),

('Интерстеллар', '07.11.2014', 'К. Нолан', 169, 'Приключение', 8.6, 'interstellar.jpg'),

('Матрица', '31.03.1999', 'Братья Вачовски', 136, 'Боевик', 8.7, 'matrix.jpg'),

('Король лев', '24.06.1994', 'Р. Аллерс', 88, 'Анимационный', 8.5, 'lionking.jpg'),

('Чужой', '22.06.1979', 'Р. Скотт', 117, 'Ужасы', 8.4, 'alien.jpg'),

('Валл-и', '27.06.2008', 'А. Стэнтон', 98, 'Анимационный', 8.4, 'walle.jpg'),

('Джокер', '04.10.2019', 'Т. Филлипс', 122, 'Драма', 8.4, 'joker.jpg'),

('Мстители: Война бесконечности', '27.04.2018', 'Братья Руссо', 160, 'Боевик', 8.4, 'avengers.jpg'),

('Бешеные псы', '02.09.1992', 'К. Тарантино', 99, 'Криминал', 8.3, 'reservoirdogs.jpg')

GO

--employeePositions

INSERT INTO EmployeePosition VALUES

('Администратор', 'Управляет кинотеатром. Нанимает персонал, ведет бухгалтерскую работу', 3, 120000),

('Менеджер по прокату', 'Управляет прокатом кинотеатра', 3, 80000),

('Менеджер по маркетингу', 'Управляет рекламой, продвижением кинотеатра. Работает с рекламодателями', 3, 90000),

('Киномеханик', 'Работает с кинолентами и аппаратурой', 2, 50000),

('Кассир', 'Работает на кассе или у зала', 1, 35000),

('Уборщик', 'Занимается уборкой', 1, 20000)

GO

--epmloyees

INSERT INTO Employees VALUES

('Администратор', 'Д. Кулаков', '1234456789', 20, '8005553535'),

('Менеджер по прокату', 'В. Пупкин', '8888777722', 10, '8005553535'),

('Менеджер по маркетингу', 'К. Пупкина', '1112223334', 12, '8005553535'),

('Киномеханик', 'А. Антонов', '9865321245', 50, '8005553535'),

('Кассир', 'А. Лопаткова', '6543216545', 5, '8005553535'),

('Кассир', 'М. Носов', '9876543211', 1, '8005553535'),

('Кассир', 'Е. Гуров', '9876555555', 3, '8005553535'),

('Уборщик', 'А. Мешкарева', '6665556665', 10, '8005553535'),

('Уборщик', 'А. Мешкарев', '6665556666', 7, '8005553535')

GO

--ticketTypes

INSERT INTO TicketTypes VALUES

('standart', 'билет без скидок', 0),

('child', 'детям до 6 включительно', 50),

('tuesday', 'акция на вечерние фильмы', 50)

--cashboxes

INSERT INTO Cashboxes VALUES

(0, 24),

(5, 12),

(5, 5)

GO

INSERT INTO EmployeeCashbox VALUES

(5, 0),

(5, 1),

(6, 2)

GO

--seanceTypes

INSERT INTO SeanceTypes VALUES

('common', 'Обычный сеанс'),

('school', 'Групповой паказ для школьников'),

('limited', 'Показ старых или ограниченных в прокате фильмов')

GO

--cinemaHolls

INSERT INTO CinemaHolls VALUES

(1, 10, 20),

(0, 10, 15),

(1, 10, 15),

(0, 10, 15)

GO

--employeeHoll

INSERT INTO EmployeeHoll VALUES

(1, 7, 5),

(2, 7, 5),

(3, 7, 5),

(4, 7, 5),

(1, 4, 12),

(2, 4, 12),

(3, 4, 12),

(4, 4, 12)

