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

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«База данных для приложения «Сеть кинотеатров» с использованием утилиты ApexSQL»

Выполнил студент Невар Юлия Валерьевна

(Ф.И.О.)

Руководитель проекта ассистент Мороз Л.С.

(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: ассистент Мороз Л.С.

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

Нормоконтролер: ассистент Мороз Л.С.

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

Курсовой проект защищен с оценкой

Минск 2020

**Реферат**

Пояснительная записка курсового проекта содержит 62 страницы пояснительной записки, 16 иллюстраций, 8 источников литературы, 4 приложения.

Основная цель курсового проекта: проектирование базы данных для приложения «Сеть кинотеатров».

В первой главе осуществляется постановка задачи.

Вторая глава посвящена процессу разработки модели базы данных.

В третьей главе приведено описание разработки объектов базы данных.

В четвертой главе описаны процедуры экспорта и импорта данных.

В пятой главе описано тестирование производительности.

В шестой главе посвящена технологии дешифрования объектов базы данных.

В заключении приведены результаты проделанной работы.

**Abstract**

The explanatory note of the course project contains 62 pages of the explanatory note, 16 illustrations, 8 literature sources, 4 appendices.

The main goal of the course project is to design a database for the cinema Chain application.

In the first Chapter, the problem is set.

The second Chapter focuses on the process of developing a database model.

The third Chapter describes the development of database objects.

The fourth Chapter describes how to export and import data.

The fifth Chapter describes performance testing.

The sixth Chapter is devoted to the technology of decrypting database objects.

In conclusion, the results of the work done are presented.

**Содержание**

[Введение 6](#_Toc58852376)

[1. Постановка задачи 7](#_Toc58852377)

[2. Разработка модели базы данных 8](#_Toc58852378)

[3. Разработка необходимых объектов 10](#_Toc58852379)

[3.1. Таблицы 10](#_Toc58852380)

[3.2. Процедуры 14](#_Toc58852381)

[3.3. Представления 17](#_Toc58852382)

[4. Описание процедур импорта и экспорта 18](#_Toc58852383)

[5. Тестирование производительности 19](#_Toc58852384)

[6. Описание технологии 22](#_Toc58852385)

[Заключение 25](#_Toc58852386)

[Список используемых источников 26](#_Toc58852387)

[Приложение A 27](#_Toc58852388)

[Приложение Б 29](#_Toc58852389)

[Приложение B 58](#_Toc58852390)

[Приложение Г 59](#_Toc58852391)

# **Введение**

В современном мире базы данных применяются практически везде: в интернет магазинах, на предприятиях, в учебных и медицинских учреждениях. В связи с этим возникает потребность в создании специальных систем, которые ответственны за управление базами данных.

Язык SQL широко применяется в работе с базами данных. Значительная часть IT-сферы так или иначе связана с ним. Несмотря на свою кажущуюся простоту язык SQL предоставляет огромные возможности для работы с базами данных. На сегодняшний день самой распространённой является реляционная модель структурирования данных. Реляционная база данных – это база данных, основанная на реляционной модели данных. Она подразумевает под собой описание каждой сущности набором атрибутов, которые хранятся в соответствующих таблицах.

Целью данной курсовой работы является разработка реляционной базы данных «Movie\_theaters» для приложения «Сеть кинотеатров». База данных должна содержать необходимые данные о кинотеатрах, залах кинотеатров, фильмах, сеансах, билетах и заказах, учетные записи пользователей, а также хранить и шифровать их пароли.

В качестве СУБД для базы данных была выбрана Microsoft SQL Server 2018 в связи с ее простотой, удобством и надежностью.

Необходимо создать основные объекты такие как таблицы, процедуры, представления, индексы. В базе данных должна быть реализована возможность экспорта и импорта данных в таблицы базы данных. Также после создание объектов и добавления данных в таблицы, нужно протестировать базу данных на производительность, вследствие чего добавить необходимые индексы для таблиц. В качестве технологии для курсового проекта была выбрана технология дешифрования объектов базы данных при помощи утилиты ApexSQL.

1. Постановка задачи

Основной задачей курсового проекта является разработка базы данных «Movie\_theaters» для приложения «Сеть кинотеатров». Главная идея этой базы данных заключается в том, что пользователи могут просматривать доступные сеансы, а также бронировать на них билеты. Данная база должна облегчить поиск сеанса по кинотеатру, фильму или дате.

Задачу можно сформулировать следующим образом: создать необходимые сущности, которые будут отражать хранимые данные; охарактеризовать эти сущности с помощью соответствующих атрибутов, с корректно подобранными ограничениями целостности; связать таблицы между собой, тем самым получив полноценную реляционную структуру.

Необходимо реализовать и протестировать на производительность базы данных на таблицах, содержащих не менее 100 000 строк.

Так же необходимо реализовать технологию дешифрования с помощью утилиты ApexSQL.

Функционально должны быть выполнены следующие задачи:

* возможность создавать, удалять и редактировать кинотеатры;
* возможность создавать, удалять и редактировать фильмы;
* возможность добавлять, удалять и редактировать сеансы;
* возможность добавлять, удалять и редактировать залы кинотеатров;
* возможность добавлять, удалять и редактировать жанры;
* авторизация пользователя по логину и паролю, хранящемуся в базе в виде хэша;
* возможность просмотра расписания сеансов определенного кинотеатра в определенный день;
* возможность бронирования билетов пользователем;
* возможность просмотра списка кинотеатров;
* возможность просматривать сеансы выбранного кинотеатра;
* возможность удалять данные из всех таблиц.

1. Разработка модели базы данных

Одним из ключевых моментов при проектировании и создании базы данных является грамотный анализ предметной области приложения. Как следствие, составление такой модели данных, которая будет правильно отражать то, как с этими с данными в общем, и этой моделью, в частности, подразумевается взаимодействовать.

Результатом корректно проведённого анализа, проектирования, и разработки, является модель, способная предоставить функционально все необходимые возможности для пользователя. Можно выделить набор таких операций, как:

* возможность регистрации, редактирования и удаления пользователей;
* возможность добавлять, удалять и редактировать кинотеатры;
* возможность добавлять, удалять и редактировать фильмы;
* возможность добавлять, удалять и редактировать сеансы;
* возможность добавлять, удалять и редактировать залы кинотеатров;
* возможность добавлять, удалять и редактировать жанры;
* возможность вывода информации о доступных сеансах, актуальных фильмах.

Основой инфраструктуры базы данных является грамотно спроектированная модель, которая отображает связь пользовательских таблиц. Правильное и корректное взаимодействие их друг с другом как раз и заключается в схеме базы данных со связями, верно отображающими их положение.

На рисунке 2.1 отображена модель базы данных, полученная в ходе анализа предметной области и необходимого функционала. Между сущностями, представленными таблицами, также настроены связи, отражающие их взаимосвязь.

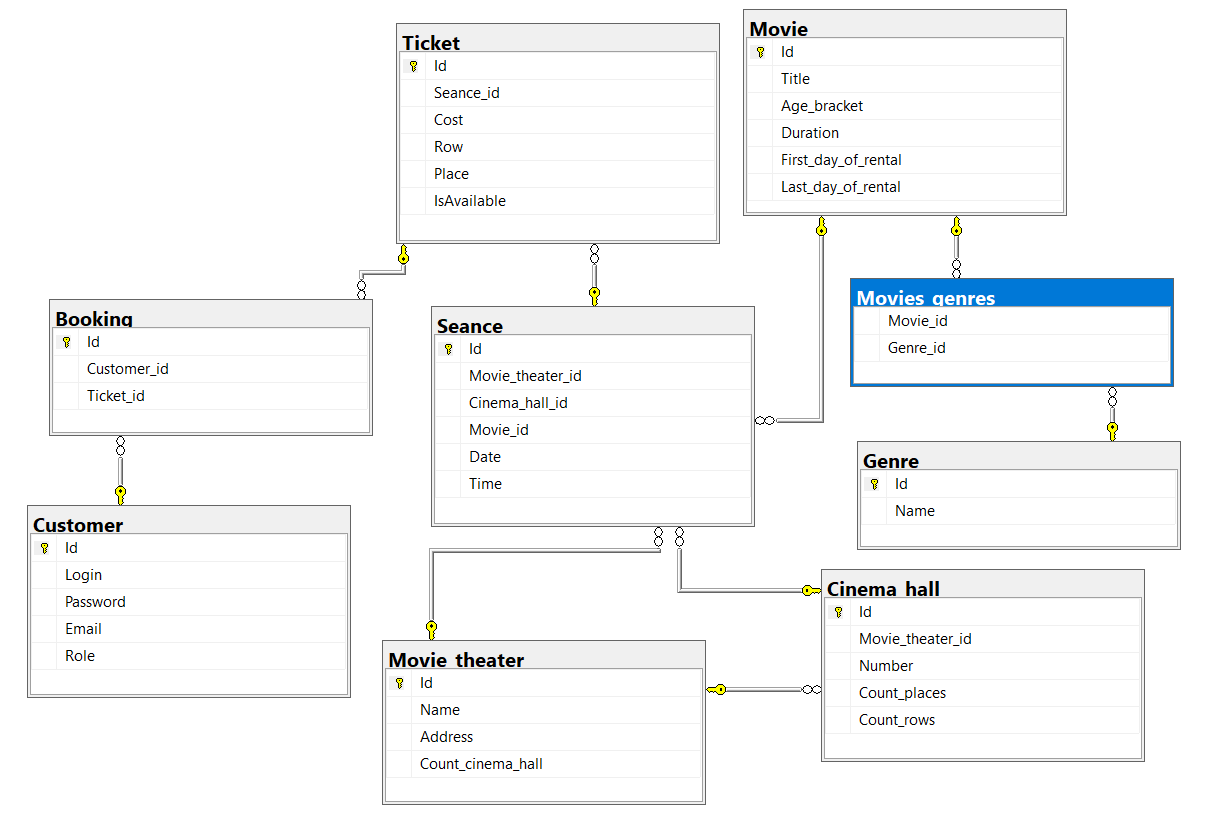


Рисунок 2.1 – Структурная схема базы данных

На рисунке 2.1 проиллюстрированы связи, которые использовались в разработке реляционной базы данных.

1. Разработка необходимых объектов

При разработке курсового проекта понадобились следующие объекты: таблицы, хранимые процедуры, представления и индексы.

* 1. Таблицы

Центральным объектом любой базы данных является связанная совокупность таблиц. Таблицы ограничивают и упорядочивают хранимую информацию, а также обеспечивают связанность за счет внешних ключей.

Для реализации базы данных «Movie\_theaters» было разработано 9 таблиц.

