МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 01 01 «Программное обеспечение информационных

технологий»

Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка базы данных сети кинотеатров»

Выполнил студент Сураго Дмитрий Александрович

(Ф.И.О.)

Руководитель проекта пр.-стаж. Нистюк О.А.

(учен. степень, звание, должность, Ф.И.О., подпись)

Заведующий кафедрой к.т.н., доц. Смелов В.В.

(учен. степень, звание, должность, Ф.И.О., подпись)

Консультант: пр.-стаж. Нистюк О.А.

(учен. степень, звание, должность, Ф.И.О., подпись)

Нормоконтролер: пр.-стаж. Нистюк О.А.

(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовой проект защищен с оценкой

Минск 2020

Содержание

[Введение 3](#_Toc58244046)

[1 Постановка задачи 4](#_Toc58244047)

[1.1 Аналитический обзор аналогов 4](#_Toc58244048)

[1.2 Описание используемых технологий 6](#_Toc58244049)

[2 Разработка модели базы данных 7](#_Toc58244050)

[3 Разработка необходимых объектов 8](#_Toc58244051)

[3.1 Таблицы 8](#_Toc58244052)

[3.2 Процедуры 12](#_Toc58244053)

[3.3 Индексы 13](#_Toc58244054)

[4 Описание процедур импорта и экспорта 15](#_Toc58244055)

[4.1 Процедура импорта данных из XML-файла 15](#_Toc58244056)

[4.2 Процедура экспорта данных в формат XML 15](#_Toc58244057)

[5 Описание технологии 17](#_Toc58244058)

[5.1 Создание пользователей 17](#_Toc58244059)

[5.2 Прозрачное шифрование данных 18](#_Toc58244060)

[5.3 Оценка уязвимостей через SSMS 19](#_Toc58244061)

[5.4 Аудит 20](#_Toc58244062)

[6 Тестирование 22](#_Toc58244063)

[6.1 Тестирование работоспособности 22](#_Toc58244064)

[6.2 Тестирование производительности 23](#_Toc58244065)

[Заключение 25](#_Toc58244066)

[Список используемых источников 26](#_Toc58244067)

[Приложение А 27](#_Toc58244068)

[Приложение Б 28](#_Toc58244069)

# Введение

Данные всегда играли важную роль в жизни людей и в разные времена их хранили по-разному. В наше время всё больше информации хранится в цифровом виде и для хранения такой информации отлично подходят различные базы данных. Использование базы данных упрощает управление и хранение данных. Одним из примеров успешного внедрения баз данных является использование их различными кинотеатрами.

Сейчас есть множество удобных приложений, позволяющих без особых усилий совершать заказы билетов на разные сеансы в кинотеатрах, с помощью таких приложений обычно можно совершить полный цикл заказа билета, начиная от выбора кинотеатра и сеанса по необходимым параметрам и заканчивая оформлением заказа.

Целью моего курсового проекта является создание базы данных сети кинотеатров.

Основными задачами курсовой работы являются:

* провести аналитический обзор литературы;
* спроектировать базу данных;
* реализовать функциональность базы данных;
* провести тестирование используемой технологии в базе данных;
* разработать приложение для работы с базой данных;
* написать руководство пользователя.

База данных — это организованная структура, предназначенная для хранения информации, систематизированная таким образом, чтобы эти материалы могли быть найдены и обработаны с помощью электронной вычислительной машины.

Реляционная база данных — база данных, основанная на реляционной модели данных.

В качестве СУБД для базы данных была выбрана «Microsoft SQL Server», в связи с ее простотой, производительностью и надежностью.

1. **Постановка задачи**

В соответствии с заданием курсового проекта следует создать базу данных для сети кинотеатров. Для того чтобы сформировать окончательные требования к проектируемому программному средству сначала рассмотрим аналоги из той же области, а так же технологии, которые используются в данном проекте.

* 1. **Аналитический обзор аналогов**

Раньше, для администрирования кинотеатров, вели бухгалтерский учет, что осложняло весь процесс. Сейчас же используются базы данных с разработанным для них программным обеспечением, позволяя уменьшить объем бумажных работ и оптимизировать работу всей системы.

Постепенно с развитием программного обеспечения ЭВМ появились идеи создания управляющих систем, которые позволяли бы накапливать, хранить и обновлять взаимосвязанные данные по целому комплексу решаемых задач. Эти идеи нашли свое воплощение в системах управления базами данных (СУБД). СУБД взаимодействуют не с локальными, а взаимосвязанными по информации массивами, называемыми базами данных. С появлением персональных компьютеров СУБД становятся наиболее популярным средством обработки табличной информации. Они являются инструментальным средством проектирования банков данных при обработке больших объемов информации.

Можно рассмотреть структуру организации работы кинотеатров среди приложений, ориентированных на бронирование билетов. Рассмотрим интернет-приложение для сети кинотеатров «Silver Screen».

Рассмотрим в качестве примера список кинотеатров данного сайта на рисунке 1.1.