GO

--seances

INSERT INTO Seances VALUES

(1, 1, '15.05.2021 12:30', 16, 'common', 200),

(1, 2, '15.05.2021 13:30', 16, 'common', 200),

(1, 3, '15.05.2021 14:30', 16, 'common', 200),

(1, 4, '15.05.2021 15:30', 16, 'common', 200),

(5, 1, '16.05.2021 10:45', 6, 'common', 200),

(5, 2, '16.05.2021 12:45', 6, 'common', 200),

(5, 3, '16.05.2021 15:45', 6, 'common', 200),

(5, 4, '16.05.2021 16:45', 6, 'common', 200),

(6, 1, '15.05.2021 18:30', 18, 'limited', 200),

(6, 2, '15.05.2021 18:30', 18, 'limited', 200),

(6, 2, '15.05.2021 22:30', 18, 'limited', 200),

(6, 4, '15.05.2021 21:30', 18, 'limited', 200),

(7, 1, '17.05.2021 11:15', 6, 'school', 200),

(7, 2, '17.05.2021 11:15', 6, 'school', 200),

(7, 3, '17.05.2021 11:15', 6, 'school', 200),

(7, 4, '17.05.2021 11:15', 6, 'school', 200),

(10, 1, '17.05.2021 20:30', 18, 'common', 200),

(10, 2, '17.05.2021 21:45', 18, 'common', 200),

(10, 3, '17.05.2021 22:30', 18, 'common', 200),

(10, 4, '17.05.2021 18:15', 18, 'common', 200),

(2, 1, '16.05.2021 21:30', 16, 'limited', 200),

(9, 4, '16.05.2021 21:30', 16, 'common', 200)

GO

--advertisers

INSERT INTO Advertisers VALUES

('С. Есенин', 'РГУ', '8005553535'),

('А. Попов', 'РГРТУ', '8005553535') --resurrected

GO

--advertisings

INSERT INTO Advertising VALUES

(2, 'radio', 100, 10000),

(2, 'radio', 100, 10000),

(1, 'tourism', 120, 15000),

(1, 'tourism', 120, 15000)

GO

INSERT INTO AdvertisingEmployee VALUES

(3, 1),

(3, 2),

(3, 3),

(3, 4)

GO

ПРИЛОЖЕНИЕ В: исходный текст клиентской программы.

Код Controller:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web.Mvc;

using DAL.Department;

using Newtonsoft.Json;

namespace CinemaClientApplication.Controllers

{

public class HomeController : Controller

{

private readonly SQLManagerDAO requestDAO;

public HomeController()

{

requestDAO = new SQLManagerDAO();

}

public ActionResult Index()

{

return View(requestDAO.GetAllFilms().ToList());

}

public ActionResult FilmInfo(int id)

{

Models.SeancesFilmModel model = new Models.SeancesFilmModel

{

Seances = requestDAO.GetSeancesForFilms(id).ToList(),

Film = requestDAO.GetFilmById(id)

};

return View(model);

}

public ActionResult SeanceSeats(int id)

{

Models.SeanceHollModel model = new Models.SeanceHollModel

{

Tickets = requestDAO.GetTicketsForSeance(id).ToList(),

Holl = requestDAO.GetCinemaHollInfoBySeance(id),

Seance = requestDAO.GetSeanceInfo(id)

};

return View(model);

}

[HttpPost]

public ActionResult CreateTickets(string tickets)

{

var jsonRow = JsonConvert.DeserializeObject<string[]>(tickets);

List<Entities.Ticket> ticketsInfo = new List<Entities.Ticket>();

foreach (var str in jsonRow)

{

string[] row = str.Split('-');

ticketsInfo.Add(new Entities.Ticket(Entities.Ticket.LastId + 1, "standart", 0, Convert.ToInt32(row[0]), Convert.ToInt32(row[1]), Convert.ToInt32(row[2]), Convert.ToInt32(row[3])));

}

requestDAO.CreateTickets(ticketsInfo);

return RedirectToAction("Index");

