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:**

«Афиша “Мероприятий”»

Выполнил студент Конопацкий А.А.

(Ф.И.О.)

Руководитель проекта Мороз Л.С.

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

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

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

Курсовой проект защищен с оценкой

Реферат

Пояснительная записка курсового проекта содержит 28 страниц пояснительной записки, 31 иллюстраций, 7 источников литературы, 1 приложение.

.net framework, Windows Presentation Foundation, MICROSOFT SQL SERVER, MICROSOFT SQL SERVER MANAGEMENT STUDIO.

Целью курсового проекта является создание базы данных афиши мероприятий.

В первой главе проводится аналитический обзор прототипов по тематике курсового проекта и содержится описание технологий, использованных во время выполнения проекта.

Вторая глава посвящена процессу проектирования модели базы данных.

В третьей главе описывается процесс разработки, принципы функционирования и назначение созданных компонент проекта.

В четвёртой главе описаны процедуры для работы с XML.

В пятой главе содержатся сведения об использованной технологии.

Шестая глава посвящена тестированию производительности приложения и самого приложения.

В заключении приведены результаты проделанной работы.

Оглавление

[Введение 4](#_Toc58347973)

[1. Постановка задачи 5](#_Toc58347974)

[1.1. Обзор прототипов 5](#_Toc58347975)

[1.2. Анализ прототипов 7](#_Toc58347976)

[1.3. Описание используемых технологий 7](#_Toc58347977)

[2. Проектирование 9](#_Toc58347978)

[2.1. Структура модели базы данных 9](#_Toc58347979)

[3. Разработка необходимых объектов 10](#_Toc58347980)

[3.1. Таблицы 10](#_Toc58347981)

[3.2. Пользователи 14](#_Toc58347982)

[3.3. Функции 16](#_Toc58347983)

[3.4. Индексы 16](#_Toc58347984)

[3.5. Процедуры 16](#_Toc58347985)

[3.6. Триггеры 18](#_Toc58347986)

[4. Описание процедур импорта и экспорта данных 19](#_Toc58347987)

[4.1. Процедура импорта данных из XML-файла 19](#_Toc58347988)

[4.2. Процедура экспорта данных в XML-файл 19](#_Toc58347989)

[5. Описание технологии 20](#_Toc58347990)

[6. Тестирование 21](#_Toc58347991)

[6.1. Тестирование производительности базы данных. 21](#_Toc58347992)

[6.2. Тестирование работоспособности приложения. 23](#_Toc58347993)

[Заключение 27](#_Toc58347994)

[Список литературы 28](#_Toc58347995)

[Приложение А 29](#_Toc58347996)

## Введение

В данной записке приведено описание реляционной базы данных и desktop приложения, разработанных в соответствии с заданием на курсовое проектирование по теме «Афиша мероприятий» по дисциплине «Базы данных».

Целью моего курсового проекта является создание базы данных афиша меропрятий, ознакомление и применение мультимедийных типов данных в базе данных.

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;

В соответствии с заданием курсового проекта для проектирования базы данных афиша мероприятий используется система управления базами данных Microsoft SQL Server.

Также в задачу разработчика входит создание дружелюбного пользовательского интерфейса для совершения действий над базой данных афиша мероприятий. Для этого создаётся приложение, которое будет облегчать пользователю работу с базой данных.

Для разработки приложения использовалась технология WPF и язык программирования C#. Для взаимодействия приложения с базой данных использовалась технология ADO.NET.

## Постановка задачи

В соответствии с заданием курсового проекта следует не только создать базу данных, но и разработать программное средство, которое должно в полной мере показать возможности базы данных. Для того чтобы сформировать окончательные требования к проектируемому программному средству сначала рассмотрим прототипы из той же области.

## Обзор прототипов

Из открытых источников (интернет) были найдены другие приложения такой же тематики. Афиша TUT.BY (Рис 1.1.1) и Яндекс Афиша (Рис 1.1.3).

Афиша TUT.BY[1]

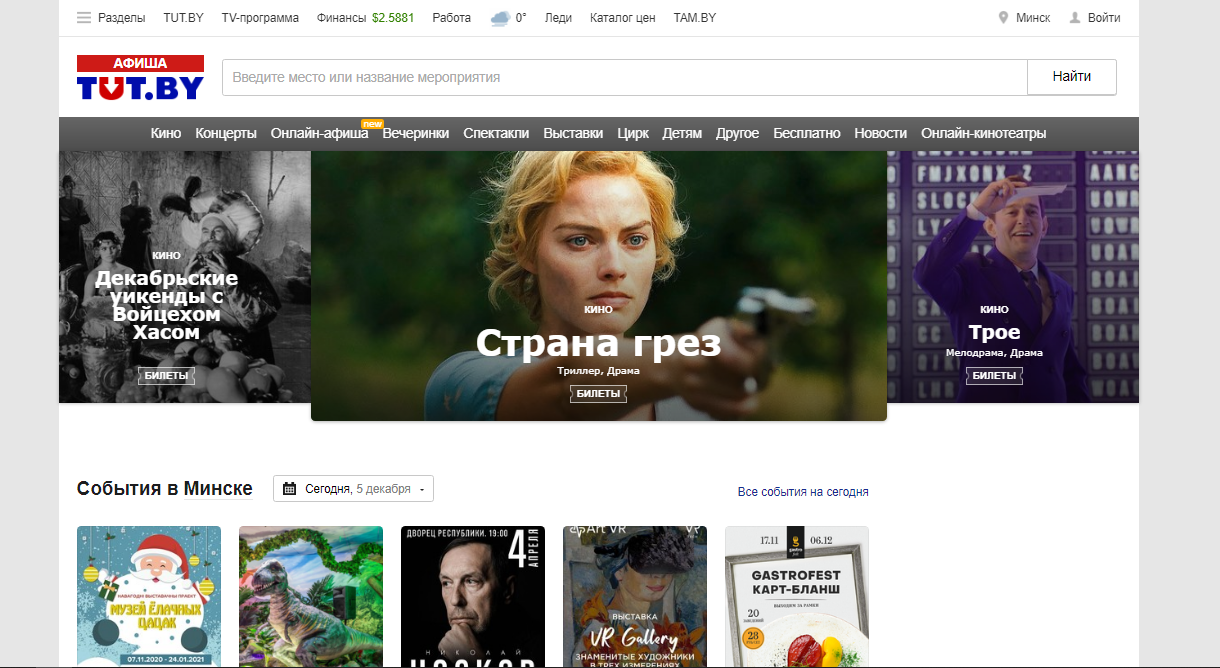


Рисунок 1.1. Главная страница Афиша TUT.BY.

Пример выбранного фильма в сервисе Афиша TUT.BY (Рис 1.1.1)

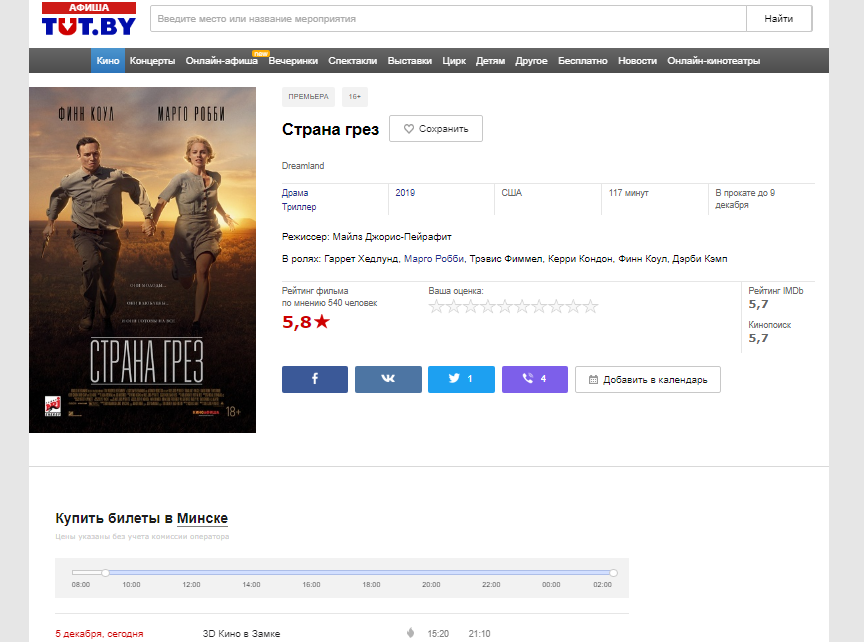


Рисунок 1.1.2 Интерфейс выбранного фильма.