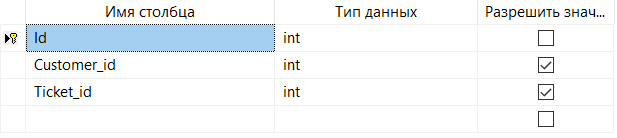


Рисунок 3.1 – Структура таблицы Booking

Таблица Booking (рисунок 3.1) содержит информацию о забронированных билетах и включает 3 столбца:

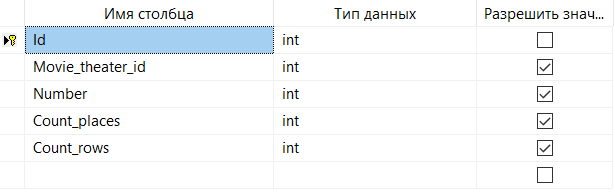
* столбец Id является первичным ключом, тип данных int, инкрементируется при добавлении новой записи;
* столбец Customer\_id является внешним ключом, хранит идентификатор пользователя, тип данных int, не может быть null;
* столбец Ticket\_id является внешним ключом, хранит идентификатор билета, тип данных int, не может быть null.

Рисунок 3.2 – Структура таблицы Cinema\_hall

Таблица Cinema\_hall (рисунок 3.2) содержит информацию о залах кинотеатров и включает 5 столбцов:

* столбец Id является первичным ключом, тип данных int, инкрементируется при добавлении зала кинотеатра;
* столбец Movie\_theater\_id является внешним ключом, хранит идентификатор кинотеатра, тип данных int, не может быть null;
* столбец Number хранит номер зала, тип данных int, не может быть null;
* столбец Count\_places хранит количество мест в зале, тип данных int, не может быть null;
* столбец Count\_rows хранит количество рядов в зале, тип данных int, не может быть null.

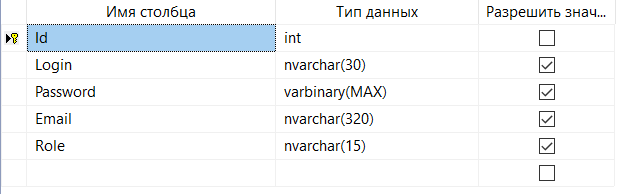


Рисунок 3.3 – Структура таблицы Customer

Таблица Customer содержит информацию о пользователях и включает 5 столбцов:

* столбец Id является первичным ключом, тип данных int, инкрементируется при добавлении пользователя;
* столбец Login хранит логин пользователя, тип данных nvarchar(30) и не может быть null;
* столбец Password хранит пароль пользователя в бинарном виде, тип данных varbinary(MAX), не может быть null;
* столбец Email хранит почту пользователя, тип данных nvarchar(320), не может быть null;
* столбец Role хранит роль пользователя, может быть User или Admin, тип данных nvarchar(15), не может быть null.

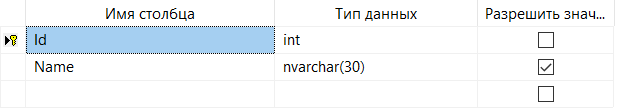


Рисунок 3.4 – Структура таблицы Genre

Таблица Genre (рисунок 3.4) содержит информацию о жанрах и включает 2 столбца:

* столбец Id является первичным ключом, тип данных int, инкрементируется при добавлении жанра;
* столбец Name хранит название жанра, тип данных nvarchar(30), не может быть null.

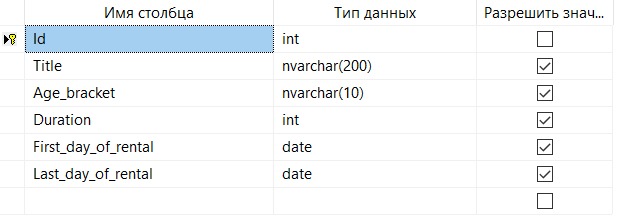


Рисунок 3.5 – Структура таблицы Movie

Таблица Movie (рисунок 3.5) содержит информацию о фильмах и включает 6 столбцов:

* столбец ID является первичным ключом, тип данных int, инкрементируется при добавлении фильма;
* столбец Title хранит название фильма, тип данных nvarchar(200), не может быть null;
* столбец Age\_bracket хранит возрастное ограничение фильма, тип данных nvarchar(10), не может быть null;
* столбец Duration хранит продолжительность фильма, тип данных int,не может быть null;
* столбец First\_day\_of\_rental хранит первый день проката фильма, тип данных date, не может быть null;
* столбец Last\_day\_of\_rental хранит последний день проката фильма, тип данных date, не может быть null.

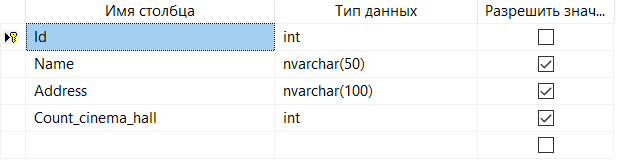


Рисунок 3.6 – Структура таблицы Movie\_theater

Таблица Movie\_theater (рисунок 3.6) содержит информацию о кинотеатрах и включает 4 столбца:

* столбец Id является первичным ключом, тип данных int, инкрементируется при добавлении кинотеатра;
* столбец Name хранит название кинотеатра, тип данных nvarchar(50), не может быть null;
* столбец Address хранит адрес кинотеатра, тип данных nvarchar(100), не может быть null;
* столбец Count\_cinema\_hall хранит количество залов в кинотеатре, тип данных int, не может быть null.

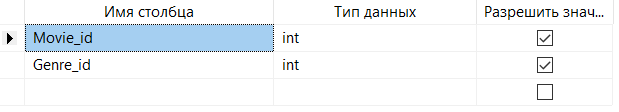


Рисунок 3.7 – Структура таблицы Movies\_genres

Таблица Movies\_genres (рисунок 3.7) связывает фильмы с соответствующими им жанрами и включает 2 столбца:

* столбец Movie\_id является внешним ключом, хранит идентификатор фильма, тип данных int, не может быть null;
* столбец Genre\_id является внешним ключом, хранит идентификатор жанра, тип данных int, не может быть null.

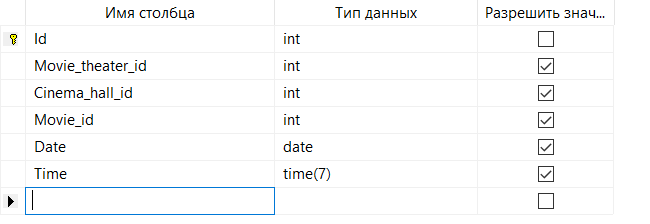


Рисунок 3.8 – Структура таблицы Seance

Таблица Seance (рисунок 3.8) содержит информацию о сеансах и включает 6 столбцов:

* столбец Id является первичным ключом, тип данных int, инкрементируется при добавлении сеанса;
* столбец Movie\_theater\_id является внешним ключом, хранит идентификатор кинотеатра, тип данных int, не может быть null;
* столбец Cinema\_hall\_id является внешним ключом, хранит идентификатор зала кинотеатра, тип данных int, не может быть null;
* столбец Movie\_id является внешним ключом, хранит идентификатор фильма, тип данных int, не может быть null;
* столбец Date хранит дату сеанса, тип данных date, не может быть null;
* столбец Time хранит время сеанса, тип данных time, не может быть null.

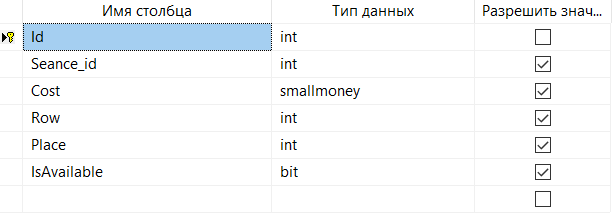


Рисунок 3.9 – Структура таблицы Ticket

Таблица Ticket (рисунок 3.9) содержит информацию о билетах и включает 6 столбцов:

* столбец Id является первичным ключом, тип данных int, инкрементируется при добавлении билета;
* столбец Seance\_id является внешним ключом, хранит идентификатор сеанса, тип данных int, не может быть null;
* столбец Cost хранит цену билета, тип данных smallmoney, не может быть null;
* столбец Row хранит номер ряда, тип данных int, не может быть null;
* столбец Place хранит номер места, тип данных int, не может быть null;
* столбец IsAvailable хранит информацию о том, забронирован ли билет, тип данных bit, не может быть null.

Листинг создания таблиц представлен в приложении А.

* 1. Процедуры

Процедура представляет собой модуль, выполняющий одно или несколько действий. Поскольку вызов процедуры в T-SQL является отдельным исполняемым оператором, блок кода T-SQL может состоять только из вызова процедуры. Процедуры относятся к числу ключевых компонентов модульного кода, обеспечивающих оптимизацию и повторное использование программной логики.

Для реализации базы данных «Movie\_theaters» было разработано 90 процедур.

Из них 70 связаны с CRUD:

* AddAdmin – добавление пользователя с ролью администартора;
* AddBooking – добавление заказа;
* AddCinemaHall – добавление зала кинотеатра;
* AddGenre – добавление жанра;
* AddMovie – добавление фильма;
* AddMovieGenre – добавление записи фильм-жанр;
* AddMovieTheater – добавление кинотеатра
* AddSeance – добавление сеанса;
* AddTicket – добавление билета;
* AddUser – добавление пользователя;
* DeleteAllBooking – удаление всех заказов;
* DeleteAllCinemaHalls – удаление всех залов;
* DeleteAllCustomers – удаление всех пользователей;
* DeleteAllGenres – удаление всех жанров;
* DeleteAllMovies – удаление всех фильмов;
* DeleteAllMoviesGenres – удаление всех записей фильм-жанр;
* DeleteAllMovieTheaters – удаление всех кинотеатров;
* DeleteAllSeances – удаление всех сеансов;
* DeleteAllTickets – удаление всех билетов;
* DeleteBooking – удаление заказа по Id;
* DeleteBookingByCustomer – удаление заказа по Customer\_id;
* DeleteBookingByTicket – удаление заказа по Ticket\_id;
* DeleteCinameHall – удаление зала по Id;
* DeleteCinemaHallByTheater – удаление зала по Movie\_theater\_id;
* DeleteCustomer – удаление пользователя по Id;
* DeleteGenre – удаление жанра по Id;
* DeleteMovie – удаление фильма по Id;
* DeleteMoviesGenres – удаление записи фильм-жанр по Movie\_id и Genre\_id;
* DeleteMoviesGenresByGenre – удаление записи фильм-жанр по Genre\_id;
* DeleteMoviesGenresByMovie – удаление записи фильм-жанр по Movie\_id;
* DeleteMovieTheater – удаление кинотеатра по Id;
* DeleteSeance – удаление сеанса по Id;
* DeleteSeanceByCinemaHall – удаление сеанса по Cinema\_hall\_id;
* DeleteSeanceByMovie – удаление сеанса по Movie\_id;
* DeleteSeanceByTheater – удаление сеанса по Movie\_theater\_id;
* DeleteTicket – удаление билета по Id;
* DeleteTicketBySeance – удаление билета по Séance\_id;
* GetAllBooking – получение всех заказов;
* GetAllCinemaHall– получение всех залов;
* GetAllCustomer– получение всех пользователей;
* GetAllGenre– получение всех жанров;
* GetAllMovies– получение всех фильмов;
* GetAllMoviesGenres– получение всех записей фильм-жанр;
* GetAllMovieTheater– получение всех кинотеатров;
* GetAllSeance– получение всех сеансов;
* GetAllTicket– получение всех билетов;
* GetCinemaHallById – получение зала по Id;
* GetCinemaHallBySeance – получение залов по Seance\_id;
* GetCustomerById – получение пользователя по Id;
* GetGenreById– получение жанра по Id;
* GetMovieById– получение фильма по Id;
* GetMovieBySeance– получение фильмов по Seance\_id;
* GetSeanceById– получение сеанса по Id;
* GetSeanceForTheater– получение сеансов по Theater\_id;
* GetTheaterByCinemaHall– получение кинотеатров по Cinema\_hall\_id;
* GetTheaterById– получение кинотеатра по Id;
* GetTheaterBySeance– получение кинотеатров по Seance\_id;
* GetTicketById– получение билета по Id;
* GetTicketForSeance– получение билетов по Seance\_id;
* ListGenreByMovieId– получение жанров по Movie\_id;
* UpdateBooking – редактирование заказа;
* UpdateCinemaHall– редактирование зала;
* UpdateCustomer– редактирование пользователя;
* UpdateGenre– редактирование жанра;
* UpdateMovie– редактирование фильма;
* UpdateMoviesGenres– редактирование записи фильм-жанр;
* UpdateMovieTheater– редактирование кинотеатра;
* UpdateSeance– редактирование сеанса;
* UpdateStatusTicket– редактирование статуса билета;
* UpdateTicket– редактирование билета.