}

}

}

Код класса «Билет» (остальные классы отношений подобны приведенному):

namespace Entities

{

public class Ticket

{

public static int LastId { private set; get; }

public int TicketID { private set; get; }

public string TicketType { private set; get; }

public int CashboxID { private set; get; }

public int SeanceId { private set; get; }

public int RowNumber { private set; get; }

public int SeatNumber { private set; get; }

public decimal Cost { private set; get; }

public Ticket(int id, string type, int cashId, int seanceId, int row, int seat, int cost)

{

TicketID = LastId = id;

TicketType = type;

CashboxID = cashId;

SeanceId = seanceId;

RowNumber = row;

SeatNumber = seat;

Cost = cost;

}

}

}

Код класса «SQLManager»:

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using Entities;

namespace DAL.Department

{

public class SQLManagerDAO

{

private readonly SqlConnection DBConnection;

public SQLManagerDAO()

{

DBConnection = DBAccess.GetDBConnection();

}

public IEnumerable<Film> GetAllFilms()

{

if (DBConnection.State != System.Data.ConnectionState.Open)

DBConnection.Open();

try

{

SqlCommand command = new SqlCommand("GetAllFilms", DBConnection)

{

CommandType = System.Data.CommandType.StoredProcedure

};

SqlDataReader dataReader = command.ExecuteReader();

while (dataReader.Read())

{

int id = dataReader.GetByte(0);

string name = dataReader.GetString(dataReader.GetOrdinal("FilmName"));

DateTime year = dataReader.GetDateTime(dataReader.GetOrdinal("ReleaseYear"));

string director = dataReader.GetString(dataReader.GetOrdinal("Director"));

int duration = dataReader.GetInt16(dataReader.GetOrdinal("Duration"));

string genre = dataReader.GetString(dataReader.GetOrdinal("Genre"));

float rating = (float)dataReader.GetDouble(dataReader.GetOrdinal("Rating"));

string image = dataReader.GetString(dataReader.GetOrdinal("FilmImage"));

yield return new Film(id, name, year, director, duration, genre, rating, image); ;

}

} finally

{

DBConnection.Close();

}

}

public IEnumerable<Ticket> GetTicketsForSeance(int seance)

{

if (DBConnection.State != System.Data.ConnectionState.Open)

DBConnection.Open();

try

{

SqlCommand command = new SqlCommand("GetTicketsForSeance", DBConnection)

{

CommandType = System.Data.CommandType.StoredProcedure

};

command.Parameters.AddWithValue("@seance", seance);

SqlDataReader dataReader = command.ExecuteReader();

while (dataReader.Read())

{

int id = dataReader.GetInt16(dataReader.GetOrdinal("TicketID"));

string type = dataReader.GetString(dataReader.GetOrdinal("TicketType"));

int cashId = dataReader.GetByte(dataReader.GetOrdinal("CashboxID"));

int seanceId = dataReader.GetByte(dataReader.GetOrdinal("SeanceId"));

int row = dataReader.GetByte(dataReader.GetOrdinal("RowNumber"));

int seat = dataReader.GetByte(dataReader.GetOrdinal("SeatNumber"));

int cost = dataReader.GetSqlMoney(dataReader.GetOrdinal("Cost")).ToInt32();

yield return new Ticket(id, type, cashId, seanceId, row, seat, cost);

}

}

finally

{

DBConnection.Close();

}

}

public Seance GetSeanceInfo(int seance)

{

if (DBConnection.State != System.Data.ConnectionState.Open)

DBConnection.Open();

try

{

SqlCommand command = new SqlCommand(@"GetSeanceInfo", DBConnection)

{

CommandType = System.Data.CommandType.StoredProcedure

};

command.Parameters.AddWithValue("@seance", seance);

SqlDataReader dataReader = command.ExecuteReader();

while (dataReader.Read())

