

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий  
Кафедра Информационных систем и технологий  
Специальность 1-40 01 01 «Программное обеспечение информационных технологий»  
Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  
К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

Применение технологии полнотекстового поиска в базе данных MS SQL  
Server для задачи “База данных сети кинотеатров”

Выполнил студент Иодковская Вероника Яновна  
(Ф.И.О.)  
Руководитель проекта асс. Жигаровская С.А.  
(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.)  
Консультанты асс. Жигаровская С.А.  
(учен. степень, звание, должность, подпись, Ф.И.О.)  
Нормоконтролер асс. Жигаровская С.А.  
(учен. степень, звание, должность, подпись, Ф.И.О.)  
Курсовой проект защищен с оценкой \_\_\_\_\_

Минск 2018

## Оглавление

Введение.....	3
1. Постановка задачи.....	4
2. Разработка модели базы данных.....	5
3. Разработка необходимых объектов .....	7
3.1. Таблицы.....	7
3.2. Пользователи .....	7
3.3. Процедуры .....	7
4. Описание процедур импорта и экспорта данных .....	8
4.1. Процедура импорта данных из XML-файла .....	8
4.2. Процедура экспорта данных в XML-файл .....	8
5. Технология полнотекстового поиска .....	9
6. Тестирование .....	12
6.1 Тестирование производительности базы данных.....	12
6.2 Тестирование приложения .....	13
7. Руководство пользователя.....	15
7.1. Руководство пользователя администратора.....	15
7.2. Руководство пользователя .....	16
Заключение .....	21
Список использованной литературы .....	22
Приложение А .....	23
Приложение Б.....	25
Приложение В .....	26
Приложение Г .....	29
Приложение Д .....	30
Приложение Е.....	31

## **Введение**

Система управления базами данных (СУБД) – совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

Основные функции СУБД:

- определение структуры создаваемой базы данных, ее инициализация и проведение начальной загрузки;
- предоставление пользователям возможности манипулирования данными (выборка необходимых данных, выполнение вычислений, разработка интерфейса ввода/вывода, визуализация);
- обеспечение логической и физической независимости данных;
- защита логической целостности базы данных;
- защита физической целостности;
- управление полномочиями пользователей на доступ к базе данных;
- синхронизация работы нескольких пользователей;
- управление ресурсами среды хранения;
- поддержка деятельности системного персонала.
- Обычно современная СУБД содержит следующие компоненты:
  - ядро, которое отвечает за управление данными во внешней и оперативной памяти и журнализацию;
  - процессор языка базы данных, обеспечивающий оптимизацию запросов на извлечение и изменение данных и создание, как правило, машинно-независимого исполняемого внутреннего кода;
  - подсистему поддержки времени исполнения, которая интерпретирует программы манипуляции данными, создающие пользовательский интерфейс с СУБД.

Современная СУБД содержит в своем составе программные средства создания баз данных, средства работы с данными и сервисные средства. С помощью средств создания БД проектировщик, используя язык описания данных (ЯОД), переводит логическую модель БД в физическую структуру, а на языке манипуляции данными (ЯМД) разрабатывает программы, реализующие основные операции с данными (в реляционных БД – это реляционные операции). При проектировании привлекаются визуальные средства, т.е. объекты, и программа-отладчик, с помощью которой соединяются и тестируются отдельные блоки разработанной программы управления конкретной БД.

СУБД существует огромное множество: Oracle, MS SQL Server, Microsoft Access, MySql и так далее. В данной работе будет использовано решение от компании MS SQL Server.

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

## **1. Постановка задачи**

Спроектировать базу данных, построить программу, обеспечивающую взаимодействие с ней в режиме диалога, для службы кинотеатров.

В БД должны храниться сведения о кинотеатрах: название, район города, где расположен кинотеатр, количество залов, вместимость залов; о фильмах: название, релиз, страна, режиссер, жанр, актёр, постер, синопсис; кроме того, должна храниться информация о цене билета, количестве свободных мест на данный сеанс. На разных сеансах в одном кинотеатре могут идти разные фильмы. Кинотеатр может ввести новый фильм в репертуар или снять старый с проката.