Также 10 из них связаны с импортом и экспортом данных:

* ExportToXML– экспорт данных в формат XML;
* ImportFromXMLBooking – импорт данных о заказах из XML файла;
* ImportFromXMLCinemaHall – импорт данных о залах из XML файла;
* ImportFromXMLCustomer – импорт данных о пользователях из XML файла;
* ImportFromXMLGenre – импорт данных о жанрах из XML файла;
* ImportFromXMLMovie – импорт данных о фильмах из XML файла;
* ImportFromXMLMoviesGenres – импорт данных о записях фильм-жанр из XML файла;
* ImportFromXMLSeance – импорт данных о сеансах из XML файла;
* ImportFromXMLTheaters – импорт данных о кинотеатрах из XML файла;
* ImportFromXMLTicket – импорт данных о билетах из XML файла.

Остальные процедуры выполняют вспомогательные функции:

* Authorization – проверяет существует ли пользователь с данными логином и паролем;
* EmailExists – проверяет есть ли такая почта в таблице;
* GetIdByLogin – получает Id пользователя по его логину;
* GetRoleByLogin – получает роль пользователя по его логину;
* IsAvailableTicket – проверяет забронирован ли билет;
* LoginExists – проверяет есть ли такой логин в таблице;
* MoviePoster – выводит расписание сеансов для определенного кинотеатра и определенной даты;
* Movies – выводит фильмы с описанием;
* SeanceProcedure – генерация сеансов;
* TicketProcedure – генерация билетов.

Листинг создания процедур представлен в приложении Б.

* 1. Представления

Представления или Views представляют виртуальные таблицы. Но в отличие от обычных стандартных таблиц в базе данных представления содержат запросы, которые динамически извлекают используемые данные.

Представления дают нам ряд преимуществ. Они упрощают комплексные SQL-операции. Они защищают данные, так как представления могут дать доступ к части таблицы, а не ко всей таблице. Представления также позволяют возвращать отформатированные значения из таблиц в нужной и удобной форме.

Представление Cinema объединяет таблицы Movie\_theater и Cinema\_hall для того, чтобы можно было увидеть сколько у каждого кинотеатра залов и какая у них вместительность.

Листинг создания представления представлен в приложении В.

1. Описание процедур импорта и экспорта

Одной из задач по разработке инфраструктуры базы данных являлась реализация процедур, которые будут осуществлять экспорт и импорт данных в формате XML.

Файл XML является расширяемым видом языка разметки Markup Language. Такие файлы формата, представляют из себя документы, использующие теги в целях определения объектов, а также их атрибутов. XML формат, в отличие от HTML, наделяет пользователя возможностью самостоятельно задавать теги, которые применяет язык XML.

За экспорт отвечает хранимая процедура ExportToXML. Экспорт производится по все таблицам.

За импорт данных из файла отвечают хранимые процедуры ImportFromXMLBooking, ImportFromXMLCinemaHall, ImportFromXMLCustomer, ImportFromXMLGenre, ImportFromXMLMovie, ImportFromXMLMoviesGenres, ImportFromXMLSeance, ImportFromXMLTheaters, ImportFromXMLTicket.

Листинг создания процедур для импорта и экспорта представлен в приложении Г.

1. Тестирование производительности

Оптимизация запросов — процесс изменения запроса и/или структуры БД с целью уменьшения использования вычислительных ресурсов при выполнении запроса. Один и тот же результат может быть получен СУБД различными способами (планами выполнения запросов), которые могут существенно отличаться как по затратам ресурсов, так и по времени выполнения.

В MS SQL Server оптимизация запросом в основном заключается в построение индексов над таблицами, и изменением плана запроса. Индекс – это объект базы данных, предназначенный для ускорения запросов к данным в таблице базы данных. Индекс состоит из набора страниц, узлов индекса, которые организованы в виде древовидной структуры — сбалансированного дерева. Эта структура является иерархической по своей природе и начинается с корневого узла на вершине иерархии и конечных узлов, листьев. Есть два основных типа индексов: кластеризованные и некластеризованные индексы.

Рассмотрим следующие рекомендации при планировании стратегии индексирования:

* для таблиц, которые часто обновляются используйте как можно меньше индексов;
* если таблица содержит большое количество данных, но их изменения незначительны, тогда используйте столько индексов, сколько необходимо для улучшения производительности ваших запросов;
* уникальность значений в столбце влияет на производительность индекса, в общем случае, чем больше у вас дубликатов в столбце, тем хуже работает индекс;
* для кластеризованных индексов старайтесь использовать настолько короткие поля насколько это возможно. Наилучшим образом будет применение кластеризованного индекса на столбцах с уникальными значениями и не позволяющими использовать NULL. Вот почему первичный ключ часто используется как кластеризованный индекс.

Важной характеристикой кластеризованного индекса является то, что все значения отсортированы в определенном порядке либо возрастания, либо убывания. Таким образом, таблица или представление может иметь только один кластеризованный индекс.

Тестирование базы данных проекта будет производиться на запросе:



В базе данных данного курсового проекта в каждой таблице находится поле с уникальным значением поля — id типа int, соответственно каждая таблица содержит кластеризованный индекс. В данном случае будет производится поиск по полям Seance\_id и IsAvailable таблицы Ticket.

Ниже на рисунке 5.1 представлена план запроса при выполнении данного запроса.

|  |
| --- |
|  |

Рисунок 5.1 – План запроса с кластеризованным индексом

Как можно увидеть, стоимость запроса достаточно. Для таблицы Ticket был создан некластеризованный индекс:



План запроса с применением некластеризованного индекса представлен на рисунке 5.2.

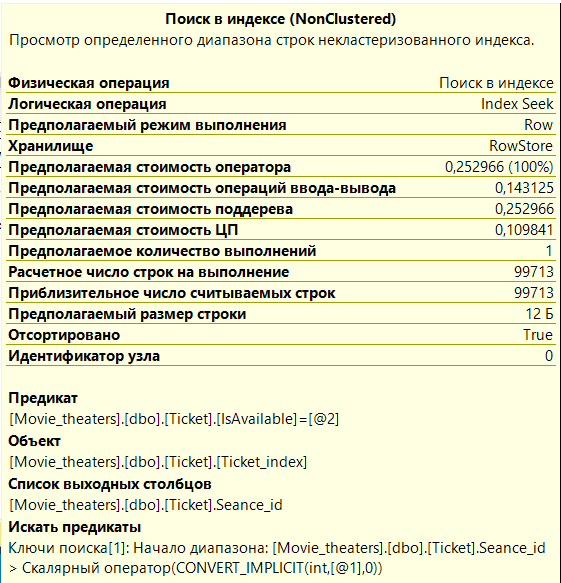


Рисунок 5.2 – План запроса с некластеризованным индексом

Как видно по планам запроса стоимость запроса с некластеризованным индексом меньше, чем с кластеризованным. В следствии этого можно сделать вывод, что мы ускорили выборку данных из таблицы.

1. Описание технологии

В наши дни всё больше программ переводятся в так называемый «web-ориентированный» вид, то есть используется принцип клиент-сервер, что позволяет хранить данные удалённо и получать к ним доступ через тонкий клиент (браузер). Одновременно с удобством использования встаёт вопрос о защищённости этих данных. Конфиденциальная информация может стать доступна другим людям. Например, на сервер могут быть произведены хакерские атаки, что позволит злоумышленникам похитить информацию.

Чтобы не допустить утечки информации, при создании процедур использовалась опция WITH ENCRYPTION (для шифрования объектов). Такие процедуры нельзя изменить или посмотреть их DDL скрипт дешифрации. Пример процедуры, созданной с применением опции WITH ENCRYPTION представлен на рисунке 6.1.

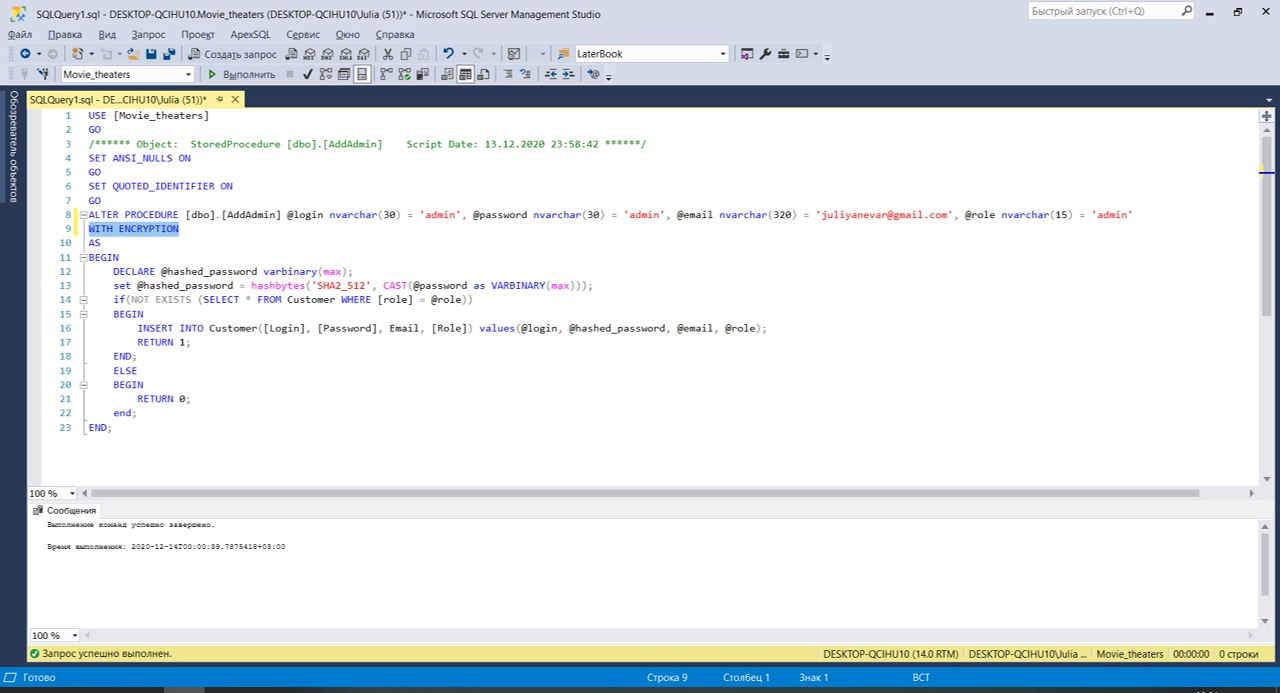


Рисунок 6.1 – Процедура AddAdmin, созданная с применением опции WITH ENCRYPTION

Для дешифрования процедур, зашифрованных таким образом, можно воспользоваться утилитой ApexSQL Decrypt.