Яндекс Афиша[2]

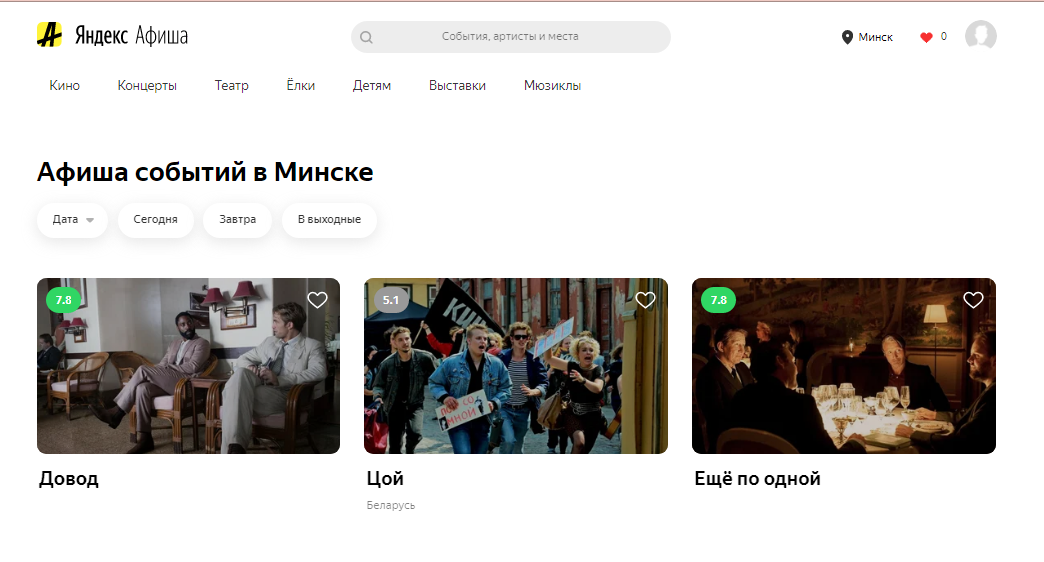


Рисунок 1.1.3 Главная страница Афиша TUT.BY.

Пример выбранного фильма в сервисе Яндекс Афиша (Рис 1.4)

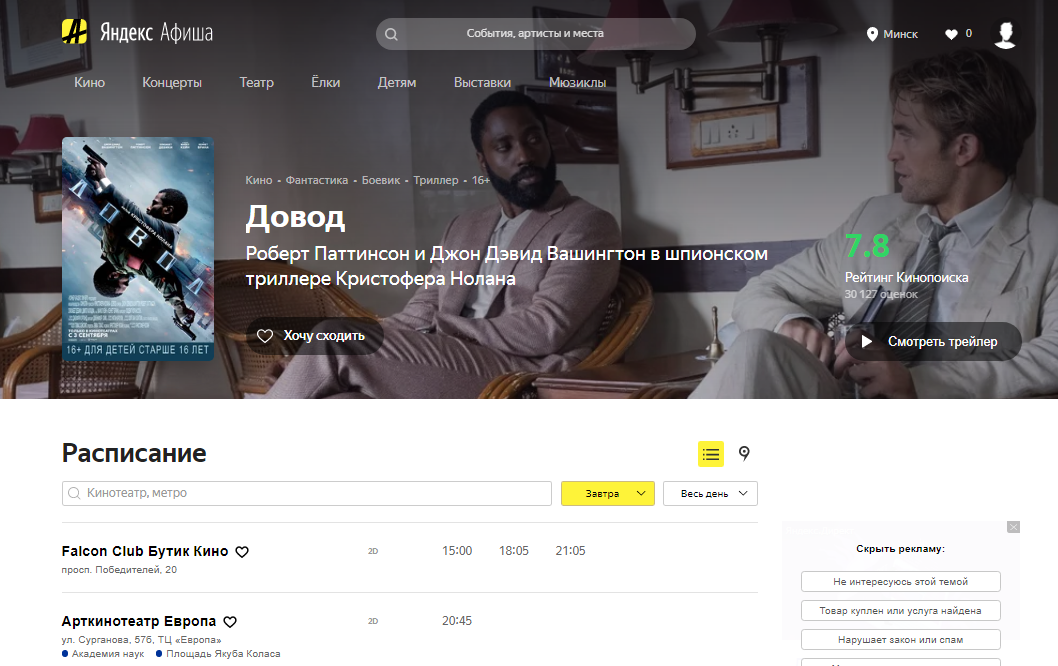


Рисунок 1.1.4 Интерфейс выбранного фильма

Функционал сервисов включает в себя:

* Регистрация и авторизация;
* Выбор досуга;
* Выбор мест проведения мероприятий;
* Фильтры;
* Поиск по мероприятиям и местам проведения;
* Бронирование билетов;

## Анализ прототипов

Проанализировав прототипы сайтов, можно отметить ожидаемую схожесть в их функциональности.

Исходя из полученных результатов, можно сформулировать основные функциональные требования курсовой работы:

* регистрация и авторизация;
* бронирование билетов;
* поиск мероприятия по названю;
* выбор кинофильмов по кинотеатрам;
* выбор концертов по концертным залам
* выбор даты, для последующего выбора досуга;
* добавление администратором мероприятий;
* редактирование данных о мероприятиях;

Данные требования будут реализованы в базе данных в виде хранимых процедур, которые описаны в главе 3 данной пояснительной записки.

## Описание используемых технологий

СУБД существует огромное множество: Oracle, MS SQL Server, Microsoft Access, MySql и так далее. Для организации работы с базой данных в данной работе было решено использовать одну из наиболее популярных СУБД «Microsoft SQL Server» [4], так как она предоставляет необходимые возможности оперирования объектами базы данных.

В качестве интерфейса прикладного программирования был выбран API-интерфейс — Windows Presentation Foundation (WPF) [5], платформа уровня представления для построения графических интерфейсов. Основывается на векторной системе визуализации и ориентирована на разработку клиентских Windows приложений, базирующихся на технологии Microsoft.NET.

WPF предоставляет средства для создания визуального интерфейса, включая язык XAML (eXtensible Application Markup Language), элементы управления, привязку данных, макеты, двухмерную и трёхмерную графику, анимацию, стили, шаблоны, документы, текст, мультимедиа и оформление. Что является преимуществом перед более ранней технологией создания пользовательских интерфейсов — Windows Forms.

Для работы с WPF использовался объектно-ориентированный язык программирования с С-подобным синтаксисом — С#, разработанный для создания приложений на платформе Microsoft .NET Framework [6].

Чтобы осуществлять связь между базой данных и приложением на C# необходим посредник. И именно таким посредником является технология ADO.NET [7]. ADO.NET – интерфейс прикладного уровня Microsoft (набор классов, предоставляющих службы доступа к данным). Является не развитием более ранней технологии ADO, а самостоятельной технологией, частью фреймворка .NET. Для обеспечения защиты паролей клиентов применяется технология хеширования.

Двоичный тип данных «Microsoft SQL Server» – это одна из тех функций, которую большинство разработчиков часто не используют, но оказывается, что данный тип является достаточно полезным для хранения мультимедийной информации. В проекте для хранения графической информации я использую тип данных varbinary(max).

## Проектирование

Для реализации поставленной задачи была создана база данных AfishaEvent.

Для базы данных было разработано 10 таблиц, которые связаны друг с другом внешними ключами.

Таблица USERS хранит данные о клиентах.

Таблица FILMS содержит подробную информацию о фильмах.

Таблица CONCERTS содержит подробную информацию о концертах.

Таблица CINEMAS содержит подробную информацию о кинотеатрах.

Таблица CONCERTHALLS содержит подробную информацию о концертных залах.

Таблица MOVIESINCINEMAS необходима для связи таблиц FILMS и CINEMAS.

Таблица CONCERTINCONCERTHALLS необходима для связи таблиц CONCERTS и CONCERTHALLS.

Таблица BOOKEDMOVIES хранит информацию о заказанных билетах на фильмы.

Таблица BOOKEDCONCERTS хранит информацию о заказанных билетах на концерты.

Таблица CALENDAR хранит даты, на которые запланированы события.

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

## Структура модели базы данных

Диаграмма связей таблиц для необходимой базы данных представлена на рисунке 3.2.1. Всего в рамках курсового проекта было разработано 10 таблиц.

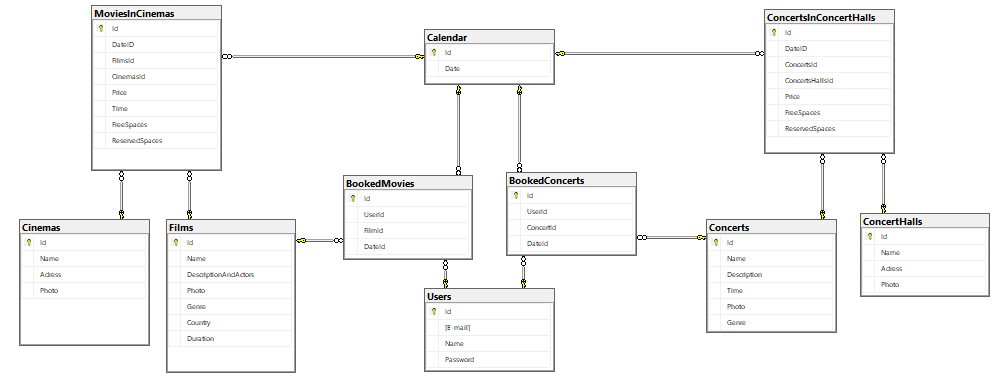


Рисунок 32.1 – Структура модели базы данных.

## Разработка необходимых объектов

## Таблицы

На рисунке 3.1.1 представлена структура таблицы Users. Таблица Users хранит в себе информацию о пользователях.

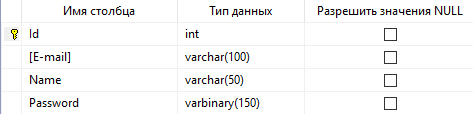


Рисунок 3.1.1 – Структура таблицы Users.

Описание всех полей приведено ниже:

* Id – идентификатора пользователя;
* E-mail – email пользователя;
* Name – имя пользователя;
* Password – зашифрованный пароль.

На рисунке 3.1.2 представлена структура таблицы Calendar. Таблица Calendar хранит информацию о датах, в которых содержаться мероприятия.