{

int id = dataReader.GetByte(dataReader.GetOrdinal("SeanceId"));

int fimlId = dataReader.GetByte(dataReader.GetOrdinal("FilmID"));

int hollId = dataReader.GetByte(dataReader.GetOrdinal("HollID"));

DateTime time = dataReader.GetDateTime(dataReader.GetOrdinal("ShowTime"));

int rating = dataReader.GetByte(dataReader.GetOrdinal("AgeRating"));

string type = dataReader.GetString(dataReader.GetOrdinal("SeanceType"));

int cost = dataReader.GetSqlMoney(dataReader.GetOrdinal("TicketCost")).ToInt32();

return new Seance(id, fimlId, hollId, time, rating, type, cost);

}

}

finally {

DBConnection.Close();

}

return null;

}

public Film GetFilmById(int film)

{

if (DBConnection.State != System.Data.ConnectionState.Open)

DBConnection.Open();

try

{

SqlCommand command = new SqlCommand(@"GetFilmById", DBConnection);

command.CommandType = System.Data.CommandType.StoredProcedure;

command.Parameters.AddWithValue("@film", film);

SqlDataReader dataReader = command.ExecuteReader();

while (dataReader.Read())

{

int id = dataReader.GetByte(0);

string name = dataReader.GetString(dataReader.GetOrdinal("FilmName"));

DateTime year = dataReader.GetDateTime(dataReader.GetOrdinal("ReleaseYear"));

string director = dataReader.GetString(dataReader.GetOrdinal("Director"));

int duration = dataReader.GetInt16(dataReader.GetOrdinal("Duration"));

string genre = dataReader.GetString(dataReader.GetOrdinal("Genre"));

float rating = (float)dataReader.GetDouble(dataReader.GetOrdinal("Rating"));

string image = dataReader.GetString(dataReader.GetOrdinal("FilmImage"));

return new Film(id, name, year, director, duration, genre, rating, image);

}

}

finally

{

DBConnection.Close();

}

return null;

}

public IEnumerable<Seance> GetSeancesForFilms(int film)

{

if (DBConnection.State != System.Data.ConnectionState.Open)

DBConnection.Open();

try

{

SqlCommand command = new SqlCommand(@"SELECT\* FROM Seances WHERE FilmID = @FilmID", DBConnection);

command.Parameters.AddWithValue("@FilmID", film);

SqlDataReader dataReader = command.ExecuteReader();

while (dataReader.Read())

{

int id = dataReader.GetByte(dataReader.GetOrdinal("SeanceId"));

int fimlId = dataReader.GetByte(dataReader.GetOrdinal("FilmID"));

int hollId = dataReader.GetByte(dataReader.GetOrdinal("HollID"));

DateTime time = dataReader.GetDateTime(dataReader.GetOrdinal("ShowTime"));

int rating = dataReader.GetByte(dataReader.GetOrdinal("AgeRating"));

string type = dataReader.GetString(dataReader.GetOrdinal("SeanceType"));

int cost = dataReader.GetSqlMoney(dataReader.GetOrdinal("TicketCost")).ToInt32();

yield return new Seance(id, fimlId, hollId, time, rating, type, cost);

}

}

finally

{

DBConnection.Close();

}

}

public CinemaHoll GetCinemaHollInfoBySeance(int seance)

{

if (DBConnection.State != System.Data.ConnectionState.Open)

DBConnection.Open();

try

{

SqlCommand command = new SqlCommand(@"GetHollInfoBySeance", DBConnection)

{

CommandType = System.Data.CommandType.StoredProcedure

};

command.Parameters.AddWithValue("@seance", seance);

SqlDataReader dataReader = command.ExecuteReader();

while (dataReader.Read())

{

int id = dataReader.GetByte(dataReader.GetOrdinal("HollID"));

bool td = dataReader.GetBoolean(dataReader.GetOrdinal("TdEnable"));

int rows = dataReader.GetByte(dataReader.GetOrdinal("RowNumber"));

int seats = dataReader.GetByte(dataReader.GetOrdinal("SeatNumber"));

return new CinemaHoll(id, td, rows, seats);