В интерфейсе можно посмотреть DDL скрипт дешифрации процедуры (рисунок 6.2).

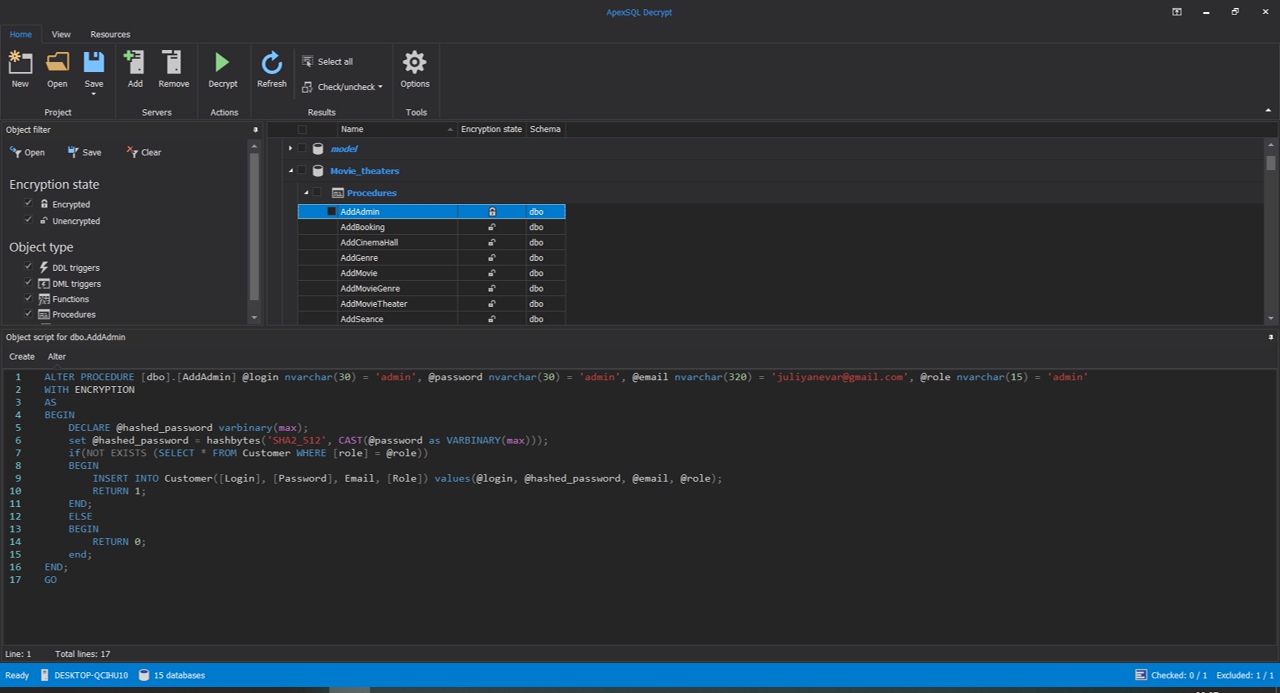


Рисунок 6.2 – Интерфейс утилиты ApexSQL Decrypt и DDL скрипт дешифрации процедуры AddAdmin

Дешифровать процедуру можно нажав правой клавишей мыши на зашифрованную процедуру и выбрать «Decrypt objects» (рисунок 6.3).

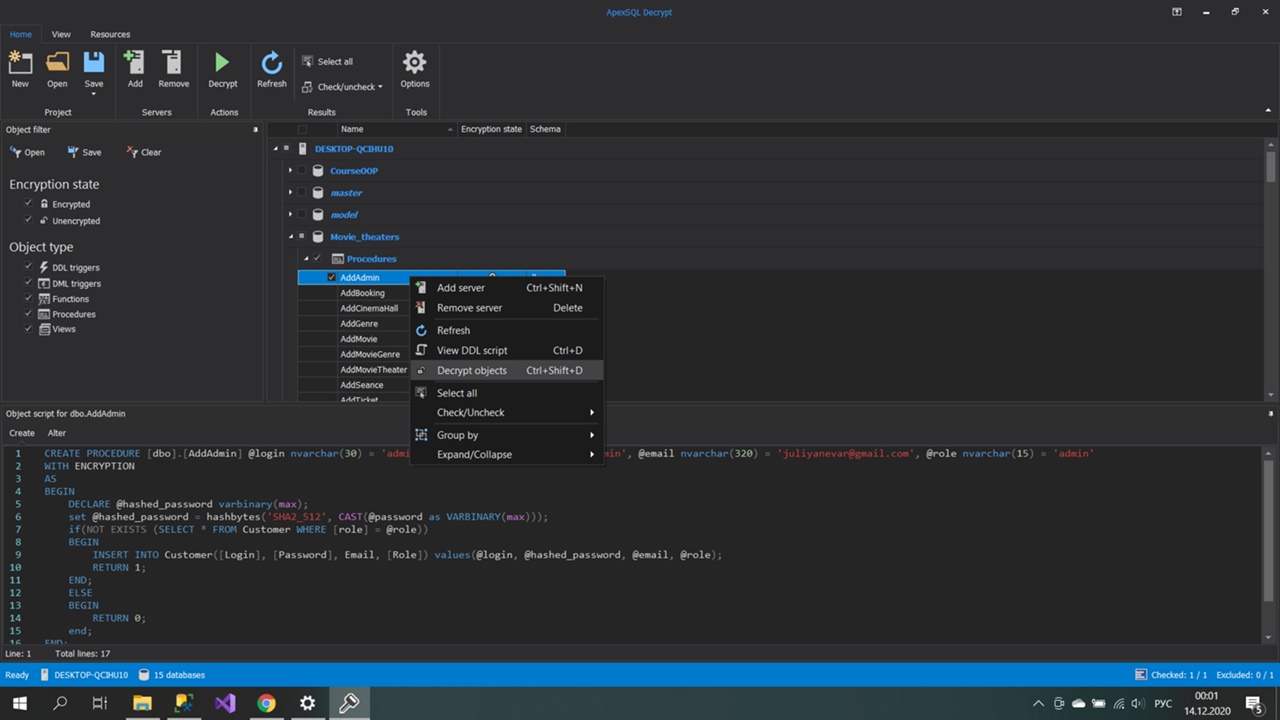


Рисунок 6.3 – Процесс дешифрации объекта из интерфейса Decrypt

Также дешифровать объект можно в Decrypt из интерфейса SSMS (рисунок 6.4).

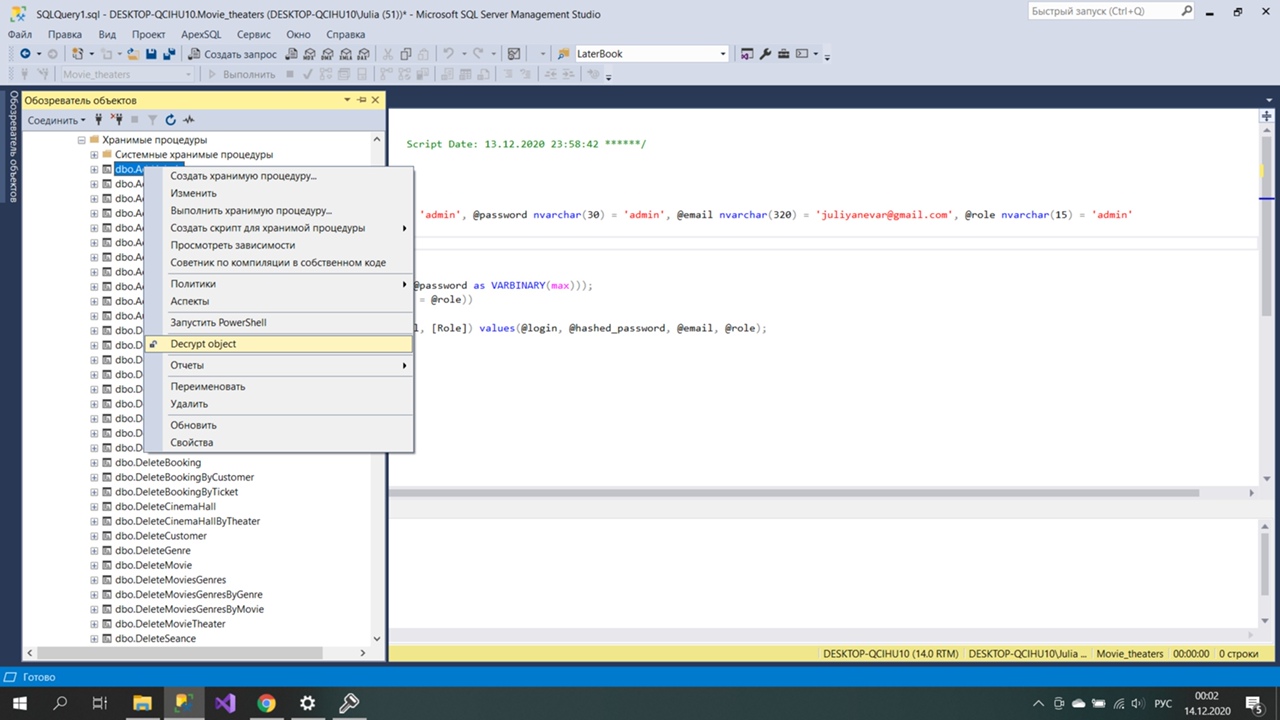


Рисунок 6.4 – Процесс дешифрации объекта из интерфейса SSMS

Заключение

В данном курсовом проекте была спроектирована и создана база данных «Movie\_theaters» для приложения «Сеть кинотеатров». База данных состоит из 9 таблиц, также содержит 90 процедур, 1 некластеризованный индекс и 1 представление. Данная база данных позволяет хранить информацию о кинотеатрах, фильмах, жанрах, сеансах, пользователях и их заказах.