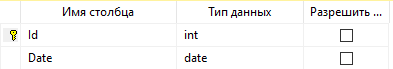


Рисунок 3.1.2 – Структура таблицы Calendar.

Описание всех полей приведено ниже:

* Id – идентификатор даты;
* Date – дата;

На рисунке 3.1.3 представлена структура таблицы Films. Таблица Films служит для хранения информации о фильмах.

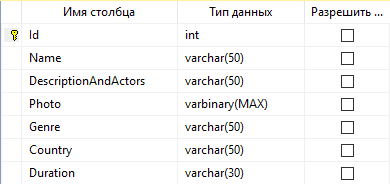


Рисунок 3.1.3 – Структура таблицы Films.

Описание всех полей приведено ниже:

* Id – идентификатор фильма;
* DescriptionAndActors – путь к файлу с описанием;
* Photo – картинка фильма;
* Genre – жанр фильма;
* Country – страна создания;
* Name – название фильма;
* Duration – продолжительность фильма в минутах.

На рисунке 3.1.4 представлена структура таблицы Cinemas. Таблица Cinemas служит для хранения информации о кинотеатрах.

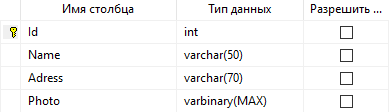


Рисунок 3.1.4 – Структура таблицы Cinemas.

Описание всех полей приведено ниже:

* Id – идентификатор кинотеатра;
* Name – название кинотеатра;
* Address – адрес кинотеатра;
* Photo – фотография кинотеатра.

На рисунке 3.1.5 представлена структура таблицы MoviesInCinemas. Таблица MoviesInCinemas служит для связи таблиц Films, Cinemas, Calendar и содержит информацию о сеансе.

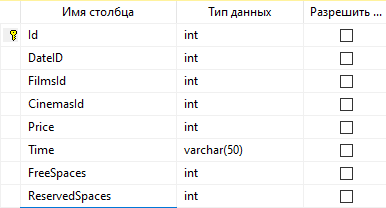


Рисунок 3.1.5 – Структура таблицы MoviesInCinemas.

Описание всех полей приведено ниже:

* Id – идентификатор;
* DateID – идентификатор даты;
* FilmsID – идентификатор фильма;
* CinemasID – идентификатор кинотеатра;
* Price – цена сеанса;
* Time – время сеанса;
* FreeSpaces – количество свободных мест;
* ReservedSpaces – количество забронированных мест.

На рисунке 3.1.6 представлена структура таблицы BookedMovies. Таблица BookedMovies служит для хранения информации о посещенных пользователями фильмах.

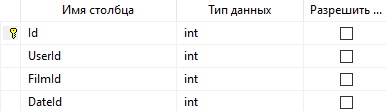


Рисунок 3.1.6 – Структура таблицы BookedMovies.

Описание всех полей приведено ниже:

* Id – идентификатор;
* UserId – идентификатор пользователя;
* FilmId – идентификатор фильма;
* DateId – идентификатор даты.

На рисунке 3.1.7 приведена структура таблицы Concerts. Таблица Concerts служит для хранения информации о концертах.

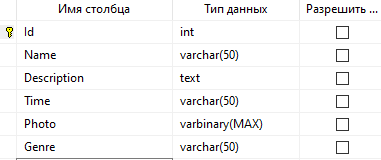


Рисунок 3.1.7 – Структура таблицы Concerts.

Описание всех полей приведено ниже:

* Id – идентификатор концерта;
* Name– название концерта;
* Description – путь к файлу с описанием;
* Time – время проведения;
* Photo – картинка концерта;
* Genre – жанр концерта.

На рисунке 3.1.8 представлена структура таблицы ConcertHall. Таблица ConcertHalls служит для хранения информации о концертных площадках.

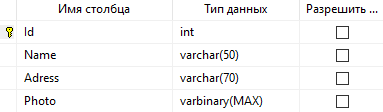


Рисунок 3.1.8 – Структура таблицы ConcertHalls.

Описание всех полей приведено ниже:

* Id – идентификатор концертной площадки;
* Name – название концертной площадки;
* Address – адрес концертной площадки;
* Photo – фотография концертной площадки.

На рисунке 3.1.9 представлена структура таблицы ConcertsInConcertHalls. Таблица ConcertsInConcertHalls служит для связи таблиц Concert, ConcertHalls, Calendar и содержит информацию о мероприятии.



Рисунок 3.1.9 – Структура таблицы ConcertsInConcertHalls.

Описание всех полей приведено ниже:

* Id – идентификатор;
* DateID – идентификатор даты;
* ConcertID – идентификатор концерта;
* ConcertHallsID – идентификатор концертной площадки;
* Price – цена концерта;
* FreeSpaces – количество свободных мест;
* ReservedSpaces – количество забронированных мест.

На рисунке 3.1.10 представлена структура таблицы BookedConcerts. Таблица BookedConcerts служит для хранения информации о посещенных пользователями концертах.

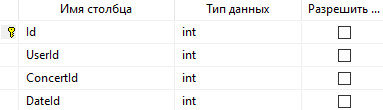


Рисунок 3.1.10 – Структура таблицы BookedConcerts.

Описание всех полей приведено ниже:

* Id – идентификатор;
* UserId – идентификатор пользователя;
* ConcertsId – идентификатор концерта;
* DateId – идентификатор даты.

Скрипты для создания всех таблиц базы данных представлены в Приложении А раздел таблицы.

## Пользователи

Рассмотрим возможности, которые предоставляет разработанное программное средство.

На рисунке 3.3.1 показана UML-диаграмма, отображающая возможности приложения с точки зрения неавторизованного пользователя.

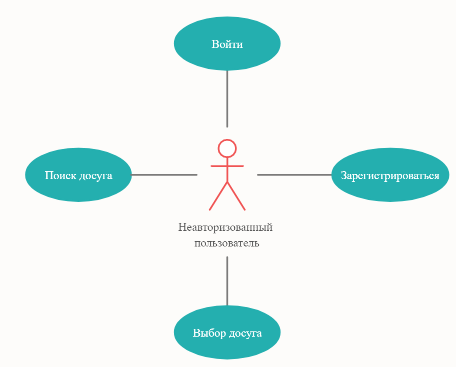


Рисунок 3.3.1 – UML диаграмма вариантов использования для неавторизованного пользователя

Неавторизованный пользователь имеет четыре варианта взаимодействия с системой: авторизоваться, зарегистрироваться, выбор досуга и поиск досуга.

Если пользователь авторизуется, то будет иметь возможности, представленные на рисунке 3.3.2.

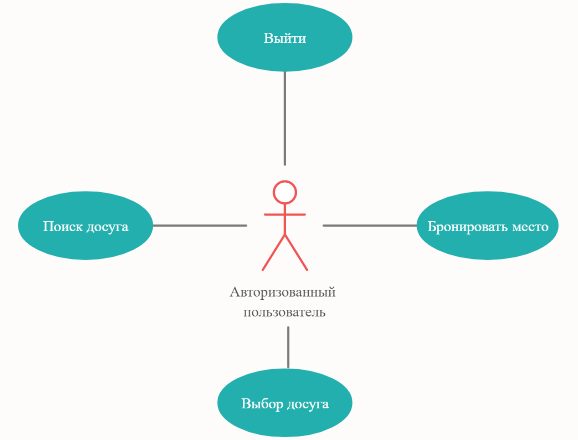


Рисунок 3.3.2 – UML диаграмма вариантов использования для авторизованного пользователя

Авторизованный пользователь может бронировать билеты, выбирать досуг, осуществлять поиск по досугам, а также может выйти из аккаунта.

Также в приложение можно войти как администратор. Тогда открываются следующие возможности: экспорт и импорт таблиц в XML, централизованное управление базой данных, выход из аккаунта. Диаграмма вариантов использования для администратора отображена на рисунке 3.3.3.

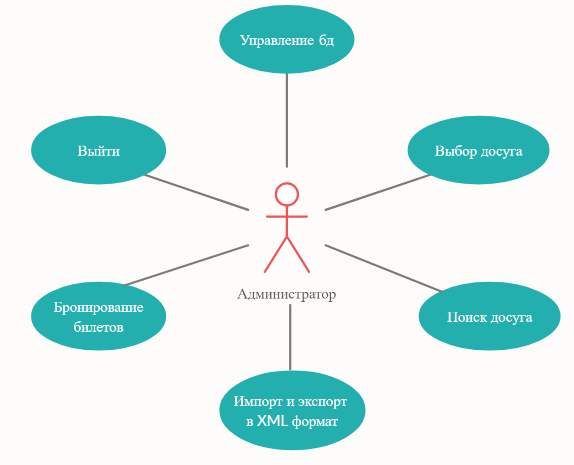


Рисунок 3.3.3 – UML диаграмма вариантов использования для администратора

## Функции

Использование SQL-функции позволяет выполнять набор операторов SQL и возвращают результат последнего запроса в списке. В простом случае (не с множеством) будет возвращена первая строка результата последнего запроса и ограничиваем доступ пользователей к прямому использованию таблиц базы данных.

## Индексы

Индекс – это объект базы данных, позволяющий ускоритьпоиск в определенной таблице, так как при этом данные организуются в виде сбалансированного бинарного дерева поиска.

Индекс, созданный в рамках курсового проекта приведён ниже:

* Индекс IX\_Users\_Name;

Определение индекса будет представлено в приложении А

## Процедуры

Использование хранимых процедур позволяет ограничить или 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. Также процедуры позволяют построить более сложную логику взаимодействия с пользователем, существенно расширяя функционал базы данных.