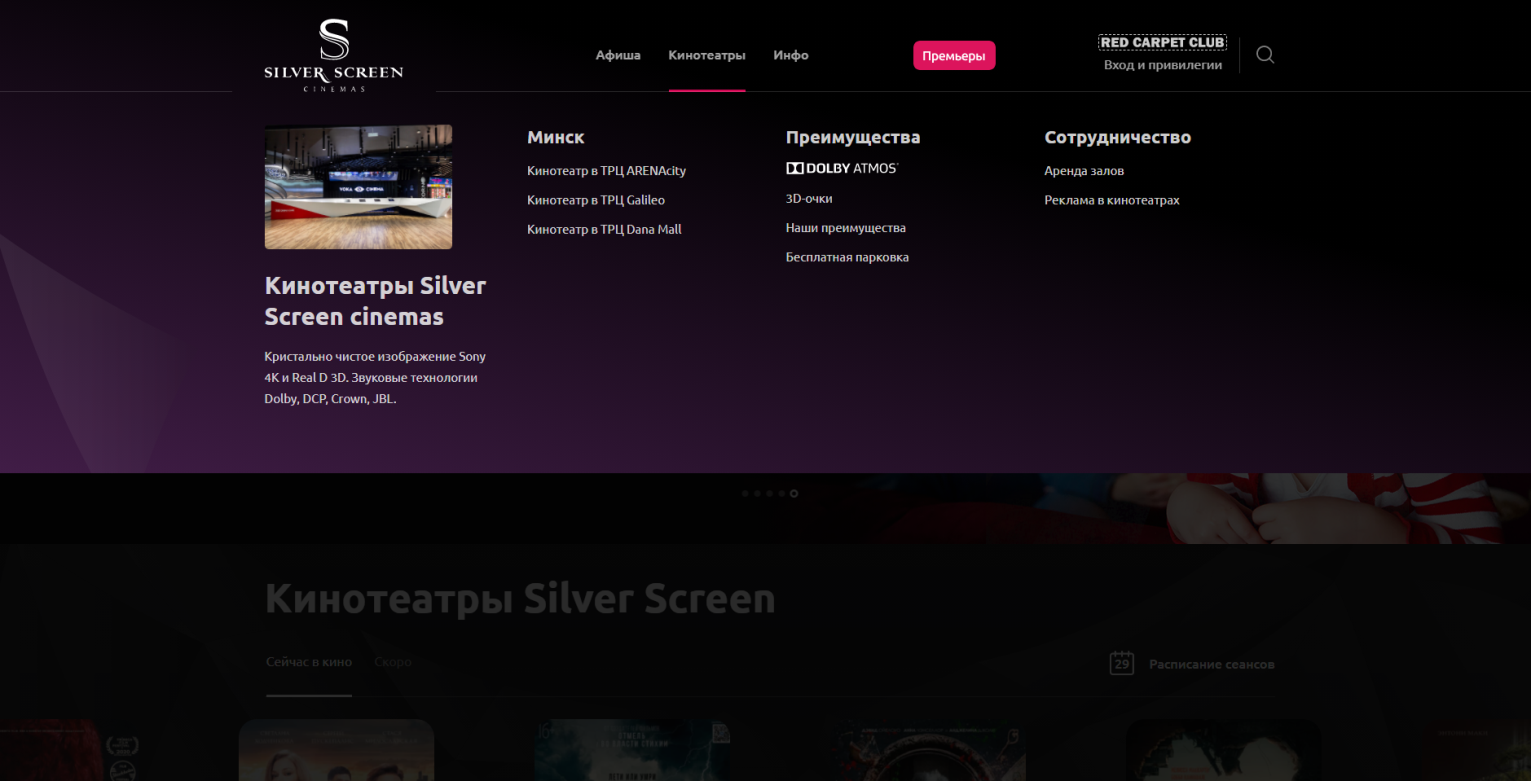


Рисунок 1.1 – Пример списка кинотеатров «Silver Screen»

Тут видно, что в данной сети кинотеатров присутствует 3 кинотеатра. Кликнув на один из них, мы попадаем на страницу описания кинотеатра, где можно ознакомиться с подробной информацией о конкретном кинотеатре, а так же заказать билет на любой сеанс, что продемонстрировано на рисунке 1.2 и 1.3.