Обратиться к данным, изменить их или удалить можно с помощью созданных процедур. Данные процедуры позволяют автоматизировать операции добавления, изменения или получения данных. Также были созданы процедуры для импорта и экспорта данных из XML файла.

Была изучена и использована технология дешифрации объектов с помощью утилиты ApexSQL. Данная технология позволяет дешифровать объекты базы данных без потери информации.

Разработанная база данных была создана для хранения информации, используемой в прототипе приложения «Сеть кинотеатров». Все обращения к данным, хранящимся в базе, были произведены через процедуры, что позволило ускорить работу приложения. Также процедуры содержат определенные проверки, что не позволяет клиенту добавить некорректные данные в базу данных.

Таким образом, разработанное программное средство и спроектированная база данных удовлетворяют всем требованиям технического задания.

# **Список используемых источников**

1. Блинова Е.А. Курс лекций + лабораторные работы по базам данных/ Е.А. Блинова
2. Свободная энциклопедия Википедия [Электронный ресурс] /CRUD – Режим доступа – URL: https://ru.wikipedia.org/wiki/CRUD
3. Об индексах в MS SQL Server [Электронный ресурс] / Режим доступа – URL: <https://www.sql.ru/articles/mssql/03013101indexes.shtml>.
4. Базы Данных. Учебное пособие [Электронный ресурс] /Базы данных – Режим доступа – URL: <https://siblec.ru/informatika-i-vychislitelnaya-tekhnika/bazy-dannykh>
5. Об процедурах в MS SQL Server [Электронный ресурс] / Режим доступа – URL: <https://metanit.com/sql/sqlserver/11.1.php>
6. Microsoft Docs – URL: https://docs.microsoft.com/ru-ru/sql/?view=sql-server-ver15
7. METANIT SQL – URL: https://metanit.com/sql/
8. METANIT C# - URL: https://metanit.com/sharp/

Приложение A

create database Movie\_theaters;

use Movie\_theaters;

create table Movie\_theater

(

Id int identity(1,1) primary key,

[Name] nvarchar(50) unique,

[Address] nvarchar(100) unique,

Count\_cinema\_hall int

)

create table Customer

(

Id int identity(1,1) primary key,

[Login] nvarchar(30) unique,

[Password] varbinary(max),

Email nvarchar(320) unique,

[Role] nvarchar(15) default 'user' check ([Role] in ('admin', 'user'))

)

create table Movie

(

Id int identity(1,1) primary key,

Title nvarchar(200),

Age\_bracket nvarchar(10),

Duration int,

First\_day\_of\_rental date,

Last\_day\_of\_rental date

)

create table Genre

(

Id int identity(1,1) primary key,

[Name] nvarchar(30) unique

)

create table Movies\_genres

(

Movie\_id int foreign key references Movie(Id),

Genre\_id int foreign key references Genre(Id)

)

create table Cinema\_hall

(

Id int identity(1,1) primary key,

Movie\_theater\_id int foreign key references Movie\_theater(Id),

[Number] int,

Count\_places int,

[Count\_rows] int

)

create table Seance

(

Id int identity(1,1) primary key,

Movie\_theater\_id int foreign key references Movie\_theater(Id),

Cinema\_hall\_id int foreign key references Cinema\_hall(Id),

Movie\_id int foreign key references Movie(Id),

[Date] date,

[Time] time

)

create table Ticket

(

Id int identity(1,1) primary key,

Seance\_id int foreign key references Seance(Id),

Cost smallmoney,

[Row] int,

Place int

)

create table Booking

(

Id int identity(1,1) primary key,

Customer\_id int foreign key references Customer(Id),

Ticket\_id int foreign key references Ticket(Id)

)

create index Ticket\_index on Ticket(Seance\_id, IsAvailable)

Приложение Б

create procedure AddMovieGenre @id\_movie int, @name\_genre nvarchar(30)

as

begin

declare @id\_genre int;

if EXISTS(select \* from Movie where Id=@id\_movie)

begin

if EXISTS(select \* from genre where [name]=@name\_genre)

begin

set @id\_genre=(select id from genre where [name]=@name\_genre);

insert into Movies\_genres(Movie\_id,Genre\_id) values(@id\_movie, @id\_genre);

return 1;

end

else

begin

return 0;

end

end

else

begin

return 0;

end

end;

create procedure Movies

as

begin

declare cursorMovie cursor local static

for select id, title, Age\_bracket, Duration, First\_day\_of\_rental, Last\_day\_of\_rental from Movie order by title;

declare cursorGenre cursor local static

for select g.Id, g.Name, mg.Movie\_id from Genre g join Movies\_genres mg on g.id=mg.Genre\_id;

declare @id\_movie int, @title nvarchar(200), @age nvarchar(10), @duration int, @first\_day date, @last\_day date,

@fetchst1 int, @fetchst2 int, @id\_genre int, @name nvarchar(30),@id\_movie1 int,@genre nvarchar(350);

open cursorMovie;

fetch cursorMovie into @id\_movie, @title, @age, @duration, @first\_day, @last\_day;

set @fetchst1=@@FETCH\_STATUS;

while @fetchst1=0

begin

print 'Movie: '+@title;

print ' Age bracket: '+@age;

print ' Duration: '+cast(@duration as varchar(10));

print ' First day of rental: '+cast(@first\_day as varchar(30));

print ' Last day of rental: '+cast(@last\_day as varchar(30));

open cursorGenre;

fetch cursorGenre into @id\_genre, @name, @id\_movie1;

set @fetchst2=@@FETCH\_STATUS;

set @genre=' Genre: ';

while @fetchst2=0

begin

if(@id\_movie=@id\_movie1)

begin

set @genre=@genre+' '+@name;

end;

fetch cursorGenre into @id\_genre, @name, @id\_movie1;

set @fetchst2 = @@FETCH\_STATUS;

end;

print @genre;

close cursorGenre;

fetch cursorMovie into @id\_movie, @title, @age, @duration, @first\_day, @last\_day;

set @fetchst1 = @@FETCH\_STATUS;

end;

close cursorMovie;

end;

create procedure AddSeance @id\_cinema\_hall int, @id\_movie int, @date date, @time time

as

begin

declare @id\_movie\_th int = (select Movie\_theater\_id from Cinema\_hall where Id=@id\_cinema\_hall);

declare @first\_day date = (select First\_day\_of\_rental from Movie where Id=@id\_movie);

declare @last\_day date = (select Last\_day\_of\_rental from Movie where Id=@id\_movie);

declare @seance\_id int;

if(not exists(select \* from Seance where date=@date and time=@time))

begin

if((@date>@first\_day and @date<@last\_day) or @date=@first\_day or @date=@last\_day)

begin

insert into Seance(Movie\_theater\_id, Cinema\_hall\_id, Movie\_id, [Date], [Time])

values(@id\_movie\_th, @id\_cinema\_hall, @id\_movie, @date, @time);

set @seance\_id=(select Id from Seance

where Movie\_theater\_id = @id\_movie\_th and Cinema\_hall\_id = @id\_cinema\_hall and Movie\_id = @id\_movie and [Date] = @date and [Time] = @time);

return @seance\_id;

end;

else

begin

return 0;

end;

end;

else

begin

return 0;

end;

end;

se

create procedure SeanceProcedure1 @date date

as

begin

declare @count\_movies int = (select count(\*) from Movie);

declare @count\_halls int = ( select count(\*) from Cinema\_hall);

declare @time time, @x\_hall int =(select top(1) Id from Cinema\_hall), @x\_time int = 1, @number\_movie int,@seance\_id int;

while(@x\_hall<(@count\_halls+1))

begin

set @x\_time=1;

while(@x\_time<7)

begin

if(@x\_time=1)

begin

set @time='10:00';

end;

else if(@x\_time=2)

begin

set @time='12:00';

end;

else if(@x\_time=3)

begin

set @time='14:00';

end;

else if(@x\_time=4)

begin

set @time='16:00';

end;

else if(@x\_time=5)

begin

set @time='18:00';

end;

else if(@x\_time=6)

begin

set @time='20:00';

end;

set @number\_movie=ABS(CHECKSUM(NewId()) % @count\_movies)+1;

exec @seance\_id= AddSeance @x\_hall, @number\_movie, @date, @time;

exec TicketProcedure @seance\_id;

set @x\_time = @x\_time + 1;

end;

set @x\_hall = @x\_hall + 1;

end;

end;

create procedure AddTicket @id\_seance int, @cost money, @row int, @place int

as

begin

if(not exists(select \* from Ticket where Seance\_id=@id\_seance and [Row]=@row and Place=@place))

begin

insert into Ticket(Seance\_id, Cost, [Row], Place)

values(@id\_seance, @cost, @row, @place);

return 1;

end;

else

begin

return 0;

end;

end;

create procedure TicketProcedure @seance\_id int

as

begin

declare @count\_seance int = (select count(\*) from seance);

declare @row int = 1, @place int = 1, @cinema\_hall\_id int, @count\_rows int, @count\_places int, @cost money = 7.5;

set @row = 1;

set @cinema\_hall\_id = (select cinema\_hall\_id from Seance where id = @seance\_id);

set @count\_rows = (select [Count\_rows] from Cinema\_hall where id = @cinema\_hall\_id);

set @count\_places = (select Count\_places from Cinema\_hall where id = @cinema\_hall\_id)/@count\_rows;

while(@row<(@count\_rows+1))

begin

set @place = 1;

while(@place<(@count\_places+1))

begin

exec AddTicket @seance\_id, @cost, @row, @place;

set @place = @place + 1;

end;

set @row = @row + 1;

end;

set @seance\_id = @seance\_id + 1;

end;

create PROCEDURE AddUser @login nvarchar(30), @password nvarchar(30), @email nvarchar(320)

AS

BEGIN

DECLARE @hashed\_password varbinary(max);

set @hashed\_password = hashbytes('SHA2\_512', CAST(@password as VARBINARY(max)));

begin try

insert into Customer([Login], [Password], Email) values(@login, @hashed\_password, @email)

return 1;

end try

begin catch

RETURN 0;

end catch;

end;

create PROCEDURE AddAdmin @login nvarchar(30) = 'admin', @password nvarchar(30) = 'admin', @email nvarchar(320) = 'juliyanevar@gmail.com', @role nvarchar(15) = 'admin'

AS

BEGIN

DECLARE @hashed\_password varbinary(max);

set @hashed\_password = hashbytes('SHA2\_512', CAST(@password as VARBINARY(max)));

if(NOT EXISTS (SELECT \* FROM Customer WHERE [role] = @role))

BEGIN

INSERT INTO Customer([Login], [Password], Email, [Role]) values(@login, @hashed\_password, @email, @role);

RETURN 1;

END;