AddBC– процедура для добавления в БД заказанного билета на концерт;

AddBM– процедура для добавления в БД заказанного билета на фильм;

AddCICH– процедура для добавления концерта в концертный зал;

AddCinema– процедура для добавления фильма;

AddConcertHalls – процедура для добавления концертного зала;

AddConcerts– процедура для добавления концерта;

AddDate – процедура для добавления даты в календарь.

AddFilm – процедура для добавления фильма;

AddMIC – процедура для добавления фильма в кинотеатр;

AddUser – процедура для добавления нового пользователя;

Auth– процедура для авторизации;

BuyConcerts – процедура для покупки билета на концерт;

BuyFilms – процедура для покупки билета на фильм;

ChangeBC– процедура для изменения информации о заказанных билетах на концерт;

ChangeBM – процедура для изменения информации о заказанных билетах на фильм;

ChangeCICH – процедура для изменения информации о концертах в концертных залах;

ChangeCinema – процедура для изменения информации о кинотеатре;

ChangeConcertHalls – процедура для изменения информации о концертном зале;

ChangeConcerts – процедура для изменения информации о концертах;

ChangeFilm – процедура для изменения информации о фильме;

ChangeMIC – процедура для изменения информации о фильмах в кинотеатрах;

DeleteBCById – процедура для удаления информации о заказанных билетах на концерт;

DeleteBMById – процедура для удаления информации о заказанных билетах на фильм;

DeleteCalendarById– процедура для удаления информации из календаря;

DeleteCICHById– процедура для удаления информации о концертах в концертных залах;

DeleteCinemasById – процедура для удаления кинотеатра;

DeleteConcertHallsById – процедура для удаления концертного зала;

DeleteConcertsById – процедура для удаления концерта;

DeleteFilmsById – процедура для удаления фильма;

DeleteMICById – процедура для удаления информации о фильмах в кинотеатрах;

DeleteUserById – процедура для удаления информации о пользовтеле;

ExportToXML – процедура для экспорта данных в xml;

GetCinemasByFilmId – процедура для получения информации о кинотеатрах по id фильма;

GetConcertHallsByConcertId – процедура для получения информации о концертных залах по id концерта;

GetConcertsByConcertHallsId – процедура для получения информации о концертах по id концертного зала;

GetFilmsByCinemaId – процедура для получения инофрмации о фильмах по id кинотеатра;

ImportFromXML – процедура для вызова всех процедур импорта в xml;

ImportFromXmlBC – процедура для импорта в таблицу BookedConcerts;

ImportFromXmlBM – процедура для импорта в таблицу BookedMovies;

ImportFromXmlCalendar – процедура для импорта в таблицу Calendar;

ImportFromXmlCICH – процедура для импорта в таблицу ConcertsInConcertHalls;

ImportFromXmlCinemas – процедура для импорта в таблицу Cinemas;

ImportFromXmlConcerts – процедура для импорта в таблицу Concerts;

ImportFromXmlFilms – процедура для импорта в таблицу Films;

ImportFromXmlMIC – процедура для импорта в таблицу MoviesInCinemas;

SearchCinemas – процедура для поиска фильма по названию;

SearchConcertHalls – процедура для поиска концертного зала по названию;

SearchConcerts – процедура для поиска концерта по названию;

SearchFilms – процедура для поиска фильма по названию;

SelectAllBC – процедура для получения информации о заказных билетах на концерты

SelectAllBM – процедура для получения информации о заказанных билетах на фильмы;

SelectAllCICH – процедура для получения информации о концертах в концертных залах;

SelectAllCinemas – процедура для получения информации о всех кинотеатрах;

SelectAllCinemasByDate – процедура для получения информации о фильмах по дате;

SelectAllConcertHalls – процедура для получения информации о всех концертных залах;

SelectAllConcertHallsByDate – процедура для получения информации о концертных залах по дате;

SelectAllConcerts – процедура для получения информации о концертах

SelectAllConcertsByDate – процедура для получения информации о концертах по дате;

SelectAllDates– процедура для получения информации из календаря;

SelectAllFilms– процедура для получения информации о фильмах;

SelectAllFilmsByDate– процедура для получения информации о фильмах по дате;

SelectAllMIC– процедура для получения информации о фильмах в кинотеатрах;

SelectAllUsers– процедура для получения информации о всех пользовтелях;

Все скрипты хранимых процедур приложены в отдельном файле.

## Триггеры

Триггер — [хранимая процедура](https://ru.wikipedia.org/wiki/%D0%A5%D1%80%D0%B0%D0%BD%D0%B8%D0%BC%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80%D0%B0) особого типа, которую пользователь не вызывает непосредственно, а исполнение которой обусловлено действием по модификации данных: добавлением insert, удалением delete строки в заданной таблице, или изменением update данных в определённом столбце заданной таблицы [реляционной базы данных](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5_%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85).

Триггеры, созданные в рамках курсового проекта приведены ниже:

* Триггер BookedMovies\_after\_insert;
* Триггер BookedConcerts\_after\_insert;

Определение триггеров будет представлено в приложении А.

## Описание процедур импорта и экспорта данных

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 объекта в виде XML.

## Процедура импорта данных из XML-файла

Для импорта используется стандартная функция, входящая в набор OLE DB – OPENROWSET, в которую передаются параметры о типе импортируемых данных и пути файла, где они находятся. В данном курсовом проекте импорт данных из XML-файла осуществляется при вызове процедуры ImportFromXML. При вызове данной процедуры происходит вызов других процедур, отвечающих за импорт данных из XML-файла в базу данных.

Скрипт процедуры ImportFromXML представлен в Приложении А.

## Процедура экспорта данных в XML-файл

В данном курсовом проекте экспорт данных в XML-файл осуществляется при вызове процедуры ExportToXML. При нажатии на соответствующую кнопку в меню вызывается процедура ExportToXML, которая в свою очередь вызывает функцию, отвечающую за экспорт данных из базы данных в XML-файлы. Скрипт процедуры экспорта в XML-файл представлен в Приложении А.

## Описание технологии

Microsoft SQL Server поддерживает несколько типов данных для работы с объектами большого размера: binary и varbinary. Binary позволяет хранить двоичные данные фиксированной длины размером до 8000 байт. Varbinary, в свою очередь, позволяет хранить двоичные данные переменной длины до 8000 байт, однако при указании в параметрах значения max, объем хранимых данных возрастает до 2 Гигабайт. Если для обмена данными лучше всего подходит тип binary, то другие типы данных удобнее всего будет преобразовывать в binary и varbinary.

В определенный момент времени вы можете преобразовать тип значения в двоичное значение достаточно большого размера, а затем преобразовать его обратно. Это преобразование всегда возвращает одно и то же значение, если эти преобразования выполняются с использованием одной и той же версии SQL Server.

Так же SQL Server содержит в себе такой тип данных, как Image, однако он считается устаревшим, поэтому рекомендуют использовать varbinary.

В данной курсовой работе мультимедийные типы данных использовались в таблицах Films, Cinemas, Concerts, ConcertHalls. Для хранения использовался тип данных varbinary. Для добавления фотографии в базу данных нужно зайти под учетной записью администратора и открыть в настройках приложения Application Manager, представленный на рисунке 5.1.1.

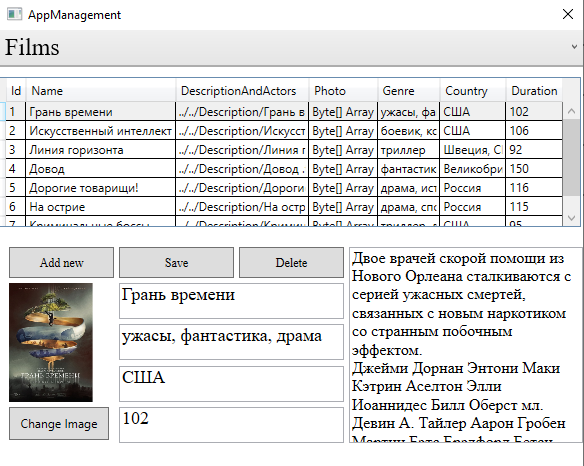


Рисунок 5.1.1 – Приложение для работы с базой данных.

## Тестирование

## Тестирование производительности базы данных.

Для тестирования производительности была взята за основу таблица USER, которую мы заполнили 100000 строками. После этого был применён SELECT-запрос к данной таблице на столбец Username и при помощи стандартных средств IDE MS SQL Server Management Studio оценена цена выборки к таблице. Результат данной оценки запроса приведён на рисунках 6.1.1, 6.1.2, 6.1.3

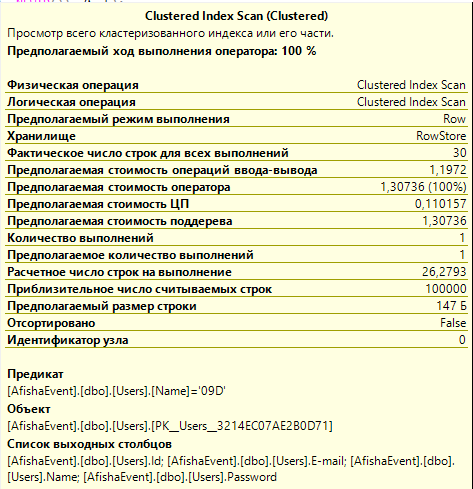


Рисунок 6.1.1 – Оценка запроса к таблице USER.