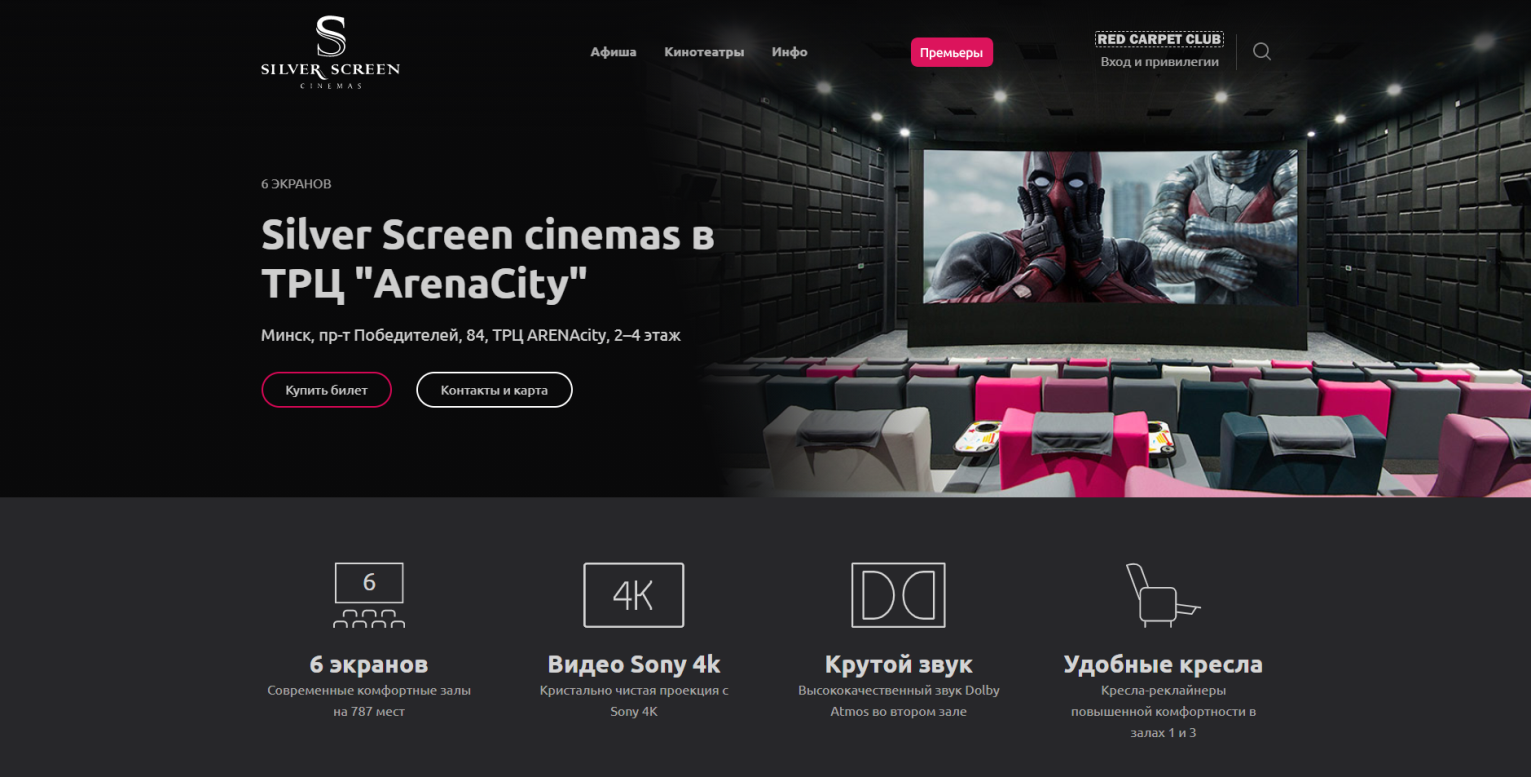


Рисунок 1.2 – Подробное описание выбранного кинотеатра

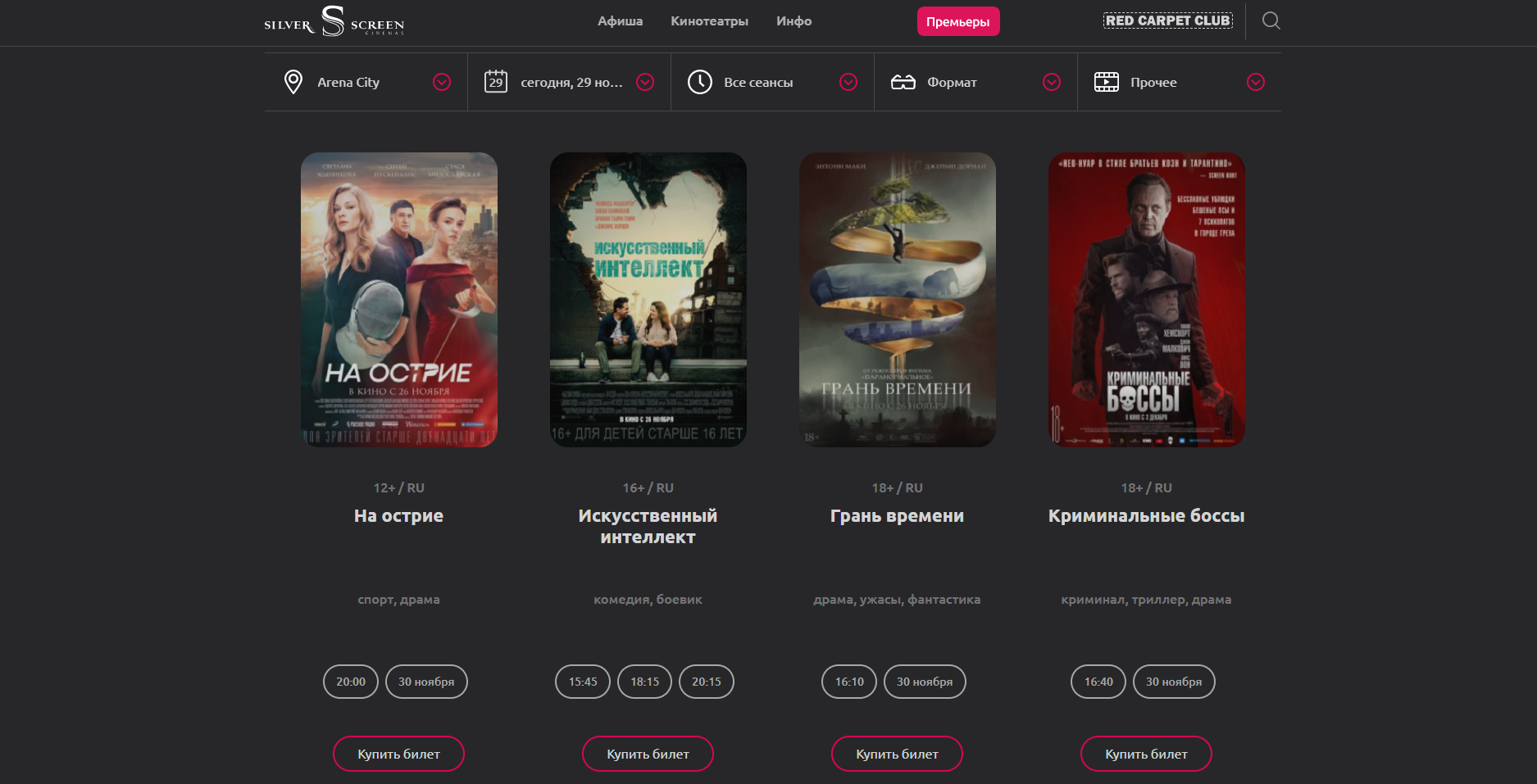


Рисунок 1.3 – Заказ билета на любой фильм

* 1. **Описание используемых технологий**

СУБД существует огромное множество: Oracle, MS SQL Server, NoSQL, MySql и так далее. Для организации работы с базой данных в данной работе было решено использовать одну из наиболее популярных СУБД «Microsoft SQL Server», так как она предоставляет необходимые возможности оперирования объектами базы данных.

Также в процессе выбора технологии была рассмотрена технология «Настройка системы безопасности сервера СУБД», которая позволяет обезопасить хранимые данные от мошенников. Данная технология включает в себя создание пользователей с определенными ролями и привилегиями, а так же прозрачное шифрование данных и Audit контроль.

1. **Разработка модели базы данных**

Для реализации базы данных для сети кинотеатров было разработано 14 таблиц. Диаграмма структуры полученной базы данных, разработанной в СУБД «Microsoft SQL Server», представлена в приложении А.

Таблица AgeResctictions хранит возрастные ограничения для фильмов.

Таблица Cinema содержит информацию о кинотеатрах.

Таблица Format хранит форматы фильмов.

Таблица FilmCompany хранит для каждого фильма киностудии, которые их снимали.

Таблица FilmGenre хранит жанры для каждого фильма.

Таблица FilmProducer содержит информацию о режиссерах для каждого фильма.

Таблица PlacesSeances связывает таблицы Places и Seances для продажи билетов на конкретный сеанс и место.

Таблица Films содержит информацию обо всех фильмах.

Таблица RentalFilms связывает таблицы Cinema и Films для получения фильмов, которые находятся в прокате, для конкретного кинотеатра.

Таблица TypePlaces хранит тип места в зале.

Таблица Hall хранит информацию о каждом зале всех кинотеатров.

Таблица Seances содержит информацию о сеансах.

Таблица Places содержит все места во всех залах кинотеатров.

Таблица Tickets содержит информацию о билетах на сеансы.

Создание перечисленных таблиц с установлением внешних связей описано в главе 3.

1. **Разработка необходимых объектов**

При разработке курсового проекта понадобились следующие объекты:

* Таблицы;
* Хранимые процедуры;
* Индексы.
  1. **Таблицы**

Таблицы являются неотъемлемой частью любой реляционной базы данных. Краткая характеристика каждой из таблиц была предоставлена в разделе 2, а код их создания можно увидеть в Приложении Б. Ниже мы рассмотрим каждую таблицу подробнее.

Таблица AgeResctictions состоит из одного столбца, рисунок 3.1:

– age\_restrictions – возрастные ограничения для фильмов.



Рисунок 3.1 – Описание таблицы AgeResctictions

Таблица Cinema состоит из шести столбцов, рисунок 3.2:

– id\_cinema – уникальный идентификатор кинотеатра;

– name\_cinema – название кинотеатра;

– region – регион, в котором расположен кинотеатр;

– address – адрес кинотеатра;

– phone – телефон кинотеатра;

– website – сайт кинотеатра.

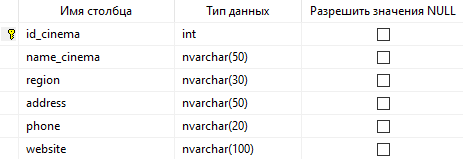


Рисунок 3.2 – Описание таблицы Cinema

Таблица Format состоит из одного столбца, рисунок 3.3:

– format – формат.



Рисунок 3.3 – Описание таблицы Format

Таблица FilmCompany состоит из трех столбцов, рисунок 3.4:

– id\_company – уникальный идентификатор киностудии;

– id\_film – идентификатор фильма;

– name\_company – название киностудии.

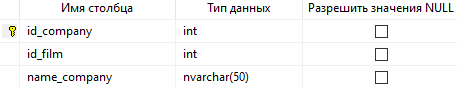


Рисунок 3.4 – Описание таблицы FilmCompany

Таблица FilmGenre состоит из трех столбцов, рисунок 3.5:

– id\_genre – уникальный идентификатор жанра;

– id\_film – идентификатор фильма;

– name\_genre – название жанра.

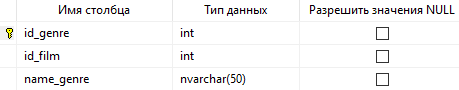


Рисунок 3.5 – Описание таблицы FilmGenre

Таблица FilmProducer состоит из трех столбцов, рисунок 3.6:

– id\_producer – уникальный идентификатор режиссера;

– id\_film – идентификатор фильма;

– name\_producer – имя режиссера.

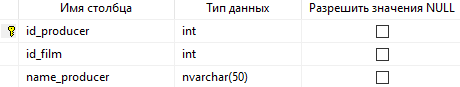


Рисунок 3.6 – Описание таблицы FilmProducer

Таблица PlacesSeances состоит из трех столбцов, рисунок 3.7:

– id\_ps – уникальный идентификатор места на сеанс;

– id\_place – идентификатор места в зале;

– id\_seance – идентификатор сеанса.