Цена билета определяется прокатной стоимостью (названием) фильма и сеансом.

Справочной службе могут потребоваться следующие сведения о текущем состоянии проката фильмов в городе:

- репертуар кинотеатра (по названию кинотеатра);
- адрес и район кинотеатра (по названию кинотеатра);
- число мест (свободных) на данный сеанс (название кинотеатра и сеанс);
- цена билетов на данный сеанс (название кинотеатра и сеанс);
- жанр, страна, релиз, актёр и режиссер данного фильма (по названию);
- вместимость зала в кинотеатре (по названию зала).

Администратор БД может вносить следующие изменения:

- введение нового фильма в репертуар;
- снятие фильма с проката;
- введение информации о новом кинотеатре;
- обновление информации о кинотеатре и фильмах;
- удаление кинотеатра из базы;
- добавление новых сеансов.

Необходимо предусмотреть возможность выдачи справки о сеансах фильма в указанном кинотеатре (названия фильмов, в каких кинотеатрах они демонстрировались, цена билета в каждом кинотеатре на них).

Проанализировав задание, внесем некоторые коррективы:

– Условимся, что вместимость кинотеатра определяется как суммарная вместимость всех работающих залов кинотеатра.

– Цена билета определяется прокатной стоимостью фильма и временем сеанса.

– Отчет о репертуаре кинотеатра выдается за один день, дата которого выбирается пользователем.