Рисунок 6.1.2 – Количество логических чтений к таблице USER.



Рисунок 6.1.3 – Время выполнения запроса к таблице.

После проведения первоначальной оценки был построен некластеризованный индекс к таблице USER, создание которого можно увидеть в приложении Б, и проведена оценка такого же SELECT-запроса к таблице USER. Результаты, полученные во время оценки, представлены на рисунках 6.1.4, 6.1.5, 6.1.6.

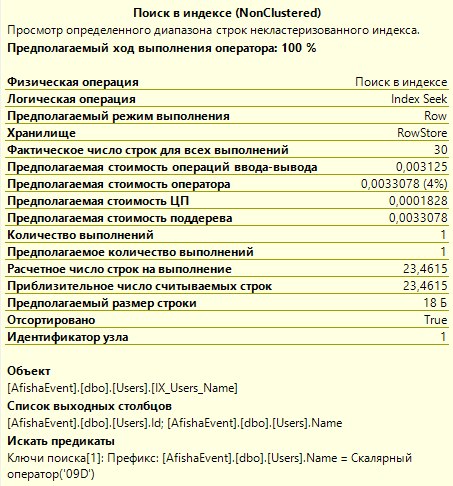


Рисунок 6.1.4 – Оценка запроса к таблице USER с построенным некластеризованным индексом.

C:\Users\konop\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Screenshot_51.png

Рисунок 6.1.5 – Количество логических чтений к таблице USER с USER с построенным некластеризованным индексом



Рисунок 6.1.6– Время выполнения запроса к таблице USER с построенным некластеризованным индексом.

По результатам проведённых оценок, можно сделать вывод, что после создания некластеризованного индекса:

– стоимость по параметру “Предполагаемая стоимость операций ввода-вывода” стала в 400 раз меньше;

– стоимость по параметру “Предполагаемая стоимость оператора” снизилась в 400 раз;

– стоимость по параметру “Предполагаемая стоимость ЦП” снизилась в 600 раз;

– стоимость по параметру “Предполагаемая стоимость поддерева” уменьшилась в 400 раз.

– стоимость по параметру “Предполагаемая стоимость поддерева” уменьшилась в 400 раз.

– количество логических чтений к таблице уменьшилось в 17 раз.

– время выполнения запроса уменьшилось с 16 мс до 0 мс.

Таким образом, постройка индекса к таблице была более чем оправдана, так как мы получили большой прирост производительности.

## Тестирование работоспособности приложения.

Приложение предназначено для трех ролей:

* Неавторизованный пользователь;
* Авторизованный пользователь;
* Администратор.

Приложение имеет две категории:

* Фильмы и кинотеатры;
* Концерты и концертные залы.

При первом открытии приложения, пользователь является неавторизованным пользователем и имеет возможность только просматривать фильмы, концерты, осуществлять поиск по этим разделам, войти в аккаунт или же зарегистрироваться (Рис 6.2.1). Неавторизованный пользователь не имеет возможность забронировать место на каком-либо мероприятии. Рис 6.2.2.

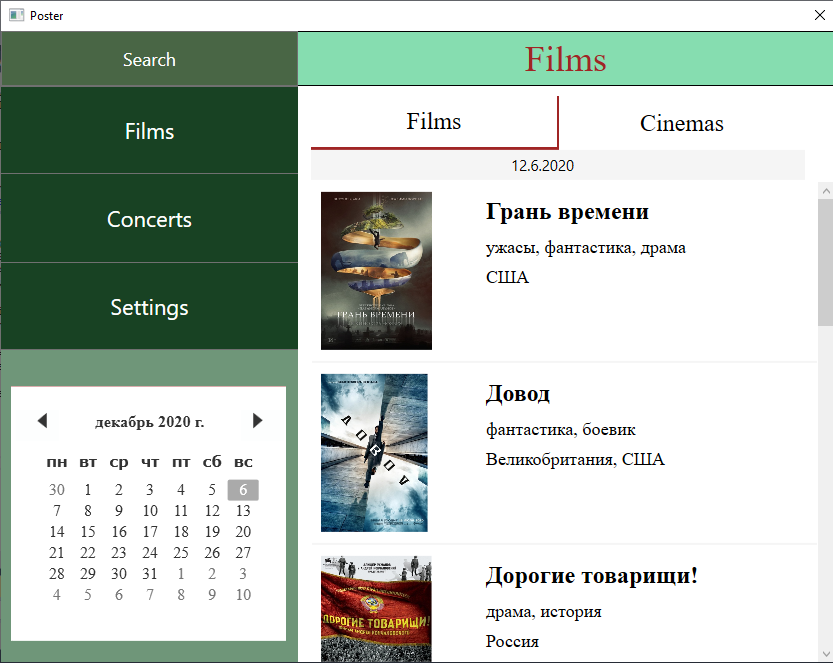


Рисунок 6.2.1 – Начальное окно при запуске приложения.

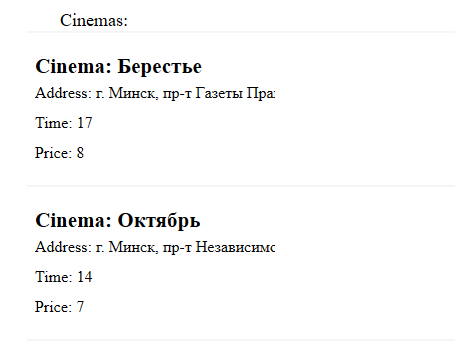


Рисунок 6.2.2 – Окно бронирования билетов для неавторизованного пользователя.

Для того чтобы неавторизованному пользователю зайти в свою учетную запись или же в аккаунт администратора ему и перейти в раздел Settings. Затем нажать кнопку Sign In, далее пользователь увидит окно входа, представленное на рисунке 6.2.3.



Рисунок 6.2.3 – Окно авторизации в приложении.

Если пользователь не имеет аккаунта в приложении, то он имеет возможность зарегистрироваться, нажав на кнопку Sign Up, откроется окно регистрации, как показано на рисунке 6.2.4.



Рисунок 6.2.4 – Окно регистрации в приложении.

При авторизации у пользователя появляется возможность бронировать билеты. При выборе какого-либо мероприятия пользователь может забронировать некоторое количество мест. Окно бронирования показано на рисунке 6.2.4

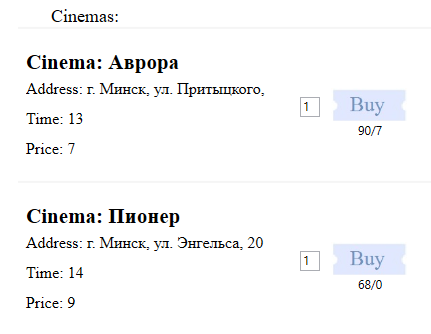


Рисунок 6.2.4 – Окно бронирования билетов.

Если пользователь авторизуется как администратор, то при переходе в раздел Settings, у него появится возможность экспорта и импорта базы данных в XML-файлы. Так же с помощью Application Manager администратор может управлять базой данных: выполнять операции редактирования данных в базе данных, удаления данных и добавления данных. Как показано на рисунке 6.2.5 и 6.2.6.

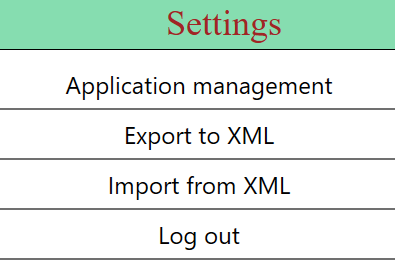


Рисунок 6.2.5 – Раздел Settings в учетной записи администратора.

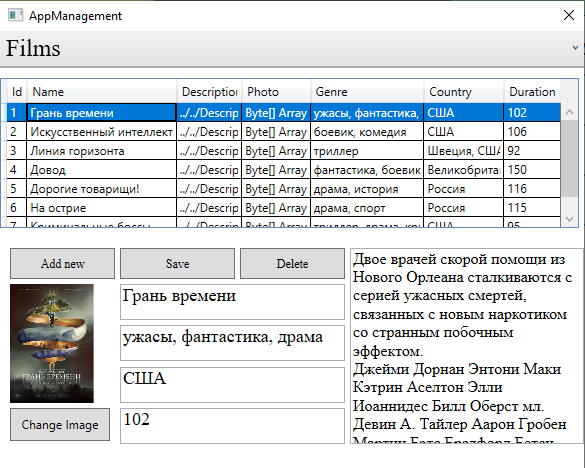


Рисунок 6.2.5 – Раздел Application management для редактирования данных.

# Заключение

В данном курсовом проекте была разработана база данных «AfishaEvent». Данная база данных состоит из 10 таблиц, 67 хранимых процедур, 23 функций и 1 индекса.

На платформе .NET Framework было разработано desktop приложение с основным интерфейсом для демонстрации возможных действий пользователей в базе данных.

Приложение прошло тестирование при использовании в БД большого количество данных.

В ходе выполнения курсового проекта были задействованы мультимедийные типы данных, хеширование паролей, поиск по базе данных. Имеет различный функционал в зависимости от учётной записи. Присутствует возможность создания новых пользователей. Авторизованный клиент имеет возможность найти и забронировать место на желаемое мероприятие. Администратор может выполнить экспорт и импорт данных в XML файл. Так же он может полностью редактировать таблицы базы данных из приложения.

Таким образом, разработанное программное средство удовлетворяет всем требованиям технического задания.