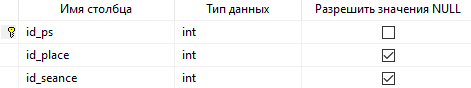


Рисунок 3.7 – Описание таблицы PlacesSeances

Таблица Films состоит из семи столбцов, рисунок 3.9:

– id\_film – уникальный идентификатор фильма;

– age\_restrictions – возрастное ограничение;

– name\_film – название фильма;

– year – год выхода фильма;

– description – описание фильма;

– duration – длительность фильма;

– format – формат фильма.

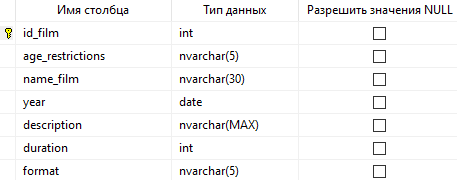


Рисунок 3.9 – Описание таблицы Films

Таблица RentalFilms состоит из пяти столбцов, рисунок 3.8:

– id\_rental – уникальный идентификатор для фильма в прокате;

– id\_film – идентификатор фильма;

– id\_cinema – идентификатор кинотеатра;

– renting\_in – дата входа фильма в прокат;

– renting\_out – дата выхода фильма из проката.

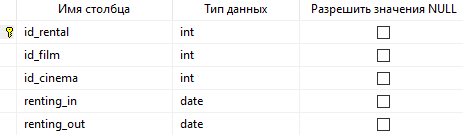


Рисунок 3.8 – Описание таблицы RentalFilms

Таблица TypePlaces состоит из одного столбца, рисунок 3.10:

– type – тип места в зале.



Рисунок 3.10 – Описание таблицы TypePlaces

Таблица Hall состоит из четырех столбцов, рисунок 3.11:

– id\_hall – уникальный идентификатор зала;

– id\_cinema – идентификатор кинотеатра;

– name\_hall – название зала;

– capacity – вместимость зала.

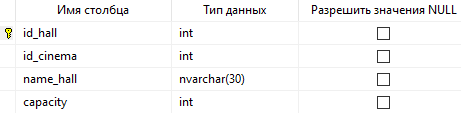


Рисунок 3.11 – Описание таблицы Hall

Таблица Seances состоит из четырех столбцов, рисунок 3.12:

– id\_seance – уникальных идентификатор сеанса;

– id\_film – идентификатор фильма;

– date – дата сеанса;

– time – время сеанса.

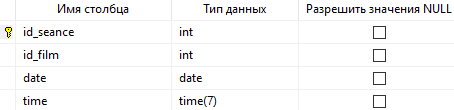


Рисунок 3.12 – Описание таблицы Seances

Таблица Places состоит из пяти столбцов, рисунок 3.13:

– id\_place – уникальный идентификатор места;

– id\_hall – идентификатор зала;

– type – тип места;

– number – номер места в зале;

– row – ряд зала, в котором находится место.

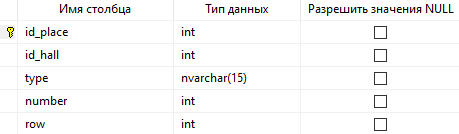


Рисунок 3.13 – Описание таблицы Places

Таблица Tickets состоит из трех столбцов, рисунок 3.14:

– id\_ticket – уникальный идентификатор билета;

– price – стоимость билета;

– status – статус места, на которое продается билет.

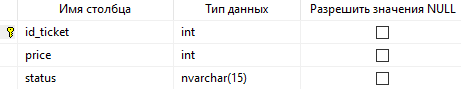


Рисунок 3.14 – Описание таблицы Tickets

* 1. **Процедуры**

Использование хранимых процедур позволяет ограничить либо вообще исключить непосредственный доступ пользователей к таблицам базы данных, оставив пользователям только разрешения на выполнение хранимых процедур, обеспечивающих косвенный и строго регламентированный доступ к данным.

Всего было разработано 39 процедур для работы с данными. Краткое описание процедур описано далее.

Пример создания хранимой процедуры GetRentalFilmsInCinema, которая выводит список всех фильмов, находящиеся в прокате, по заданному кинотеатру, представлен на рисунке 3.15.

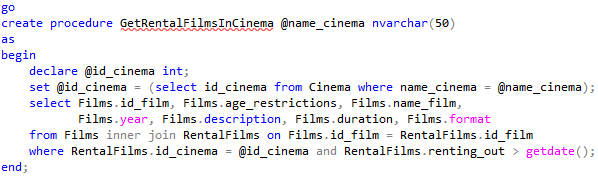


Рисунок 3.15 – Описание процедуры GetRentalFilmsInCinema

Данная процедура принимает один параметр – это имя кинотеатра (name\_cinema). Процедура, путем соединения двух таблиц Films и RentalFilms, находит все фильмы, которые в прокате на данный момент, и отбирает из них те фильмы, которые идут в кинотеатре с именем name\_cinema.

Далее представлен список всех процедур и их описание:

– CreateCinema – добавление нового кинотеатра;

– CreateFilmCompany – добавление киностудии для фильма;

– CreateFilmGenre – добавление жанра для фильма;

– CreateFilmProducer – добавление режиссера для фильма;

– CreateFilms – добавление фильма;

– CreateHall – добавление зала для кинотеатра;

– CreatePlaces – добавление мест для залов;

– CreatePlacesSeances – установление мест на сеансы;

– CreateRentalFilms – добавление фильмов в прокат для кинотеатра;

– CreateSeances – добавление сеансов;

– CreateTickets – установление цен на билеты;

– DeleteEndRentalFilms – удаление фильмов, вышедших с проката;

– DeleteEndSeances – удаление законченных сеансов;

– ExportXMLCinema – экспорт данных из таблицы Cinema в XML файл;

– GetAllCinema – получить список всех кинотеатров;

– GetCinemaByAddress – получить кинотеатр по адресу;

– GetCinemaByName – получить кинотеатр по названию;

– GetCinemaByRegion – получить кинотеатры в заданном регионе;

– GetFilmsByName – получение фильма по названию;

– GetFreeSeatsOnSeances – получение свободных мест на сеанс;

– GetRentalFilmsInCinema – получение фильмов в прокате в кинотеатре;

– GetSeancesByDate – получение сеансов по указанной дате;

– GetSeancesByFilmName – получение сеансов по названию фильма;

– GetSeancesInCinema – получение всех сеансов в кинотеатре;

– ImportXMLCinema – импорт данных из XML файла в таблицу Cinemaж

– InsertCinema – генерация 100.000 записей для таблицы Cinema;

– InsertRentalFilms – генерация 100.000 записей для таблицы RentalFilms;

– ReturnTickets – возврат билета;

– SaleTickets – продажа билета;

– UpdateCinema – обновление данных о кинотеатре;

– UpdateFilmCompany – обновление данных о киностудии;

– UpdateFilmGenre – обновление данных о жанре;

– UpdateFilmProducer – обновление данных по режиссере;

– UpdateFilms – обновление данных о фильме;

– UpdateHall – обновление данных о зале кинотеатра;

– UpdatePlaces – обновление данных места в зале;

– UpdatePlacesSeances – обновление списка мест на сеанс;

– UpdateRentalFilms – обновление списка фильмов в прокате;

– UpdateSeances – обновление данных о сеансе;

– UpdateTickets – обновление данных о билете на сеанс.