ELSE

BEGIN

RETURN 0;

end;

END;

CREATE PROCEDURE AddBooking @id\_user int, @id\_ticket int

AS

BEGIN

IF(NOT EXISTS(SELECT \* FROM Booking WHERE Ticket\_id = @id\_ticket))

BEGIN

IF((SELECT [Role] FROM Customer WHERE Id=@id\_user) = 'user')

BEGIN

INSERT INTO Booking(Customer\_id, Ticket\_id) values(@id\_user, @id\_ticket);

RETURN 1;

END;

ELSE

BEGIN

RETURN 0;

END;

END;

ELSE

BEGIN

RETURN 0;

END;

END;

CREATE PROCEDURE AddMovie @title nvarchar(200), @age\_bracket nvarchar(10), @duration int, @first\_day date, @last\_day date

AS

BEGIN

IF(NOT exists(SELECT \* FROM Movie WHERE title=@title AND First\_day\_of\_rental=@first\_day))

BEGIN

INSERT INTO Movie(title, Age\_bracket, Duration, First\_day\_of\_rental, Last\_day\_of\_rental)

values(@title, @age\_bracket, @duration, @first\_day, @last\_day);

RETURN 1;

END;

ELSE

BEGIN

RETURN 0;

END;

END;

CREATE PROCEDURE GetAllMovies

AS

BEGIN

SELECT \* FROM Movie;

END;

CREATE PROCEDURE UpdateMovie @id int, @title nvarchar(200), @age\_bracket nvarchar(10), @duration int, @first\_day date, @last\_day date

AS

BEGIN

IF(exists(SELECT \* FROM Movie WHERE id=@id))

BEGIN

UPDATE Movie SET title=@title WHERE id=@id;

UPDATE Movie SET Age\_bracket=@age\_bracket WHERE id=@id;

UPDATE Movie SET Duration=@duration WHERE id=@id;

UPDATE Movie SET First\_day\_of\_rental=@first\_day WHERE id=@id;

UPDATE Movie SET Last\_day\_of\_rental=@last\_day WHERE id=@id;

RETURN 1;

END;

ELSE

BEGIN

RETURN 0;

END;

END;

CREATE PROCEDURE AddMovieTheater @name nvarchar(50), @address nvarchar(100), @count\_hall int

AS

BEGIN

IF(NOT exists(SELECT \* FROM Movie\_theater WHERE [name]=@name))

BEGIN

INSERT INTO Movie\_theater([Name], [Address], Count\_cinema\_hall) values(@name, @address, @count\_hall);

RETURN 1;

END;

ELSE

BEGIN

RETURN 0;

END;

END;

CREATE PROCEDURE AddGenre @name nvarchar(30)

AS

BEGIN

if(NOT EXISTS(SELECT \* FROM Genre WHERE [name]=@name))

BEGIN

INSERT INTO Genre([Name]) VALUES(@name);

RETURN 1;

END;

ELSE

BEGIN

RETURN 0;

END;

END;

CREATE PROCEDURE AddCinemaHall @id\_movie\_theater int, @number int, @count\_places int, @count\_rows int

AS

BEGIN

IF(NOT EXISTS(SELECT \* FROM Cinema\_hall WHERE Movie\_theater\_id=@id\_movie\_theater AND [Number]=@number))

BEGIN

INSERT INTO Cinema\_hall(Movie\_theater\_id, [Number], Count\_places, [Count\_rows]) values(@id\_movie\_theater, @number, @count\_places, @count\_rows);

RETURN 1;

END;

ELSE

BEGIN

RETURN 0;

END;

END;

CREATE PROCEDURE GetAllMovieTheater

AS

BEGIN

SELECT \* FROM Movie\_theater;

END;

CREATE PROCEDURE GetAllCustomer

AS

BEGIN

SELECT \* FROM Customer;

END;

CREATE PROCEDURE GetAllGenre

AS

BEGIN

SELECT \* FROM Genre;

END;

CREATE PROCEDURE GetAllMoviesGenres

AS

BEGIN

SELECT \* FROM Movies\_genres;

END;

CREATE PROCEDURE GetAllCinemaHall

AS

BEGIN

SELECT \* FROM Cinema\_hall;

END;

CREATE PROCEDURE GetAllSeance

AS

BEGIN

SELECT \* FROM Seance;

END;

CREATE PROCEDURE GetAllTicket

AS

BEGIN

SELECT \* FROM Ticket;

END;

CREATE PROCEDURE GetAllBooking

AS

BEGIN

SELECT \* FROM Booking;

END;

CREATE PROCEDURE UpdateMovieTheater @id int, @name nvarchar(50), @address nvarchar(100), @countHall int

AS

BEGIN

if(EXISTS(SELECT \* FROM Movie\_theater WHERE Id=@id))

BEGIN

UPDATE Movie\_theater SET [Name]=@name WHERE Id=@id;

UPDATE Movie\_theater SET [Address]=@address WHERE Id=@id;

UPDATE Movie\_theater SET Count\_cinema\_hall=@countHall WHERE Id=@id;

RETURN 1;

END;

ELSE

BEGIN

RETURN 0;

END;

END;

CREATE PROCEDURE UpdateCustomer @id int,@login nvarchar(30), @password nvarchar(30), @email nvarchar(320), @role nvarchar(15)

AS

BEGIN

DECLARE @hashed\_password varbinary(max);

set @hashed\_password = hashbytes('SHA2\_512', CAST(@password as VARBINARY(max)));

IF(exists(SELECT \* FROM Customer WHERE Id=@id))

BEGIN

UPDATE Customer SET [Login]=@login WHERE Id=@id;

UPDATE Customer SET [Password]=@hashed\_password WHERE Id=@id;

UPDATE Customer SET Email=@email WHERE Id=@id;

UPDATE Customer SET [Role]=@role WHERE Id=@id;

RETURN 1;

END;

ELSE

BEGIN

RETURN 0;

END;

END;

CREATE PROCEDURE UpdateGenre @id int,@name nvarchar(30)

AS

BEGIN

if(exists(SELECT \* FROM Genre WHERE Id=@id))

BEGIN

UPDATE Genre SET [Name]=@name WHERE Id=@id;

RETURN 1;

END;

ELSE

BEGIN

RETURN 0;

END;

END;

CREATE PROCEDURE UpdateMoviesGenres @id\_movie int, @id\_genre\_old int, @id\_genre\_new int

AS

BEGIN

if(exists(SELECT \* FROM Movies\_genres WHERE Movie\_id=@id\_movie AND Genre\_id=@id\_genre\_old))

BEGIN

UPDATE Movies\_genres SET Genre\_id=@id\_genre\_new WHERE Movie\_id=@id\_movie AND Genre\_id=@id\_genre\_old;

RETURN 1;

END;

ELSE

BEGIN

RETURN 0;

END;

END;

CREATE PROCEDURE UpdateCinemaHall @id int, @id\_theater int, @number int, @places int, @rows int

AS

BEGIN

IF(EXISTS(SELECT \* FROM Cinema\_hall WHERE Id=@id))

BEGIN

UPDATE Cinema\_hall SET Movie\_theater\_id=@id\_theater WHERE Id=@id;

UPDATE Cinema\_hall SET [Number]=@number WHERE Id=@id;

UPDATE Cinema\_hall SET Count\_places=@places WHERE Id=@id;

UPDATE Cinema\_hall SET [Count\_rows]=@rows WHERE Id=@id;

RETURN 1;

END;

ELSE

BEGIN

RETURN 0;

END;

END;

CREATE PROCEDURE UpdateSeance @id int, @id\_theater int, @id\_hall int, @id\_movie int, @date date, @time time

AS

BEGIN

if(exists(SELECT \* FROM Seance WHERE Id=@id))

BEGIN

UPDATE Seance SET Movie\_theater\_id=@id\_theater WHERE Id=@id;

UPDATE Seance SET Cinema\_hall\_id=@id\_hall WHERE Id=@id;

UPDATE Seance SET Movie\_id=@id\_movie WHERE Id=@id;

UPDATE Seance SET [Date]=@date WHERE Id=@id;

UPDATE Seance SET [Time]=@time WHERE Id=@id;

RETURN 1;

END;

ELSE

BEGIN

RETURN 0;

END;

END;

CREATE PROCEDURE UpdateTicket @id int, @id\_seance int, @cost smallmoney, @row int, @place int

AS

BEGIN

if(exists(SELECT \* FROM Ticket WHERE Id=@id))

BEGIN

UPDATE Ticket SET Seance\_id=@id\_seance WHERE Id=@id;

UPDATE Ticket SET Cost=@cost WHERE Id=@id;

UPDATE Ticket SET [Row]=@row WHERE Id=@id;

UPDATE Ticket SET Place=@place WHERE Id=@id;

RETURN 1;

END;

ELSE

BEGIN

RETURN 0;

END;

END;

CREATE PROCEDURE UpdateBooking @id int, @id\_customer int, @id\_ticket int

AS

BEGIN

if(exists(SELECT \* FROM Booking WHERE Id=@id))

BEGIN

UPDATE Booking SET Customer\_id=@id\_customer WHERE Id=@id;

UPDATE Booking SET Ticket\_id=@id\_ticket WHERE Id=@id;

RETURN 1;

END;

ELSE

BEGIN

RETURN 0;

END;

END;

CREATE PROCEDURE DeleteBooking @id int

AS

BEGIN

if(exists(SELECT \* FROM Booking WHERE Id=@id))

BEGIN

DELETE Booking WHERE Id=@id;

RETURN 1;

END;

ELSE

BEGIN

RETURN 0;

END;

END;

CREATE PROCEDURE DeleteMoviesGenres @id\_movie int, @id\_genre int

AS

BEGIN

if(exists(SELECT \* FROM Movies\_genres WHERE Movie\_id=@id\_movie AND Genre\_id=@id\_genre))

BEGIN

DELETE Movies\_genres WHERE Movie\_id=@id\_movie AND Genre\_id=@id\_genre;

RETURN 1;

END;

ELSE

BEGIN

RETURN 0;

END;

END;

CREATE PROCEDURE DeleteMoviesGenresByMovie @id\_movie int

AS

BEGIN

if(exists(SELECT \* FROM Movies\_genres WHERE Movie\_id=@id\_movie))

BEGIN

DELETE Movies\_genres WHERE Movie\_id=@id\_movie;

RETURN 1;

END;

ELSE

BEGIN

RETURN 0;

END;

END;

CREATE PROCEDURE DeleteMoviesGenresByGenre @id\_genre int

AS

BEGIN

if(exists(SELECT \* FROM Movies\_genres WHERE Genre\_id=@id\_genre))

BEGIN

DELETE Movies\_genres WHERE Genre\_id=@id\_genre;

RETURN 1;

END;

ELSE

BEGIN

RETURN 0;

END;

END;

CREATE PROCEDURE DeleteGenre @id int

AS

BEGIN

if(exists(SELECT \* FROM Genre WHERE Id=@id))