# Список литературы

1. Блинова Е.А. Курс лекций по базам данных / Е.А. Блинова
2. Microsoft Docs – https://docs.microsoft.com/ru-ru/sql/?view=sql-server-ver15.
3. METANIT SQL – https://metanit.com/sql/.
4. METANIT C# – https://metanit.com/sharp/.
5. MSDN C# – https://docs.microsoft.com/ru-ru/dotnet/csharp/.
6. Сайт Afisha TUT.BY – <https://afisha.tut.by/film/>.
7. Сайт Яндекс Афиша – https://afisha.yandex.ru/.

# Приложение А

Листинг скриптов создания таблиц

create database AfishaEvent

use AfishaEvent

create table Calendar(

Id int primary key,

[Date] date unique not null

)

create table Films(

Id int primary key,

[Name] varchar(50) unique not null,

[DescriptionAndActors] varchar(50) not null,

Photo varbinary(max) not null,

Genre varchar(50) not null,

Country varchar(50) not null,

Duration varchar(30) not null

)

create table Cinemas(

Id int primary key,

[Name] varchar(50) unique not null,

[Adress] varchar(70) not null,

Photo varbinary(max) not null,

)

create table MoviesInCinemas(

Id int primary key,

DateID int references Calendar(Id) on delete cascade not null,

FilmsId int references Films(Id) on delete cascade not null,

CinemasId int references Cinemas(id) on delete cascade not null,

Price int not null,

[Time] varchar(50) not null,

[FreeSpaces] int not null,

[ReservedSpaces] int not null

)

create table Users(

Id int primary key,

[E-mail] varchar(100) unique not null,

[Name] varchar(50) not null,

[Password] varbinary(150) not null

)

create table Concerts(

Id int primary key,

[Name] varchar(50) unique not null,

[Description] text not null,

[Time] varchar(50) not null,

Photo varbinary(max) not null,

Genre varchar(50) not null

)

create table ConcertHalls(

Id int primary key,

[Name] varchar(50) unique not null,

[Adress] varchar(70) not null,

Photo varbinary(max) not null

)

create table ConcertsInConcertHalls(

Id int primary key,

DateID int references Calendar(Id) on delete cascade not null,

ConcertsId int references Concerts(Id) on delete cascade not null,

ConcertsHallsId int references ConcertHalls(id) on delete cascade not null,

Price int not null,

[FreeSpaces] int not null,

[ReservedSpaces] int not null

)

create table BookedMovies(

Id int primary key,

UserId int references Users(Id) on delete cascade not null,

FilmId int references Films(Id) on delete cascade not null,

DateId int references Calendar(Id) not null

)

create table BookedConcerts(

Id int primary key,

UserId int references Users(Id) on delete cascade not null,

ConcertId int references Concerts(Id) on delete cascade not null,

DateId int references Calendar(Id) not null

)

insert into Users(Id,[E-mail],[Name],[Password])

values (1,'Admin','Admin',pwdencrypt('Pa$$word'))

Листинг функций

create function CheckIdCalendar()

returns int

as

begin

declare @count int

set @count = 0

while @count < 10000

begin

set @count = @count + 1

if(select count(Id) from Calendar where Id = @count) = 0

break;

end;

return @count

end;

create function GetConcertId(@newId int)

returns int

as

begin

declare @Id int

select @Id = ConcertsId from ConcertsInConcertHalls where Id = @newId

return @Id

end;

create function GetFilmId(@newId int)

returns int

as

begin

declare @Id int

select @Id = FilmsId from MoviesInCinemas where Id = @newId

return @Id

end;

create function SelectIdCinema(@newCinemaName varchar(50))

returns int

as

begin

declare @Id int

select @Id = Id from Cinemas where [Name] = @newCinemaName

return @Id

end;

Листинг процедур

CREATE PROCEDURE AddBM

@newId int,

@newUserMail varchar(50),

@newFilmsName varchar(50),

@newDate date

AS

BEGIN

insert into BookedMovies(Id,UserId,FilmId,DateId)

values(dbo.CheckIdBookedMovies(@newId), dbo.SelectIdUsers(@newUserMail), dbo.SelectIdFilms(@newFilmsName), dbo.SelectIdDate(@newDate))

END;

alter procedure AddCinema

@newId int,

@newName varchar(50),

@newAddress varchar(70),

@newPhoto varbinary(MAX)

as

begin

insert into Cinemas(Id, [Name], Adress, Photo)

values (dbo.CheckIdCinema(@newId), @newName, @newAddress, @newPhoto);

end;

create procedure AddFilm

@newId int,

@newName varchar(50),

@newDescriptionAndActors varchar(50),

@newPhoto varbinary(MAX),

@newGenre varchar(50),

@newCountry varchar(50),

@newDuration varchar(30)

as

begin

insert into Films(Id, [Name], DescriptionAndActors, Photo, Genre, Country, Duration)

values (dbo.CheckIdFilm(@newId), @newName, @newDescriptionAndActors, @newPhoto, @newGenre, @newCountry, @newDuration);

end;

create procedure AddDate

@newId int,

@newDate date

as

begin

insert into Calendar(Id, [Date]) values (dbo.CheckIdCalendar(), @newDate);

end;

create procedure AddMIC

@newId int,

@newDate date,

@newFilmsName varchar(50),

@newCinemasName varchar(50),

@newPrice int,

@newTime varchar(50),

@newReservedSpaces int,

@newFreeSpaces int as

begin

if(@newFreeSpaces > @newReservedSpaces)

begin

if not exists(select Id from Calendar where [Date] = @newDate)

insert into Calendar(Id, [Date]) values (dbo.CheckIdCalendar(), @newDate);

insert into MoviesInCinemas(Id,[DateID],[FilmsId],[CinemasId],[Price],[Time],[FreeSpaces],[ReservedSpaces])

values(dbo.CheckIdMIC(@newId), dbo.SelectIdDate(@newDate), dbo.SelectIdFilms(@newFilmsName), dbo.SelectIdCinema(@newCinemasName), @newPrice,@newTime, @newFreeSpaces, @newReservedSpaces)

end;

end;

create procedure AddUser

@newId int,

@newMail varchar(100),

@newName varchar(50),

@newPassword varchar(150)

as

begin

insert into Users(Id, [E-mail], [Name], [Password])

values (dbo.CheckIdUser(@newId), @newMail, @newName, pwdencrypt(@newPassword));

end;

create procedure ChangeUser

@Id int,

@newMail varchar(100),

@newName varchar(50),

@newPassword varchar(150)

as

begin

update Users

set [E-mail] = @newMail,

[Name] = @newName,

[Password] = pwdencrypt(@newPassword)

where [Id] = @Id

end;

create procedure auth

@Pass varchar(150),

@Mail varchar(100),

@out int output as

begin

declare @Userpass varbinary(150)

select @Userpass = Password from Users where [E-mail] = @Mail

set @out = pwdcompare(@Pass,@Userpass,0)

end;

create procedure BuyFilms

@Id int,

@count int,

@User nvarchar(50),

@Date date as

begin

declare @freeSpace int

declare @IdUpdate int

declare @IdUserUpdate int

declare @IdFilmUpdate int

declare @IdDateUpdate int

declare @totalfreeSpace int

declare @reservedSpaces int

select @freeSpace = FreeSpaces from MoviesInCinemas where Id = @Id

select @reservedSpaces = ReservedSpaces from MoviesInCinemas where Id = @Id

set @totalfreeSpace = @freeSpace - @reservedSpaces

--if(@count <= @totalfreeSpace)

begin

update MoviesInCinemas set ReservedSpaces = ReservedSpaces + @count where Id = @Id

set @IdUpdate = dbo.CheckIdBookedMovies(@id)

set @IdFilmUpdate = dbo.GetFilmId(@Id);

set @IdDateUpdate = dbo.SelectIdDate(@Date)

set @IdUserUpdate = dbo.SelectIdUsers(@User)

insert BookedMovies(Id,UserId,FilmId,DateId)

values (@IdUpdate, @IdUserUpdate, @IdFilmUpdate, @IdDateUpdate)

end;

end;

create procedure BuyConcerts

@Id int,

@count int,

@User nvarchar(50),

@Date date as

begin

declare @freeSpace int

declare @IdUpdate int

declare @IdUserUpdate int

declare @IdConcertUpdate int

declare @IdDateUpdate int

declare @totalfreeSpace int

declare @reservedSpaces int

select @freeSpace = FreeSpaces from MoviesInCinemas where Id = @Id

select @reservedSpaces = ReservedSpaces from MoviesInCinemas where Id = @Id

set @totalfreeSpace = @freeSpace - @reservedSpaces

--if(@count <= @totalfreeSpace)

begin

update ConcertsInConcertHalls set ReservedSpaces = ReservedSpaces + @count where Id = @Id

set @IdUpdate = dbo.CheckIdBookedConcerts(@id)

set @IdConcertUpdate = dbo.GetConcertId(@Id)

set @IdDateUpdate = dbo.SelectIdDate(@Date)

set @IdUserUpdate = dbo.SelectIdUsers(@User)

insert BookedConcerts(Id,UserId,ConcertId,DateId)

values (@IdUpdate, @IdUserUpdate, @IdConcertUpdate, @IdDateUpdate)

end;

end;

create procedure ChangeFilm

@Id int,

@newName varchar(50),

@newDescriptionAndActors varchar(50),

@newPhoto varbinary(MAX),

@newGenre varchar(50),

@newCountry varchar(50),

@newDuration varchar(30) as

begin

update Films

set

[Name] = @newName,

DescriptionAndActors = @newDescriptionAndActors,

Photo = @newPhoto,

Genre = @newGenre,

Country = @newCountry,

Duration = @newDuration

where [Id] = @Id

end;