* 1. **Индексы**

Индекс – объект базы данных, создаваемый с целью повышения производительности поиска данных. Таблицы в базе данных могут иметь большое количество строк, которые хранятся в произвольном порядке, и их поиск по заданному критерию путём последовательного просмотра таблицы строка за строкой может занимать много времени. В связи с необходимостью выборки таблиц, в которых может храниться большой объем данных, было принято решение разработать 22 индекса, рисунок 3.16.

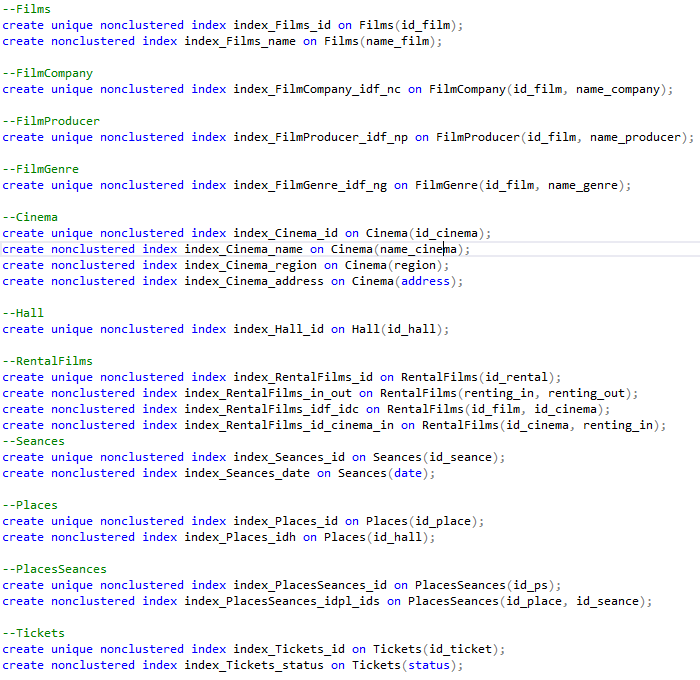


Рисунок 3.16 – Индексы

1. **Описание процедур импорта и экспорта**

База данных обычно имеет не самостоятельную ценность, является частью информационной системы. Независимо от того, как устроена эта система, на противоположном от БД конце находится интерфейс взаимодействия с пользователем, и задача программиста предоставить простой и понятный способ работы с хранящимися в БД данными и объектами.

При всей своей отлаженности и очевидности, классический способ хранения и представления объектов развитой структуры имеет и вполне определенные недостатки и может вызывать проблемы, с которыми сталкивался любой разработчик, пытавшийся реализовать таким способом достаточно сложную систему. В некоторых ситуациях, решить эти проблемы позволяет хранение объектов в виде XML.

* 1. **Процедура импорта данных из XML-файла**

Для импорта данных в таблицу Cinema, из файла формата xml, была разработана процедура ImportXMLCinema. Для получения xml файла и последующего разбора со вставкой используется конструкция FROM OPENROWSET совместно с параметром BULK. Процедура ImportXMLCinema представлена ниже на рисунке 4.1.

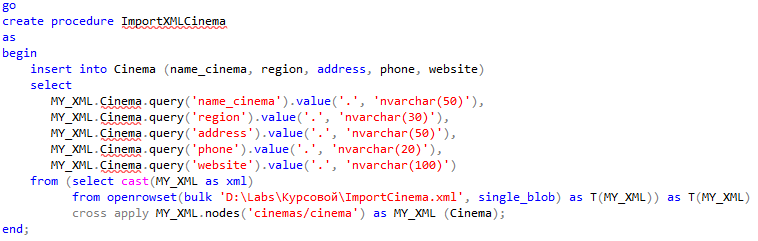


Рисунок 4.1 – Описание процедуры ImportXMLCinema

* 1. **Процедура экспорта данных в формат XML**

Для экспорта таблицы PRODUCTS в формате xml была разработана процедура ExportXMLCinema. Для формирования xml в select запросе используется конструкция FOR XML. Для вывода в файл была использована расширенная хранимая процедура xp\_cmdshell. Процедура ExportXMLCinema представлена ниже на рисунке 4.1.

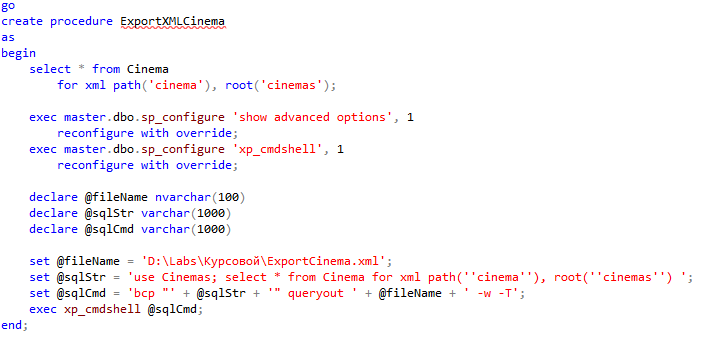


Рисунок 4.2 – Описание процедуры ExportXMLCinema

1. **Описание технологии**

Тут рассматриваются средства SQL Server для обеспечения безопасности, которые были применены к проекту, связанные с настройкой и обеспечением безопасности в СУБД.

* 1. **Создание пользователей**

Пользователь базы данных – это физическое или юридическое лицо, которое имеет доступ к базе данных и пользуется услугами информационной системы для получения информации. Пользователи базы данных получают права доступа для чтения, вставки, обновления и удаления конкретных объектов.

Для того, чтобы войти на сервер СУБД, требуется логин сервера и пароль. Так же выдается соответствующая роль сервера. Далее, для взаимодействия с самой БД, необходимо создать пользователя с определенной ролью базы данных.

На данном сервере создано два пользователя – администратор и клиент.

Администратору разрешено выполнять все действия с базой данных, включая ее изменение и удаление. На рисунке 5.1 показано создание логина ADMINISTATOR\_LOG с паролем “admin”, наделение ролью сервера sysadmin, что означает любые выполняемые действия на сервере, и создание пользователя для базы данных ADMINISTRATOR с ролью базы данных db\_owner – владелец этой базы данных.

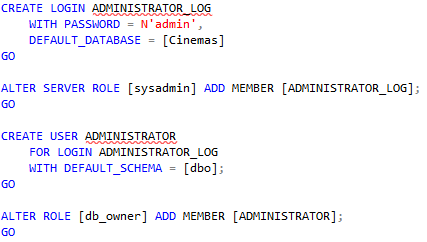


Рисунок 5.1 – Создание логина сервера и администратора БД

Клиенту разрешено выполнять только чтение и запись данных в БД. На рисунке 5.2 продемонстрировано создание логина CLIENT\_LOG с паролем “12345” и создание пользователя для базы данных CLIENT с ролью базы данных db\_datareader и db\_datawriter – чтение и запись данных.

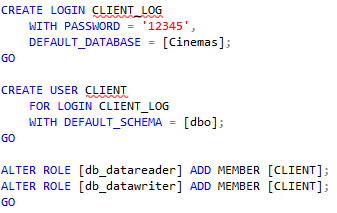


Рисунок 5.2 – Создание логина сервера и клиента БД

* 1. **Прозрачное шифрование данных**

При прозрачном шифровании можно защитить базы данных без изменения существующих приложений, структур баз данных или процессов. Это лучший вариант для выполнения требований нормативных актов и правил корпоративной безопасности, поскольку шифруется вся база данных на жестком диске.