}

}

finally

{

DBConnection.Close();

}

return null;

}

public void CreateTickets(List<Ticket> tickets)

{

if (DBConnection.State != System.Data.ConnectionState.Open)

DBConnection.Open();

try

{

foreach (var ticket in tickets)

{

SqlCommand command = new SqlCommand(@"CreateTicket", DBConnection)

{

CommandType = System.Data.CommandType.StoredProcedure

};

command.Parameters.AddWithValue("@SeanceId", ticket.SeanceId);

command.Parameters.AddWithValue("@TypeName", ticket.TicketType);

command.Parameters.AddWithValue("@CashboxID", ticket.CashboxID);

command.Parameters.AddWithValue("@RowNumber", ticket.RowNumber);

command.Parameters.AddWithValue("@SeatNumber", ticket.SeatNumber);

command.Parameters.AddWithValue("@Cost", ticket.Cost);

if (command.ExecuteNonQuery() < 1)

{

throw new Exception("Error with ticket");

}

}

}

finally

{

DBConnection.Close();

Код стартовой страницы:

@{

ViewBag.Title = "КиноТеатр";

}

<div class="films-area">

@foreach (var item in Model)

{

<div class="film-item">

<div class="film-logo">

<img src="~/Content/img/films.img/@item.FilmImage" alt="Alternate Text" />

</div>

<div class="film-label">

<span>@item.FilmName</span>

<div class="film-short-info">

<div class="film-director">

<span>Режиссер:</span> <span>@item.Director</span>

</div>

<div class="film-director">

<span>Жанр: </span> <span>@item.Genre</span>

</div>

</div>

</div>

<div class="film-button">

@Html.ActionLink("Перейти к сеансам", "FilmInfo", "Home", new { id = @item.FilmID }, new { @class = "button" })

</div>

</div>

}

</div>

Код страницы с информацией о фильме:

@{

ViewBag.Title = "FilmInfo";

}

<div class="film-area">

<div class="image">

<img src="~/Content/img/films.img/@Model.Film.FilmImage" alt="Alternate Text" />

</div>

<div class="info">

<div class="label">

<span>@Model.Film.FilmName</span>

</div>

<div class="info-table">

<div><span>Год выхода: &nbsp;</span><span> @Model.Film.ReleaseYear.ToString("dd/MM/yyyy")</span></div>

<div><span>Режиссер: &nbsp;</span><span> @Model.Film.Director</span></div>

<div><span>Длительность фильма: &nbsp;</span><span> @Model.Film.Duration</span></div>

<div><span>Жанр: &nbsp;</span><span> @Model.Film.Genre</span></div>

<div><span>Рейтинг (IMDb): &nbsp;</span><span> @Model.Film.Rating</span></div>

</div>

</div>

</div>

<div class="seances-area">

@if (Model.Seances.Count == 0)

{

<span class="no-seances">На данные фильм сеансов нет!</span>

} else

{

<div class="seance-general-label">

<span>Расписание сеансов</span>

</div>

}

@foreach (var seance in Model.Seances)

{

<div class="seance-item">

<div class="seance-time">

<span>@seance.ShowTime</span>

</div>

<div class="seance-button">

@Html.ActionLink("Перейти к сеансу", "SeanceSeats", "Home", new { id = @seance.SeanceId }, new { @class = "button" })

</div>

</div>

}

</div>

Код страницы выбранного сеанса:

@using Newtonsoft.Json;

@{

ViewBag.Title = "SeanceSeats";

}

<script>

let lockedSeats = [];

let selectedTickets = [];

let ticketCost = @Model.Seance.TicketCost;

let seanceId = @Model.Seance.SeanceId;

let seatPlace;

let ticketsOrdered;

function LockSeats(row, seat) {

let str = row + '-' + seat;

seat = document.querySelector(`img[alt = "${str}"]`);