create procedure ChangeCinema

@Id int,

@newName varchar(50),

@newAddress varchar(50),

@newPhoto varbinary(MAX)

as

begin

update Cinemas

set

[Name] = @newName,

Adress = @newAddress,

Photo = @newPhoto

where [Id] = @Id

end;

create procedure ChangeMIC

@Id int,

@newDate date,

@newFilmsName varchar(50),

@newCinemasName varchar(50),

@newPrice int,

@newTime varchar(50),

@newFreeSpaces int,

@newReservedSpaces int as

begin

update MoviesInCinemas

set [DateID] = dbo.SelectIdDate(@newDate),

FilmsId = dbo.SelectIdFilms(@newFilmsName),

CinemasId = dbo.SelectIdCinema(@newCinemasName),

Price = @newPrice,

[Time] = @newTime,

FreeSpaces = @newFreeSpaces,

ReservedSpaces = @newReservedSpaces

where [Id] = @Id

end;

create procedure DeleteConcertsById

@deleteId int as

begin

delete from Concerts where Id = @deleteId;

end

create PROCEDURE GetCinemasByFilmId -- GetAllCinemasByIdFilm

@FilmId int,

@currentDate date AS

BEGIN

select t3.Name, t3.Adress, t3.Photo, t1.Price, t1.Time, t1.ReservedSpaces, t1.FreeSpaces , t1.Id from MoviesInCinemas as t1

inner join Calendar as t2 on (t1.DateID = t2.Id)

inner join Cinemas as t3 on (t1.CinemasId = t3.Id)

inner join Films as t4 on (t1.FilmsId = t4.Id)

where(t2.Date = @currentDate and t4.Id = @FilmId)

END;

CREATE PROCEDURE GetConcertHallsByConcertId

@ConcertId int,

@currentDate date AS

BEGIN

select t3.Name, t3.Adress, t3.Photo, t1.Price, t1.ReservedSpaces, t1.FreeSpaces , t1.Id from ConcertsInConcertHalls as t1

inner join Calendar as t2 on (t1.DateID = t2.Id)

inner join ConcertHalls as t3 on (t1.ConcertsHallsId = t3.Id)

inner join Concerts as t4 on (t1.ConcertsId = t4.Id)

where(t2.Date = @currentDate and t4.Id = @ConcertId)

END;

create PROCEDURE GetConcertsByConcertHallsId

@ConcertHallsId int,

@currentDate date AS

BEGIN

select t3.[Name], t3.Genre, t3.[Time], t3.Photo, t1.Price, t1.ReservedSpaces, t1.FreeSpaces , t1.Id from ConcertsInConcertHalls as t1

inner join Calendar as t2 on (t1.DateID = t2.Id)

inner join Concerts as t3 on (t1.ConcertsId = t3.Id)

inner join ConcertHalls as t4 on (t1.ConcertsHallsId = t4.Id)

where(t2.[Date] = @currentDate and t4.Id = @ConcertHallsId)

END;

alter PROCEDURE GetFilmsByCinemaId

@CinemaId int,

@currentDate date AS

BEGIN

select t3.Name, t4.Genre, t3.Photo, t1.Price, t1.ReservedSpaces, t1.FreeSpaces, t1.Id, t4.Name from MoviesInCinemas as t1

inner join Calendar as t2 on (t1.DateID = t2.Id)

inner join Cinemas as t3 on (t1.CinemasId = t3.Id)

inner join Films as t4 on (t1.FilmsId = t4.Id)

where(t2.Date = @currentDate and t3.Id = @CinemaId)

END;

CREATE PROCEDURE SearchFilms

@searchString varchar(50)

AS

BEGIN

select \* from Films where [Name] = @searchString

END;

CREATE PROCEDURE SearchConcerts

@searchString varchar(50)

AS

BEGIN

select \* from Concerts where [Name] = @searchString

END;

create PROCEDURE SelectAllCinemas

AS

BEGIN

select \* from Cinemas;

END;

create PROCEDURE SelectAllConcerts

AS

BEGIN

select \* from Concerts;

END;

create PROCEDURE SelectAllMIC

AS

BEGIN

select distinct t1.Id, t2.Date, t4.Name, t3.Name, t1.Price, t1.Time, t1.FreeSpaces, t1.ReservedSpaces

from MoviesInCinemas as t1

inner join Calendar as t2 on (t1.DateID = t2.Id)

inner join Cinemas as t3 on (t1.CinemasId = t3.Id)

inner join Films as t4 on (t1.FilmsId = t4.Id)

END;

create PROCEDURE SelectAllCICH

AS

BEGIN

select distinct t1.Id, t2.Date, t4.Name, t3.Name, t1.Price, t1.FreeSpaces, t1.ReservedSpaces

from ConcertsInConcertHalls as t1

inner join Calendar as t2 on (t1.DateID = t2.Id)

inner join ConcertHalls as t3 on (t1.ConcertsHallsId = t3.Id)

inner join Concerts as t4 on (t1.ConcertsId = t4.Id)

END;

create PROCEDURE SelectAllBM

AS

BEGIN

select distinct t1.Id, t4.[E-mail], t3.Name, t2.Date

from BookedMovies as t1

inner join Calendar as t2 on (t1.DateId = t2.Id)

inner join Films as t3 on (t1.FilmId = t3.Id)

inner join Users as t4 on (t1.UserId = t4.Id)

END;

create PROCEDURE SelectAllBC

AS

BEGIN

select distinct t1.Id, t4.[E-mail], t3.Name, t2.Date

from BookedConcerts as t1

inner join Calendar as t2 on (t1.DateId = t2.Id)

inner join Concerts as t3 on (t1.ConcertId = t3.Id)

inner join Users as t4 on (t1.UserId = t4.Id)

END;

create PROCEDURE SelectAllDates

AS

BEGIN

select \* from Calendar;

END;

CREATE PROCEDURE SelectAllUsers

AS

BEGIN

select \* from Users;

END;

Листинг триггеров

CREATE TRIGGER BookedMovies\_after\_insert

ON BookedMovies

AFTER INSERT

AS

IF EXISTS (SELECT id FROM MoviesInCinemas where FreeSpaces < ReservedSpaces)

begin

update MoviesInCinemas set ReservedSpaces = FreeSpaces where Id = (SELECT id FROM MoviesInCinemas where FreeSpaces < ReservedSpaces)

end;

CREATE TRIGGER BookedConcerts\_after\_insert

ON BookedConcerts

AFTER INSERT

AS

IF EXISTS (SELECT id FROM ConcertsInConcertHalls where FreeSpaces < ReservedSpaces)

begin

update ConcertsInConcertHalls set ReservedSpaces = FreeSpaces where Id = (SELECT id FROM ConcertsInConcertHalls where FreeSpaces < ReservedSpaces)

end;

Листинг индекса

create index IX\_Users\_Name on Users (Name);

Экспорт XML

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

RECONFIGURE;

exec master.dbo.sp\_configure 'xp\_cmdshell', 1;

RECONFIGURE;

create function ExportToXMLTable(@path nvarchar(500), @tableName nvarchar(100))

returns int as

begin

declare @fullTableName nvarchar(100) = 'AfishaEvent.dbo.' + @tableName;

declare @sql nvarchar(500) = 'BCP "SELECT \* FROM '+@fullTableName+' FOR XML PATH(''Order''), ROOT(''Root'')" QUERYOUT '+@path+' -r -t -T -w -S .\SQLEXPRESS';

EXEC xp\_cmdshell @sql;

return 1;

end;

create procedure ExportToXML

@path nvarchar(500)

as

begin

declare @films int

declare @cinemas int

declare @moviesInCinemas int

declare @concerts int

declare @concertHalls int

declare @concertInConcertHall int

declare @bookedMovies int

declare @bookedConcerts int

declare @users int

declare @calendar int

set @films = dbo.ExportToXMLTable(@path + 'Film.xml', 'Films');

set @cinemas = dbo.ExportToXMLTable(@path + 'Cinemas.xml', 'Cinemas');

set @moviesInCinemas = dbo.ExportToXMLTable(@path + 'MoviesInCinemas.xml', 'MoviesInCinemas');

set @concerts = dbo.ExportToXMLTable(@path + 'Concerts.xml', 'Concerts');

set @concertHalls = dbo.ExportToXMLTable(@path + 'ConcertHalls.xml', 'ConcertHalls');

set @concertInConcertHall = dbo.ExportToXMLTable(@path + 'ConcertsInConcertHalls.xml', 'ConcertsInConcertHalls');

set @bookedMovies = dbo.ExportToXMLTable(@path + 'BookedMovies.xml', 'BookedMovies');

set @bookedConcerts = dbo.ExportToXMLTable(@path + 'BookedConcerts.xml', 'BookedConcerts');

set @users = dbo.ExportToXMLTable(@path + 'Users.xml', 'Users');

set @calendar = dbo.ExportToXMLTable(@path + 'Calendar.xml', 'Calendar');

end;

exec dbo.ExportToXML 'D:\XMLAfishaEvent\';

Импорт XML

use Poster

create procedure ImportFromXmlFilms

@path nvarchar(500) as

begin

SET XACT\_ABORT ON

BEGIN TRAN

declare @results table (x xml)

declare @sql nvarchar(300)='SELECT CAST(x AS XML) FROM OPENROWSET(BULK '''+@path+''', SINGLE\_BLOB) AS T(x)';