Прозрачное шифрование данных шифрует базы данных в реальном времени, по мере внесения записей в файлы (\*.mdf) базы данных SQL Server и файлы (\*.ldf) журнала транзакций. Записи также шифруются в реальном времени во время резервного копирования базы данных, а затем формируются моментальные снимки. Данные шифруются перед записью на диск и расшифровываются перед извлечением. Процесс полностью прозрачен для пользователя или приложения, поскольку выполняется на уровне SQL Server Service.

При прозрачном методе SQL Server шифрует базу данных с помощью ключа шифрования базы данных. Этот асимметричный ключ хранится в загрузочной записи базы данных и потому всегда доступен при восстановлении.

Чтобы разрешить прозрачное шифрование данных, необходимо создать DMK и сертификат сервера в базе данных master, рисунок 5.3.

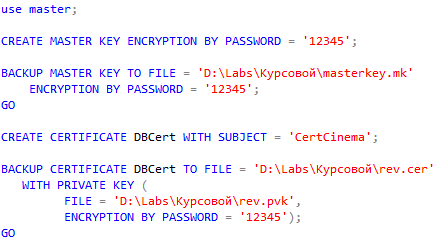


Рисунок 5.3 – Создание DMK и сертификата сервера

Важно немедленно сделать резервную копию сертификата и ключа DMK, связанного с сертификатом, что демонстрируется на том же рисунке 5.3 с применением оператора BACKUP. Если сертификат становится недоступным то потребуются резервные копии как сертификата, так и DMK.

Последний шаг – создать ключ шифрования базы данных и включить шифрование для базы данных, которую нужно защитить. На рисунке 5.4 включается прозрачное шифрование в базе данных Cinemas.

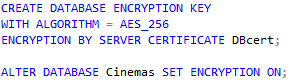


Рисунок 5.4 – Включение прозрачного шифрования

* 1. **Оценка уязвимостей через SSMS**

В SQL Server Management Studio есть функция оценки уязвимостей для базы данных, рисунок 5.5.

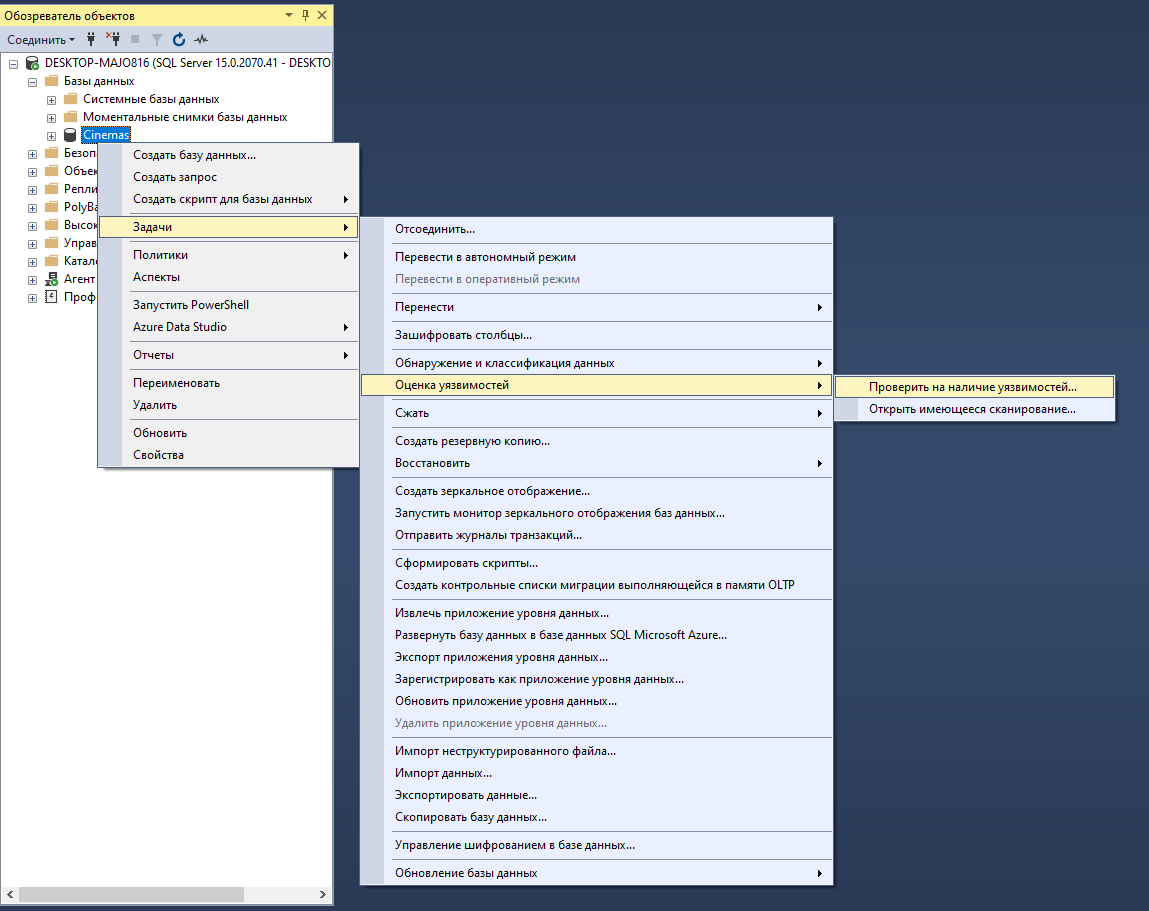


Рисунок 5.5 – Открытие оценки уязвимостей в SSMS

Сканнер оценит базу данных на предмет популярных ошибок в конфигурации безопасности и даст соответствующие рекомендации, рисунок 5.6.

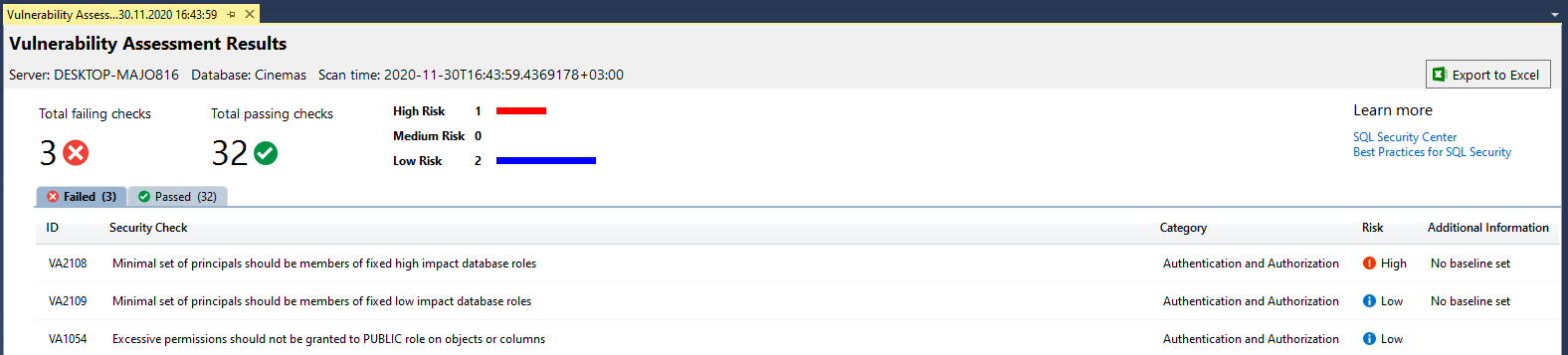


Рисунок 5.6 – Ошибки в конфигурации безопасности

Обязательно стоит проходить этим сканнером по базам данных. Он может выявить скрытые проблемы, которых не видно на первый взгляд.