BEGIN

EXEC DeleteMoviesGenresByGenre @id;

DELETE Genre WHERE Id=@id;

RETURN 1;

END;

ELSE

BEGIN

RETURN 0;

END;

END;

CREATE PROCEDURE DeleteBookingByTicket @id\_ticket int

AS

BEGIN

if(exists(SELECT \* FROM Booking WHERE Ticket\_id=@id\_ticket))

BEGIN

DELETE Booking WHERE Ticket\_id=@id\_ticket;

RETURN 1;

END;

ELSE

BEGIN

RETURN 0;

END;

END;

CREATE PROCEDURE DeleteTicket @id int

AS

BEGIN

if(exists(SELECT \* FROM Ticket WHERE Id=@id))

BEGIN

EXEC DeleteBookingByTicket @id;

DELETE Ticket WHERE Id=@id;

RETURN 1;

END;

ELSE

BEGIN

RETURN 0;

END;

END;

USE Movie\_theaters;

CREATE PROCEDURE DeleteTicketBySeance @id\_seance int

AS

BEGIN

DECLARE @id\_ticket int;

if(exists(SELECT \* FROM Ticket WHERE Seance\_id=@id\_seance))

BEGIN

DECLARE Tickets CURSOR LOCAL for SELECT Id FROM Ticket WHERE Seance\_id=@id\_seance;

OPEN Tickets;

FETCH Tickets INTO @id\_ticket;

WHILE @@FETCH\_STATUS=0

BEGIN

EXEC DeleteBookingByTicket @id\_ticket;

FETCH Tickets INTO @id\_ticket;

END;

CLOSE Tickets;

DELETE Ticket WHERE Seance\_id=@id\_seance;

RETURN 1;

END;

ELSE

BEGIN

RETURN 0;

END;

END;

CREATE PROCEDURE DeleteSeance @id int

AS

BEGIN

if(exists(SELECT \* FROM Seance WHERE Id=@id))

BEGIN

EXEC DeleteTicketBySeance @id;

DELETE Seance WHERE Id=@id;

RETURN 1;

END;

ELSE

BEGIN

RETURN 0;

END;

END;

USE Movie\_theaters;

CREATE PROCEDURE DeleteBookingByCustomer @id\_customer int

AS

BEGIN

if(exists(SELECT \* FROM Booking WHERE Customer\_id=@id\_customer))

BEGIN

DELETE Booking WHERE Customer\_id=@id\_customer;

RETURN 1;

END;

ELSE

BEGIN

RETURN 0;

END;

END;

CREATE PROCEDURE DeleteCustomer @id int

AS

BEGIN

if(exists(SELECT \* FROM Customer WHERE Id=@id))

BEGIN

EXEC DeleteBookingByCustomer @id;

DELETE Customer WHERE Id=@id;

RETURN 1;

END;

ELSE

BEGIN

RETURN 0;

END;

END;

create PROCEDURE DeleteSeanceByCinemaHall @id\_hall int

AS

BEGIN

DECLARE @id int;

if(exists(SELECT \* FROM Seance WHERE Cinema\_hall\_id=@id\_hall))

BEGIN

declare Seances cursor local for SELECT Id FROM Seance WHERE Cinema\_hall\_id=@id\_hall;

open Seances;

fetch Seances into @id;

while @@FETCH\_STATUS=0

begin

EXEC DeleteTicketBySeance @id;

fetch Seances into @id;

end;

close Seances;

DELETE Seance WHERE Cinema\_hall\_id=@id\_hall;

RETURN 1;

END;

ELSE

BEGIN

RETURN 0;

END;

END;

CREATE PROCEDURE DeleteCinemaHall @id int

AS

BEGIN

if(EXISTS(SELECT \* FROM Cinema\_hall WHERE Id=@id))

BEGIN

EXEC DeleteSeanceByCinemaHall @id;

DELETE Cinema\_hall WHERE Id=@id;

RETURN 1;

END;

ELSE

BEGIN

RETURN 0;

END;

END;

create PROCEDURE DeleteSeanceByMovie @id\_movie int

AS

BEGIN

DECLARE @id int;

if(exists(SELECT \* FROM Seance WHERE Movie\_id=@id\_movie))

BEGIN

declare Seances cursor local for SELECT Id FROM Seance WHERE Movie\_id=@id\_movie;

open Seances;

fetch Seances into @id;

while @@FETCH\_STATUS=0

begin

EXEC DeleteTicketBySeance @id;

fetch Seances into @id;

end;

close Seances;

DELETE Seance WHERE Movie\_id=@id\_movie;

RETURN 1;

END;

ELSE

BEGIN

RETURN 0;

END;

END;

create procedure DeleteMovie @id int

as

begin

if(exists(select \* from Movie where Id=@id))

begin

exec DeleteSeanceByMovie @id;

exec DeleteMoviesGenresByMovie @id;

delete Movie where Id=@id;

return 1;

end;

else

begin

return 0;

end;

end;

create PROCEDURE DeleteCinemaHallByTheater @id\_theater int

AS

BEGIN

declare @id int;

if(EXISTS(SELECT \* FROM Cinema\_hall WHERE Movie\_theater\_id=@id\_theater))

BEGIN

declare Halls cursor local for SELECT Id FROM Cinema\_hall WHERE Movie\_theater\_id=@id\_theater;

open Halls;

fetch Halls into @id;

while @@FETCH\_STATUS=0

begin

EXEC DeleteSeanceByCinemaHall @id;

fetch Halls into @id;

end;

DELETE Cinema\_hall WHERE Movie\_theater\_id=@id\_theater;

RETURN 1;

END;

ELSE

BEGIN

RETURN 0;

END;

END;

create PROCEDURE DeleteSeanceByTheater @id\_theater int

AS

BEGIN

DECLARE @id int;

if(exists(SELECT \* FROM Seance WHERE Movie\_theater\_id=@id\_theater))

BEGIN

declare Seances cursor local for SELECT Id FROM Seance WHERE Movie\_theater\_id=@id\_theater;

open Seances;

fetch Seances into @id;

while @@FETCH\_STATUS=0

begin

EXEC DeleteTicketBySeance @id;

fetch Seances into @id;

end;

close Seances;

DELETE Seance WHERE Movie\_theater\_id=@id\_theater;

RETURN 1;

END;

ELSE

BEGIN

RETURN 0;

END;

END;

create procedure DeleteMovieTheater @id int

as

begin

if(exists(select \* from Movie\_theater where Id=@id))

begin

exec DeleteSeanceByTheater @id;

exec DeleteCinemaHallByTheater @id;

delete Movie\_theater where Id=@id;

return 1;

end;

else

begin

return 0;

end;

end;

create procedure DeleteAllBooking

as

begin

delete Booking;

DBCC CHECKIDENT ('Booking', RESEED, 0);

end;

create procedure DeleteAllCustomers

as

begin

exec DeleteAllBooking;

delete Customer;

DBCC CHECKIDENT ('Customer', RESEED, 0);

end;

create procedure DeleteAllTickets

as

begin

exec DeleteAllBooking;

delete Ticket;

DBCC CHECKIDENT ('Ticket', RESEED, 0);

end;

create procedure DeleteAllMoviesGenres

as

begin

delete Movies\_genres;

DBCC CHECKIDENT ('Movies\_genres', RESEED, 0);

end;

create procedure DeleteAllGenres

as

begin

exec DeleteAllMoviesGenres;

delete Genre;

DBCC CHECKIDENT ('Genre', RESEED, 0);

end;

create procedure DeleteAllSeances

as

begin

exec DeleteAllTickets;

delete Seance;

DBCC CHECKIDENT ('Seance', RESEED, 0);

end;

create procedure DeleteAllMovies

as

begin

exec DeleteAllMoviesGenres;

exec DeleteAllSeances;

delete Movie;

DBCC CHECKIDENT ('Movie', RESEED, 0);

end;

create procedure DeleteAllCinemaHalls

as

begin

exec DeleteAllSeances;

delete Cinema\_hall;

DBCC CHECKIDENT ('Cinema\_hall', RESEED, 0);

end;

create procedure DeleteAllMovieTheaters

as

begin

exec DeleteAllCinemaHalls;

delete Movie\_theater;

DBCC CHECKIDENT ('Movie\_theater', RESEED, 0);

end;

create procedure MoviePoster @id\_theater int, @date date

as

begin

select mt.[Name] as Theater , ch.Number as [Number of hall], m.Title, s.[Date], s.[Time]

from Seance s join Movie\_theater mt on s.Movie\_theater\_id=mt.Id

join Cinema\_hall ch on s.Cinema\_hall\_id=ch.Id

join Movie m on s.Movie\_id=m.Id

where s.Movie\_theater\_id=@id\_theater and s.[Date]=@date order by [Time];

end;

create procedure GetTheaterById @id\_movie int

as

begin

if(exists(select \* from Movie\_theater where Id=@id\_movie))

begin

select \* from Movie\_theater where Id=@id\_movie;

return 1;

end;

else

begin

return 0;

end;

end;

create procedure GetCustomerById @id int

as

begin

if(exists(select \* from Customer where Id=@id))

begin

select \* from Customer where Id=@id;

return 1;

end;

else

begin

return 0;

end;

end;

create procedure LoginExists @login nvarchar(30)

as

begin

if(exists(select id from Customer where [Login]=@login))

begin

select \* from Customer where [Login]=@login;

end;

end;

create procedure EmailExists @email nvarchar(30)

as

begin

if(exists(select id from Customer where Email=@email))

begin

select \* from Customer where Email=@email;

end;

end;

create procedure [Authorization] @login nvarchar(30), @password nvarchar(30)

as

begin

DECLARE @hashed\_password varbinary(max);

set @hashed\_password = hashbytes('SHA2\_512', CAST(@password as VARBINARY(max)));

if(exists(select \* from Customer where [Login]=@login and [Password]=@hashed\_password))

begin

select \* from Customer where [Login]=@login and [Password]=@hashed\_password;

end;

end;

create procedure GetRoleByLogin @login nvarchar(30)

as

begin

select [Role] from Customer where [Login]=@login;

end;

create procedure GetMovieById @id int

as

begin

select \* from Movie where Id=@id;

end;

create procedure GetGenreById @id int

as

begin

select \* from Genre where Id=@id;

end;

create procedure ListGenreByMovieId @id\_movie int

as

begin

select g.Id, g.[Name]

from Movies\_genres mg join Genre g

on mg.Genre\_id=g.Id

where Movie\_id=@id\_movie;

end;

create procedure GetTheaterByCinemaHall @id\_hall int

as

begin

select mt.Id, mt.Name, mt.Address, mt.Count\_cinema\_hall from Movie\_theater mt join Cinema\_hall ch

on mt.Id = ch.Movie\_theater\_id

where ch.Id=@id\_hall;

end;

create procedure GetCinemaHallById @id int

as

begin

select \* from Cinema\_hall where Id=@id;

end;