INSERT INTO @results EXEC (@sql)

declare @xml XML = (SELECT TOP 1 x from @results);

insert into [Films]([Id],Name,DescriptionAndActors,Photo,Genre, Country, Duration)

SELECT

P.value('Id[1]', 'int') AS Id,

P.value('Name[1]', 'varchar(50)') AS [Name],

P.value('DescriptionAndActors[1]', 'varchar(50)') AS DescriptionAndActors,

P.value('Photo[1]', 'varbinary(max)') AS Photo,

P.value('Genre[1]', 'varchar(50)') AS Genre,

P.value('Country[1]', 'varchar(50)') AS Country,

P.value('Duration[1]', 'varchar(30)') AS Duration

FROM @xml.nodes('Root/Order') AS T3(P)

COMMIT;

end;

go

create procedure ImportFromXmlCinemas

@path nvarchar(500) as

begin

SET XACT\_ABORT ON

BEGIN TRAN

declare @results table (x xml)

declare @sql nvarchar(300)='SELECT CAST(x AS XML) FROM OPENROWSET(BULK '''+@path+''', SINGLE\_BLOB) AS T(x)';

INSERT INTO @results EXEC (@sql)

declare @xml XML = (SELECT TOP 1 x from @results);

insert into [Cinemas]([Id],[Name],[Adress],[Photo])

SELECT

P.value('Id[1]', 'int') AS Id,

P.value('Name[1]', 'varchar(50)') AS [Name],

P.value('Adress[1]', 'varchar(70)') AS [Adress],

P.value('Photo[1]', 'varbinary(max)') AS Photo

FROM @xml.nodes('Root/Order') AS T3(P)

COMMIT;

end;

go

alter procedure ImportFromXmlMIC

@path nvarchar(500) as

begin

SET XACT\_ABORT ON

BEGIN TRAN

declare @results table (x xml)

declare @sql nvarchar(300)='SELECT CAST(x AS XML) FROM OPENROWSET(BULK '''+@path+''', SINGLE\_BLOB) AS T(x)';

INSERT INTO @results EXEC (@sql)

declare @xml XML = (SELECT TOP 1 x from @results);

insert into MoviesInCinemas([Id],DateID,FilmsId,CinemasId,Price,[Time],[FreeSpaces],[ReservedSpaces])

SELECT

P.value('Id[1]', 'int') AS Id,

P.value('DateID[1]', 'int') AS DateID,

P.value('FilmsId[1]', 'int') AS FilmsId,

P.value('CinemasId[1]', 'int') AS CinemasId,

P.value('Price[1]', 'int') AS Price,

P.value('Time[1]', 'varchar(50)') AS [Time],

P.value('FreeSpaces[1]', 'int') AS [FreeSpaces],

P.value('ReservedSpaces[1]', 'int') AS [ReservedSpaces]

FROM @xml.nodes('Root/Order') AS T3(P)

COMMIT;

end;

go

create procedure ImportFromXmlConcerts

@path nvarchar(500) as

begin

SET XACT\_ABORT ON

BEGIN TRAN

declare @results table (x xml)

declare @sql nvarchar(300)='SELECT CAST(x AS XML) FROM OPENROWSET(BULK '''+@path+''', SINGLE\_BLOB) AS T(x)';

INSERT INTO @results EXEC (@sql)

declare @xml XML = (SELECT TOP 1 x from @results);

insert into Concerts([Id],[Name],[Description],[Time],Photo,Genre)

SELECT

P.value('Id[1]', 'int') AS Id,

P.value('Name[1]', 'varchar(50)') AS [Name],

P.value('Description[1]', 'varchar(70)') AS [Description],

P.value('Time[1]', 'varchar(50)') AS [Time],

P.value('Photo[1]', 'varbinary(max)') AS Photo,

P.value('Genre[1]', 'varchar(50)') AS Genre

FROM @xml.nodes('Root/Order') AS T3(P)

COMMIT;

end;

create procedure ImportFromXmlConcertHalls

@path nvarchar(500) as

begin

SET XACT\_ABORT ON

BEGIN TRAN

declare @results table (x xml)

declare @sql nvarchar(300)='SELECT CAST(x AS XML) FROM OPENROWSET(BULK '''+@path+''', SINGLE\_BLOB) AS T(x)';

INSERT INTO @results EXEC (@sql)

declare @xml XML = (SELECT TOP 1 x from @results);

insert into ConcertHalls([Id],[Name],[Adress],[Photo])

SELECT

P.value('Id[1]', 'int') AS Id,

P.value('Name[1]', 'varchar(50)') AS [Name],

P.value('Adress[1]', 'varchar(70)') AS [Adress],

P.value('Photo[1]', 'varbinary(max)') AS Photo

FROM @xml.nodes('Root/Order') AS T3(P)

COMMIT;

end;

go

create procedure ImportFromXmlCICH

@path nvarchar(500) as

begin

SET XACT\_ABORT ON

BEGIN TRAN

declare @results table (x xml)

declare @sql nvarchar(300)='SELECT CAST(x AS XML) FROM OPENROWSET(BULK '''+@path+''', SINGLE\_BLOB) AS T(x)';

INSERT INTO @results EXEC (@sql)

declare @xml XML = (SELECT TOP 1 x from @results);

insert into ConcertsInConcertHalls([Id],DateID,ConcertsId,ConcertsHallsId,Price,[FreeSpaces],[ReservedSpaces])

SELECT

P.value('Id[1]', 'int') AS Id,

P.value('DateID[1]', 'int') AS DateID,

P.value('ConcertsId[1]', 'int') AS ConcertsId,

P.value('ConcertsHallsId[1]', 'int') AS ConcertsHallsId,

P.value('Price[1]', 'int') AS Price,

P.value('FreeSpaces[1]', 'int') AS [FreeSpaces],

P.value('ReservedSpaces[1]', 'int') AS [ReservedSpaces]

FROM @xml.nodes('Root/Order') AS T3(P)

COMMIT;

end;

go

create procedure ImportFromXmlBM

@path nvarchar(500) as

begin

SET XACT\_ABORT ON

BEGIN TRAN

declare @results table (x xml)

declare @sql nvarchar(300)='SELECT CAST(x AS XML) FROM OPENROWSET(BULK '''+@path+''', SINGLE\_BLOB) AS T(x)';

INSERT INTO @results EXEC (@sql)

declare @xml XML = (SELECT TOP 1 x from @results);

insert into BookedMovies([Id],UserId,FilmId,DateId)

SELECT

P.value('Id[1]', 'int') AS Id,

P.value('UserId[1]', 'int') AS UserId,

P.value('FilmId[1]', 'int') AS FilmId,

P.value('DateId[1]', 'int') AS DateId

FROM @xml.nodes('Root/Order') AS T3(P)

COMMIT;

end;

create procedure ImportFromXmlBC

@path nvarchar(500) as

begin

SET XACT\_ABORT ON

BEGIN TRAN

declare @results table (x xml)

declare @sql nvarchar(300)='SELECT CAST(x AS XML) FROM OPENROWSET(BULK '''+@path+''', SINGLE\_BLOB) AS T(x)';

INSERT INTO @results EXEC (@sql)

declare @xml XML = (SELECT TOP 1 x from @results);

insert into BookedConcerts([Id],UserId,ConcertId,DateId)

SELECT

P.value('Id[1]', 'int') AS Id,

P.value('UserId[1]', 'int') AS UserId,

P.value('ConcertId[1]', 'int') AS ConcertId,

P.value('DateId[1]', 'int') AS DateId

FROM @xml.nodes('Root/Order') AS T3(P)

COMMIT;

end;

go

create procedure ImportFromXmlCalendar

@path nvarchar(500) as

begin

SET XACT\_ABORT ON

BEGIN TRAN

declare @results table (x xml)

declare @sql nvarchar(300)='SELECT CAST(x AS XML) FROM OPENROWSET(BULK '''+@path+''', SINGLE\_BLOB) AS T(x)';

INSERT INTO @results EXEC (@sql)

declare @xml XML = (SELECT TOP 1 x from @results);

insert into Calendar([Id],[Date])

SELECT

P.value('Id[1]', 'int') AS Id,

P.value('Date[1]', 'date') AS [Date]

FROM @xml.nodes('Root/Order') AS T3(P)

COMMIT;

end;

exec dbo.ImportFromXmlCalendar 'D:\Calendar.xml';

create procedure ImportFromXML

as

begin

exec dbo.ImportFromXmlFilms 'D:\XMLAfishaEventImport\Film.xml'

exec dbo.ImportFromXmlCinemas 'D:XMLAfishaEventImport\Cinemas.xml'

exec dbo.ImportFromXmlCalendar 'D:\XMLAfishaEventImport\Calendar.xml'

exec dbo.ImportFromXmlMIC 'D:\XMLAfishaEventImport\MoviesInCinemas.xml'

exec dbo.ImportFromXmlConcerts 'D:\XMLAfishaEventImport\Concerts.xml'

exec dbo.ImportFromXmlConcertHalls 'D:\XMLAfishaEventImport\ConcertHalls.xml'

exec dbo.ImportFromXmlCICH 'D:\XMLAfishaEventImport\ConcertsInConcertHalls.xml'

exec dbo.ImportFromXmlBM 'D:\XMLAfishaEventImport\BookedMovies.xml'

exec dbo.ImportFromXmlBC 'D:\XMLAfishaEventImport\BookedConcerts.xml'

end;

exec dbo.ImportFromXML;