* 1. **Аудит**

SQL Server предоставляет возможность вести аудит любой пользовательской активности в экземпляре сервера. Это очень мощный инструмент, который позволяет полностью контролировать действия пользователей.

На рисунке 5.7 показана базовая настройка аудита.

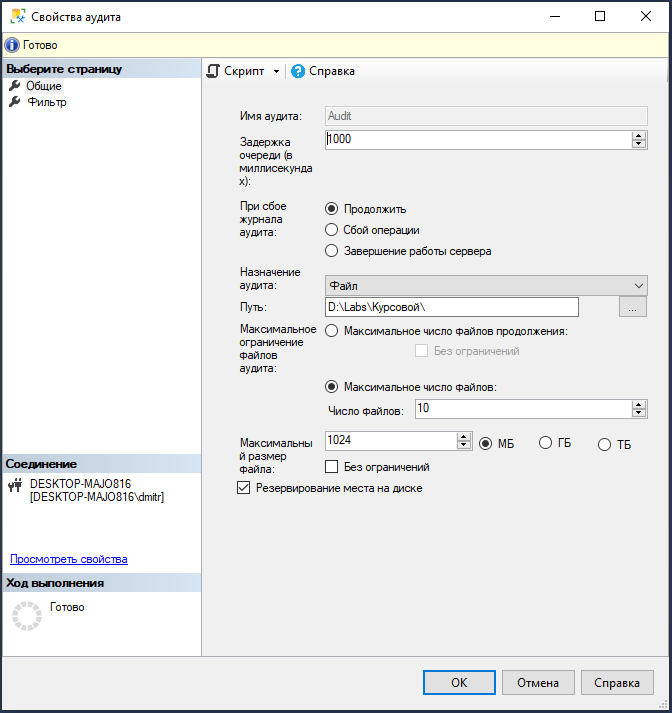


Рисунок 5.7 – Настройка аудита

После того создания аудита, в журнале аудита можно будет посмотреть события, которые зафиксированы процедурой аудита, на рисунке 5.8.



Рисунок 5.8 – Журнал аудита

1. **Тестирование**

Тестирование приложения предназначено для выявления возможных ошибок при разработке и для того, чтобы исключить действия пользователя, которые могут повлечь за собой аварийное завершение программы.

* 1. **Тестирование работоспособности**

Рассмотрим несколько процедур для работы с базой данных Cinemas. Для начала выведем фильмы, которые вышли в прокат в заданном кинотеатре. Для этого используем процедуру GetRentalFilmsInCinema. Вызов процедуры продемонстрирован на рисунке 6.1, а результат ее работы на рисунке 6.2.



Рисунок 6.1 – Вызов процедуры GetRentalFilmsInCinema



Рисунок 6.2 – Результат работы процедуры GetRentalFilmsInCinema

Далее добавим фильм, который выйдет в прокат в кинотеатре. Для этого воспользуемся процедурой CreateRentalFilms. Для начала введем правильные данные, чтобы добавление прошло успешно, рисунок 6.3. Чтобы посмотреть, успешно ли выполнилась процедура, выполним Select запрос к таблице RentalFilms, который продемонстрирован на рисунке 6.4, и посмотрим, добавилась ли новая запись.



Рисунок 6.3 – Добавление фильма, вышедшего в прокат в кинотеатре

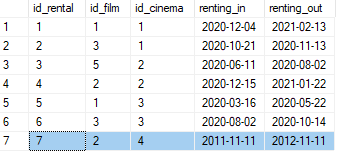


Рисунок 6.4 – Результат выборки из таблицу RentalFilms

Как видно на рисунке 6.4, выделенная запись и есть та запись, которую мы добавили. Но для демонстрации всех возможностей процедуры CreateRentalFilms введем заранее неправильные данных, например, введем дату выхода с проката меньше, чем дату поступления фильма в прокат. Результат продемонстрирован на рисунке 6.5.



Рисунок 6.5 – Результат работы процедуры CreateRentalFilms с неправильными данными

Из-за того, что мы ввели неверные данные, мы получили такое сообщение, которое показано на рисунке 6.5. Так же, если мы введем несуществующий id\_film или id\_cinema, то получим точно такую же ошибку, и новая запись не будет добавлена.

Это далеко не все возможности базы данных Cinemas, а лишь только часть. Все процедуры и их подробное описание находятся в разделе 3.

* 1. **Тестирование производительности**

Тестирование производительности проводилось на таблице Cinema, которая заполнена на 100000 строк. Далее вызывалась процедура GetCinemaByRegion, которая ищет кинотеатры в заданном регионе. Результат работы процедуры продемонстрирован на рисунке 6.6.



Рисунок 6.6 – Результат работы процедуры GetCinemaByRegion

На рисунке 6.7 показана стоимость Select запроса, который содержит в себе вызванная процедура.

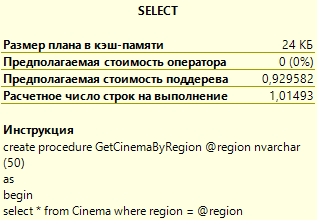


Рисунок 6.7 – Стоимость Select запроса в процедуре GetCinemaByRegion

Теперь, чтобы повысить производительность запроса, был создан некластеризованный индекс, который продемонстрирован на рисунке 6.8.



Рисунок 6.8 – Некластеризованный индекс таблицы Cinema

После создание индекса снова был выведен план запроса, рисунок 6.9.

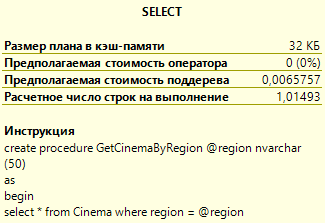


Рисунок 6.9 – Стоимость Select запроса после создания индекса

Как видно из рисунка 6.9, стоимость запроса уменьшилась примерно в 1,5 раза (пункт «Предполагаемая стоимость поддерева»). Таким образом, с помощью индексов мы можем увеличить производительность наших процедур, что позволит нам быстрее получать необходимую информацию.

# Заключение

В процессе решения поставленной задачи была достигнута поставленная цель по созданию базы данных «Сеть кинотеатров». Основой целью курсового проекта стало проектирование базы данных для дальнейшей интеграции с приложением. Разработанная база данных отвечает всем требованиям предметной области, ее таблицы созданной отвечают требованиям нормализации, что позволяет обеспечить целостность и непротиворечивость информации. База данных была разработана с помощью «Microsoft SQL Server». Помимо этого, была изучена и применена при разработке приложения технология настройки системы безопасности сервера СУБД.