create procedure GetTheaterBySeance @id int

as

begin

select mt.Id, mt.Name, mt.Address, mt.Count\_cinema\_hall

from Movie\_theater mt join Seance s

on

mt.Id=s.Movie\_theater\_id

where s.Id=@id;

end;

create procedure GetCinemaHallBySeance @id int

as

begin

select ch.Id, ch.Movie\_theater\_id, ch.Number, ch.Count\_places, ch.Count\_rows

from Cinema\_hall ch join Seance s

on

ch.Id=s.Cinema\_hall\_id

where s.Id=@id;

end;

create procedure GetMovieBySeance @id int

as

begin

select m.Id, m.Title, m.Age\_bracket, m.Duration, m.First\_day\_of\_rental, m.Last\_day\_of\_rental

from Movie m join Seance s

on

m.Id=s.Movie\_id

where s.Id=@id;

end;

create procedure GetSeanceById @id int

as

begin

select \* from Seance where Id=@id;

end;

create procedure GetSeanceForTheater @id int

as

begin

select \* from Seance where Movie\_theater\_id=@id order by date,time;

end;

create procedure IsAvailableTicket @id\_seance int, @row int, @place int

as

begin

declare @id int =(select id from Ticket where Seance\_id=@id\_seance and [Row]=@row and Place=@place)

if(exists(select \* from Booking where Ticket\_id=@id))

begin

select \* from Ticket where Id=@id;

end;

end;

create procedure GetTicketForSeance @id\_seance int

as

begin

select \* from Ticket where Seance\_id=@id\_seance and IsAvailable=0;

end;

create procedure UpdateStatusTicket @id\_seance int, @row int, @place int

as

begin

update Ticket set IsAvailable=1 where Seance\_id=@id\_seance and [Row]=@row and Place=@place;

end;

create procedure GetTicketById @id int

as

begin

select \* from Ticket where Id=@id;

end;

create procedure GetIdByLogin @login nvarchar(30)

as

begin

select \* from Customer where [login]=@login;

end;

Приложение B

create view Cinema as select m.Id, m.Name, m.Address, c.Number [Number of hall], c.Count\_places, c.Count\_rows, c.Count\_places/c.Count\_rows [Count places on rows]

from Movie\_theater m join Cinema\_hall c

on m.id=c.Movie\_theater\_id;

Приложение Г

Импорт

use Movie\_theaters;

create procedure ImportFromXMLTheaters

as

begin

insert into Movie\_theater([Name], [Address], Count\_cinema\_hall)

select

my\_xml.rec.query('Name').value('.', 'nvarchar(50)'),

my\_xml.rec.query('Address').value('.', 'nvarchar(100)'),

my\_xml.rec.query('Count\_cinema\_hall').value('.', 'int')

from (select cast(my\_xml as xml)

from openrowset(bulk 'C:\temp.xml', single\_blob) as t(my\_xml)) as t(my\_xml)

cross apply my\_xml.nodes('Theaters/Movie\_theaters/Movie\_theater') as my\_xml (rec);

end;

create procedure ImportFromXMLCustomer

as

begin

insert into Customer([Login], [Password], Email, [Role])

select

my\_xml.rec.query('Login').value('.', 'nvarchar(30)'),

my\_xml.rec.query('Password').value('.', 'varbinary(max)'),

my\_xml.rec.query('Email').value('.', 'nvarchar(320)'),

my\_xml.rec.query('Role').value('.', 'nvarchar(15)')

from (select cast(my\_xml as xml)

from openrowset(bulk 'C:\temp.xml', single\_blob) as t(my\_xml)) as t(my\_xml)

cross apply my\_xml.nodes('Theaters/Customers/Customer') as my\_xml (rec);

end;

create procedure ImportFromXMLMovie

as

begin

insert into Movie(Title, Age\_bracket, Duration, First\_day\_of\_rental, Last\_day\_of\_rental)

select

my\_xml.rec.query('Title').value('.', 'nvarchar(200)'),

my\_xml.rec.query('Age\_bracket').value('.', 'nvarchar(10)'),

my\_xml.rec.query('Duration').value('.', 'int'),

my\_xml.rec.query('First\_day\_of\_rental').value('.', 'date'),

my\_xml.rec.query('Last\_day\_of\_rental').value('.', 'date')

from (select cast(my\_xml as xml)

from openrowset(bulk 'C:\temp.xml', single\_blob) as t(my\_xml)) as t(my\_xml)

cross apply my\_xml.nodes('Theaters/Movies/Movie') as my\_xml (rec);

end;

create procedure ImportFromXMLGenre

as

begin

insert into Genre([Name])

select

my\_xml.rec.query('Name').value('.', 'nvarchar(30)')

from (select cast(my\_xml as xml)

from openrowset(bulk 'C:\temp.xml', single\_blob) as t(my\_xml)) as t(my\_xml)

cross apply my\_xml.nodes('Theaters/Genres/Genre') as my\_xml (rec);

end;

create procedure ImportFromXMLMoviesGenres

as

begin

insert into Movies\_genres(Movie\_id, Genre\_id)

select

my\_xml.rec.query('Movie\_id').value('.', 'int'),

my\_xml.rec.query('Genre\_id').value('.', 'int')

from (select cast(my\_xml as xml)

from openrowset(bulk 'C:\temp.xml', single\_blob) as t(my\_xml)) as t(my\_xml)

cross apply my\_xml.nodes('Theaters/Movies\_genres/Movie\_tgenre') as my\_xml (rec);

end;

create procedure ImportFromXMLCinemaHall

as

begin

insert into Cinema\_hall(Movie\_theater\_id, [Number], Count\_places, [Count\_rows])

select

my\_xml.rec.query('Movie\_theater\_id').value('.', 'int'),

my\_xml.rec.query('Number').value('.', 'int'),

my\_xml.rec.query('Count\_places').value('.', 'int'),

my\_xml.rec.query('Count\_rows').value('.', 'int')

from (select cast(my\_xml as xml)

from openrowset(bulk 'C:\temp.xml', single\_blob) as t(my\_xml)) as t(my\_xml)

cross apply my\_xml.nodes('Theaters/Cinema\_halls/Cinema\_hall') as my\_xml (rec);

end;

create procedure ImportFromXMLSeance

as

begin

insert into Seance(Movie\_theater\_id, Cinema\_hall\_id, Movie\_id, [Date], [Time])

select

my\_xml.rec.query('Movie\_theater\_id').value('.', 'int'),

my\_xml.rec.query('Cinema\_hall\_id').value('.', 'int'),

my\_xml.rec.query('Movie\_id').value('.', 'int'),

my\_xml.rec.query('Date').value('.', 'date'),

my\_xml.rec.query('Time').value('.', 'time')

from (select cast(my\_xml as xml)

from openrowset(bulk 'C:\temp.xml', single\_blob) as t(my\_xml)) as t(my\_xml)

cross apply my\_xml.nodes('Theaters/Seances/Seance') as my\_xml (rec);

end;

create procedure ImportFromXMLTicket

as

begin

insert into Ticket(Seance\_id, Cost, [Row], Place)

select

my\_xml.rec.query('Seance\_id').value('.', 'int'),

my\_xml.rec.query('Cost').value('.', 'smallmoney'),

my\_xml.rec.query('Row').value('.', 'int'),

my\_xml.rec.query('Place').value('.', 'int')

from (select cast(my\_xml as xml)

from openrowset(bulk 'C:\temp.xml', single\_blob) as t(my\_xml)) as t(my\_xml)

cross apply my\_xml.nodes('Theaters/Tickets/Ticket') as my\_xml (rec);

end;

create procedure ImportFromXMLBooking

as

begin

insert into Booking(Customer\_id, Ticket\_id)

select

my\_xml.rec.query('Customer\_id').value('.', 'int'),

my\_xml.rec.query('Ticket\_id').value('.', 'int')

from (select cast(my\_xml as xml)

from openrowset(bulk 'C:\temp.xml', single\_blob) as t(my\_xml)) as t(my\_xml)

cross apply my\_xml.nodes('Theaters/Booking/Booking') as my\_xml (rec);

end;

Экспорт

use Movie\_theaters;

--drop procedure ExportToXML

create procedure ExportToXML

as

begin

--to use xp\_cmdshell

EXEC master.dbo.sp\_configure 'show advanced options', 1

RECONFIGURE WITH OVERRIDE;

EXEC master.dbo.sp\_configure 'xp\_cmdshell', 1

RECONFIGURE WITH OVERRIDE;

-- Save XML records to a file

declare @fileName nvarchar(500);

declare @sep char(1)=',';

declare @presqlStr varchar(50)='use Movie\_theaters; select ';

declare @postsqlStr varchar(100)=' for xml path(''''), type, root(''Theaters'')';

declare @theaterQ varchar(250)='(select Id, RTRIM([Name])as [Name], RTRIM([Address])as [Address], Count\_cinema\_hall from Movie\_theater for xml path(''Movie\_theater''),type) as Movie\_theaters ';

declare @customerQ varchar(250)='(select Id, RTRIM([Login])as [Login], [Password], RTRIM(Email)as Email, RTRIM([Role])as [Role] from Customer for xml path(''Customer''), type) as Customers ';

declare @movieQ varchar(250)='(select Id, RTRIM(Title)as Title, RTRIM(Age\_bracket)as Age\_bracket, Duration, First\_day\_of\_rental, Last\_day\_of\_rental from Movie for xml path(''Movie''), type) as Movies ';

declare @genreQ varchar(250)='(select Id, RTRIM([Name])as [Name] from Genre for xml path(''Genre''), type) as Genres ';

declare @movieGenreQ varchar(250)='(select Movie\_id, Genre\_id from Movies\_genres for xml path(''Movie\_genre''), type)as Movies\_genres ';

declare @hallQ varchar(250)='(select Id, Movie\_theater\_id, [Number], Count\_places, [Count\_rows] from Cinema\_hall for xml path(''Cinema\_hall''), type)as Cinema\_halls ';

declare @seanceQ varchar(250)='(select Id, Movie\_theater\_id, Cinema\_hall\_id, Movie\_id, [Date], [Time] from Seance for xml path(''Seance''), type)as Seances ';

declare @ticketQ varchar(250)='(select Id, Seance\_id, Cost, [Row], Place from Ticket for xml path(''Ticket''), type)as Tickets ';

declare @bookingQ varchar(250)='(select Id, Customer\_id, Ticket\_id from Booking for xml path(''Booking''), type)as Booking ';

declare @sqlStr varchar(3500)=@presqlStr+

@theaterQ+@sep+

@customerQ+@sep+

@movieQ+@sep+

@genreQ+@sep+

@movieGenreQ+@sep+

@hallQ+@sep+

@seanceQ+@sep+

@ticketQ+@sep+

@bookingQ+

@postsqlStr;

declare @sqlCmd varchar(4000);

set @fileName='C:\temp.xml';

SET @sqlCmd = 'bcp "' + @sqlStr + '" queryout ' + @fileName + ' -w -T';

EXEC xp\_cmdshell @sqlCmd;

end;