## 2. Разработка модели базы данных

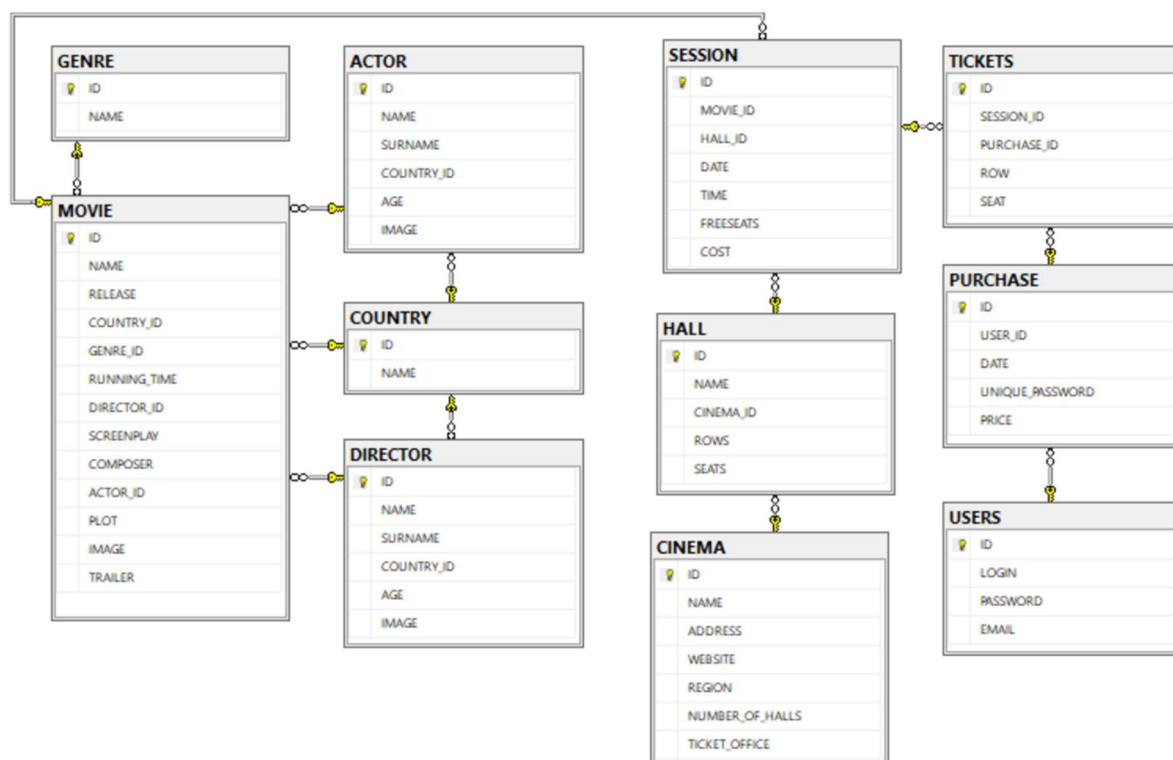


Рисунок 2.1 – Диаграмма базы данных

В реляционных базах данных и плоских файлах баз данных, таблица — это набор элементов данных (значений), использующий модель вертикальных столбцов (имеющих уникальное имя) и горизонтальных строк. Ячейка — место, где строка и столбец пересекаются. Таблица содержит определенное число столбцов, но может иметь любое количество строк. Каждая строка однозначно определяется одним или несколькими уникальными значениями, которые принимают её ячейки из определенного подмножества столбцов. Подмножество столбцов, которое уникально идентифицирует строку, называется первичным ключом.

В каждой таблице БД может существовать первичный ключ. Под первичным ключом понимают поле или набор полей, однозначно (уникально) идентифицирующих запись. Первичный ключ должен быть минимально достаточным: в нем не должно быть полей, удаление которых из первичного ключа не отразится на его уникальности.

«Таблица» — это ещё один термин для «отношения»; разница между ними в том, что таблица обычно представляет собой мультимножество (набор) строк, тогда как отношение представляет собой множество и не допускает дубликатов. Помимо обычных данных, таблицы, как правило, имеют связанные с ними метаданные, такие как ограничения, относящиеся к таблицам в целом или к значениям в определенных столбцах.

Для базы данных кинотеатра были разработаны 11 таблиц. Диаграмма связей таблиц для необходимой базы данных представлена на рисунке 1.1.

Таблица MOVIE, содержащая фильмы:

- ID - уникальный идентификатор;
- NAME – название фильма;
- RELEASE - релиз;
- COUNTRY\_ID – идентификатор страны;
- GENRE\_ID – идентификатор жанра;
- RUNNING\_TIME – длительность;
- DIRECTOR\_ID – идентификатор режиссёра»
- SCREENPLAY – сценарист;
- COMPOSER – композитор;
- ACTOR\_ID – идентификатор актёра;
- PLOT – синопсис;
- IMAGE – постер.

Таблица GENRE, содержащая жанры:

- ID – уникальный идентификатор;
- NAME – название жанра.

Таблица COUNTRY, содержащая страны:

- ID – уникальный идентификатор;
- NAME – название страны.

Таблица ACTOR, предназначенная для хранения информации об актёрах:

- ID – уникальный идентификатор;
- NAME – имя актёра;
- SURNAME – фамилия актёра;
- COUNTRY\_ID – идентификатор страны;
- AGE – возраст актёра;
- IMAGE – изображение актёра.

Таблица DIRECTOR, предназначенная для хранения информации о режиссёрах:

- ID – уникальный идентификатор;
- NAME – имя режиссёра;
- SURNAME – фамилия режиссёра;
- COUNTRY\_ID – идентификатор страны;
- AGE – возраст режиссёра;
- IMAGE – изображение режиссёра.

Таблица SESSION, предназначенная для хранения информации о сеансах:

- ID – уникальный идентификатор;
- MOVIE\_ID – идентификатор фильма;
- HALL\_ID – идентификатор зала;
- DATE – дата сеанса;
- TIME – время сеанса;
- FREESEATS – количество свободных мест.

- COST – цена билета на сеанс.

Таблица HALL, предназначенная для хранения информации о залах:

- ID – уникальный идентификатор;
- NAME – название зала;
- CINEMA\_ID – идентификатор кинотеатра;
- ROWS – количество рядов;
- SEATS – количество мест в ряду.