При разработке базы данных:

– создано 14 таблиц;

– создано 39 хранимых процедур;

– создано 22 индексов;

– создано 2 пользователя базы данных;

– включено прозрачное шифрование;

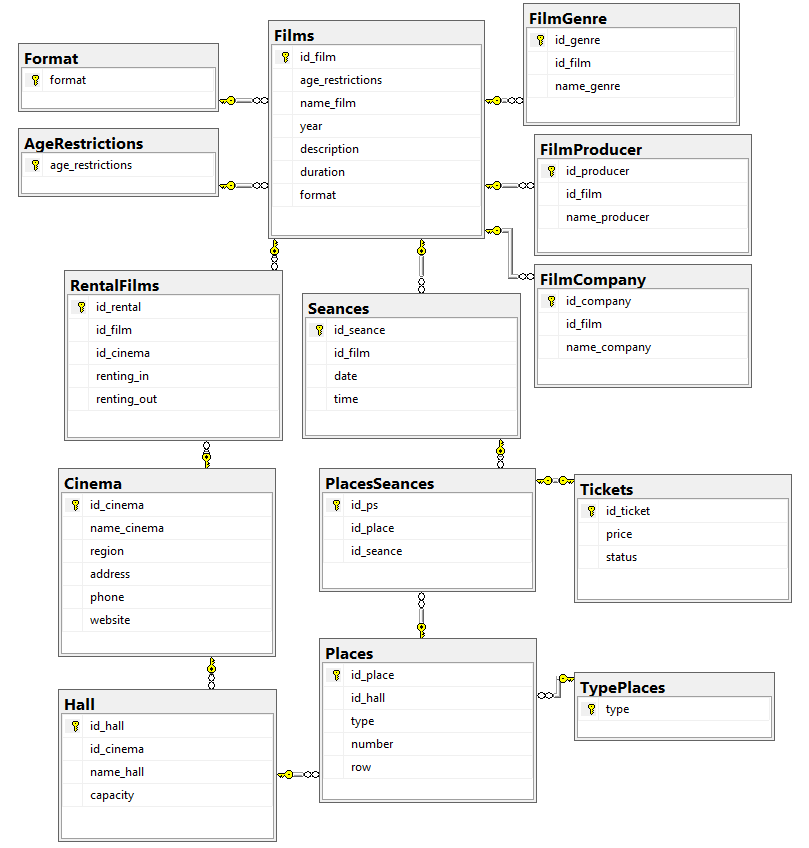
– подключен аудит.

В соответствии с полученным результатом работы можно сделать вывод, что разработанная БД работает верно, а требования технического задания выполнены в полном объёме.

# Список используемых источников

* 1. Документация СУБД «Microsoft SQL Server» [Электронный ресурс] – Режим доступа: <https://docs.microsoft.com/en-us/sql/?view=sql-server-ver15>
  2. Restore Statements (Transact-SQL) [Электронный ресурс] – Режим доступа: <https://docs.microsoft.com/en-us/sql/t-sql/statements/restore-statements-transact-sql?view=sql-server-ver15>
  3. Restore a Database Backup Using SSMS [Электронный ресурс] – Режим доступа: <https://docs.microsoft.com/en-us/sql/relational-databases/backup-restore/restore-a-database-backup-using-ssms?view=sql-server-ver15>.
  4. Шифрование в базах данных SQL Server [Электронный ресурс] – Режим доступа: <https://www.osp.ru/winitpro/2013/05/13035359>
  5. Основные средства обеспечения безопасности в SQL Server [Электронный ресурс] – Режим доступа: <https://winitpro.ru/index.php/2020/02/07/bezopasnost-sql-server/>
  6. Роли уровня базы данных [Электронный ресурс] – Режим доступа: <https://docs.microsoft.com/ru-ru/sql/relational-databases/security/authentication-access/database-level-roles?view=sql-server-ver15>
  7. Роли уровня сервера [Электронный ресурс] – Режим доступа: <https://docs.microsoft.com/ru-ru/sql/relational-databases/security/authentication-access/server-level-roles?view=sql-server-ver15>
  8. Итеративный учебник по SQL – [Электронный ресурс] – Режим доступа: <http://sql-tutorial.ru/ru/content.html>

# Приложение А



# Приложение Б

create table Cinema

(

id\_cinema int identity(1,1) primary key,

name\_cinema nvarchar(50) not null,

region nvarchar(30) not null,

address nvarchar(50) not null,

phone nvarchar(20) not null,

website nvarchar(100) not null

);

create table AgeRestrictions

(

age\_restrictions nvarchar(5) primary key

);

create table Format(

format nvarchar(5) primary key

);

create table Films

(

id\_film int identity(1,1) primary key,

age\_restrictions nvarchar(5) constraint Films\_AgeRestrictions\_FK foreign key references AgeRestrictions(age\_restrictions) on delete cascade not null,

name\_film nvarchar(30) not null,

year date not null,

description nvarchar(MAX) not null,

duration int not null,

format nvarchar(5) constraint Films\_Format\_FK foreign key references Format(format) on delete cascade not null

);

create table FilmCompany

(

id\_company int identity(1,1) primary key,

id\_film int constraint FileCompany\_Films\_FK foreign key references Films(id\_film) on delete cascade not null,

name\_сompany nvarchar(50) not null

);

create table FilmProducer

(

id\_producer int identity(1,1) primary key,

id\_film int constraint FileProducer\_Films\_FK foreign key references Films(id\_film) on delete cascade not null,

name\_producer nvarchar(50) not null

);

create table FilmGenre

(

id\_genre int identity(1,1) primary key,

id\_film int constraint FileGenre\_Films\_FK foreign key references Films(id\_film) on delete cascade not null,

name\_genre nvarchar(50) not null

);

create table RentalFilms

(

id\_rental int identity(1,1) primary key,

id\_film int constraint RentalFilms\_Films\_FK foreign key references Films(id\_film) on delete cascade not null,

id\_cinema int constraint RentalFilms\_Cinema\_FK foreign key references Cinema(id\_cinema) on delete cascade not null,

renting\_in date not null,

renting\_out date not null

);

create table Hall

(

id\_hall int identity(1,1) primary key,

id\_cinema int constraint Hall\_Cinema\_FK foreign key references Cinema(id\_cinema) on delete cascade not null,

name\_hall nvarchar(30) not null,

сapacity int not null

);

create table TypePlaces

(

type nvarchar(15) primary key

);

create table Places

(

id\_place int identity(1,1) primary key,

id\_hall int constraint Places\_Hall\_FK foreign key references Hall(id\_hall) on delete cascade not null,

type nvarchar(15) constraint Places\_TypePlaces\_FK foreign key references TypePlaces(type) on delete cascade not null,

number int not null,

row int not null

);

create table Seances

( id\_seance int identity(1,1) primary key,

id\_film int constraint Seances\_Films\_FK foreign key references Films(id\_film) on delete cascade not null,

date date not null,

time time not null

);

create table PlacesSeances

(

id\_ps int identity(1,1) primary key,

id\_place int constraint PlacesSeances\_Places\_FK foreign key references Places(id\_place) on delete cascade,

id\_seance int constraint PlacesSeances\_Seances\_FK foreign key references Seances(id\_seance) on delete cascade,

);

create table Tickets

(

id\_ticket int primary key constraint Tickets\_PlacesSeances\_FK foreign key references PlacesSeances(id\_ps) on delete cascade,

price int not null,

status nvarchar(15) not null

);