$(seat).css({ 'opacity': 1, 'cursor': 'default' });

lockedSeats.push(str);

}

</script>

<div class="seance-info">

<div><span>Время показа: &nbsp;</span><span>@Model.Seance.ShowTime</span></div>

<div><span>Номер зала: &nbsp;</span><span>@Model.Holl.HollID</span></div>

@if (Model.Holl.TdEnable)

{<div><span style="text-decoration: underline">Сеанс в 3D</span></div>}

<div><span style="text-decoration: underline">@Model.Seance.AgeRating +</span></div>

<div><span>Стоимость билета: &nbsp;</span><span>@Model.Seance.TicketCost рублей</span></div>

</div>

<div class="seats-handler">

<div><span>Количество выбранных мест: &nbsp;</span> <span class="seats-count"></span></div>

<div><span>Общая стоимость: &nbsp;</span> <span class="ticket-cost"></span></div>

<div><button disabled>Заказать билеты</button></div>

</div>

<div class="seats-area">

@for (int i = 0; i < Model.Holl.RowNumber; i++)

{

<div class="seat-row">

@for (int j = 0; j < Model.Holl.SeatNumber; j++)

{

<div class="seat">

<img src="~/Content/img/seat.svg" alt="@(i+1)-@(j+1)" />

</div>

}

</div>

}

</div>

<div class="script-area">

@foreach (var ticket in Model.Tickets)

{

int row = ticket.RowNumber;

int seat = ticket.SeatNumber;

<script>

LockSeats(@(row), @(seat));

</script>

}

</div>

Код файла «Script.js»:

$(document).ready(function () {

$('.seance-button').hover(function () {

$(this).siblings('.seance-time').css('text-decoration', 'underline');

}, function () {

$(this).siblings('.seance-time').css('text-decoration', 'none');

});

$('.seat').on("click", function () {

let seatPlace = $(this).children('img').attr('alt');

let row = seatPlace.split('-')[0];

let seat = seatPlace.split('-')[1];

if (lockedSeats.indexOf(row + '-' + seat) != -1) {

return;

}

if (selectedTickets.indexOf(row + '-' + seat) != -1) {

selectedTickets.splice(selectedTickets.indexOf(row + '-' + seat), 1);

$(this).children('img').css({ 'outline': 'none', 'opacity': .4 });

} else {

$(this).children('img').css({ 'outline': '3px solid #5c6b7f', 'opacity': 1 });

selectedTickets.push(row + '-' + seat);

}

drawTicketInfo();

});

function drawTicketInfo() {

if (selectedTickets.length > 0) {

$('.seats-count').text(selectedTickets.length);

$('.ticket-cost').text(selectedTickets.length \* ticketCost + ' рублей');

$('.seats-handler > div > button').prop('disabled', false);

$('.seats-handler > div > button').css({ 'opacity': 1, 'cursor': 'pointer' });

}

else {

$('.seats-count').text("");

$('.ticket-cost').text("");

$('.seats-handler > div > button').prop('disabled', true);

$('.seats-handler > div > button').css({ 'opacity': .5, 'cursor': 'default' });

}

}

$(".seats-handler button").on("click", ticketOrdered = function () {

let dataRow = [];

for (var i = 0; i < selectedTickets.length; i++) {

dataRow.push(seanceId + '-' + selectedTickets[i] + '-' + ticketCost);

}

let data = JSON.stringify(dataRow);

$.ajax({

url: "../CreateTickets",

type: "POST",

data: { tickets: data },

success: function () {

ShowOrdered();

//window.location = "../Index";

},

error: function () {

console.log(data);

}

});

});

function ShowOrdered() {

$('.wrapper').css('z-index', -1);

$('.ordered-active').css('visibility', 'visible');

}

});