Таблица CINEMA, предназначена для хранения информации о кинотеатрах:

- ID – уникальный идентификатор;
- NAME – название кинотеатра;
- ADDRESS – адрес кинотеатра;
- WEBSITE – сайт кинотеатра;
- REGION – район города;
- NUMBER\_OF\_HALLS – количество залов;
- TICKET\_OFFICE – время работы билетных касс.

Таблица TICKETS, предназначена для хранения информации о билетах:

- ID – уникальный идентификатор;
- SESSION\_ID – идентификатор сеанса;
- PURCHASE\_ID – идентификатор покупки;
- ROW – ряд;
- SEAT – место.

Таблица PURCHASE, предназначена для хранения информации о покупках:

- ID – уникальный идентификатор;
- USER\_ID – идентификатор пользователя;
- DATE – дата покупки;
- UNIQUE\_PASSWORD – уникальный пароль покупки;
- PRICE – стоимость покупки.

Таблица USERS, предназначена для хранения данных о пользователях:

- ID – уникальный идентификатор;
- LOGIN – логин;
- PASSWORD – пароль;
- EMAIL – электронная почта.

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

### **3. Разработка необходимых объектов**

#### **3.1. Таблицы**

Таблицы являются основой любой базы данных, именно в них хранится вся информация. При проектировании базы данных было создано 11 таблиц, которые подробно описаны ранее в разделе 1, а SQL-скрипты для их создания находятся в Приложении А.

#### **3.2. Пользователи**

Пользователь базы данных – это физическое или юридическое лицо, которое имеет доступ к БД и пользуется услугами информационной системы для получения информации.

Пользователи базы данных получают права доступа для чтения, вставки, обновления и удаления конкретных объектов, которые задают набор полей и бизнес-правил. Эти объекты могут также обновлять одну или несколько таблиц базы данных.

Администратор кинотеатра был наделён привилегией на выполнение всех хранимых процедур, разработанных для данной базы данных.

Клиенту разрешено выполнять только 2 процедуры для авторизации и регистрации, 5 процедур, связанных с получением информации для покупки и 4 процедуры, необходимых для оформления заказа.

Скрипты для создания логинов и пользователей базы данных, а также выделение привилегий на осуществление определённых операций с базой данных представлены в приложении Б.

#### **3.3 Процедуры**

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

Хранимые процедуры могут возвращать множества результатов, то есть результаты запроса SELECT. Такие множества результатов могут обрабатываться, используя курсоры, другими сохранёнными процедурами, возвращая указатель результирующего множества, либо же приложениями. Хранимые процедуры могут также содержать объявленные переменные для обработки данных и курсоров, которые позволяют организовать цикл по нескольким строкам в таблице. Стандарт SQL предоставляет для работы выражения IF, LOOP, REPEAT, CASE и многие другие. Хранимые процедуры могут принимать переменные, возвращать результаты или изменять переменные и возвращать их, в зависимости от того, где переменная объявлена.

Хранимые процедуры обычно создаются с помощью языка SQL и конкретной его реализации в выбранной СУБД.



Листинги некоторых хранимых процедур представлены в приложении В.

Всего было разработано 24 процедуры:

- Authorisation – процедура для авторизации пользователя;
- Registration – процедура для регистрации пользователя;
- DeleteCinema – процедура для удаления кинотеатр;
- DeleteMovie – процедура для удаления фильма;
- ExportToXML – процедура для экспорта кинотеатра в файл \*.xml;
- FullTextSearch – процедура для полнотекстового поиска;
- GetActorInfo – процедура для вывода информации об актёрах;
- GetCinemaInfo – для вывода информации о кинотеатрах;
- GetMovieInfo – процедура для вывода информации о фильмах;
- GetSessionInfo – процедура для вывода информации о сеансах;
- ImportFromXML – процедура для импорта кинотеатров из файла \*.xml;
- InsertActor – процедура для добавления информации об актёрах;
- InsertCinema – процедура для добавления информации о фильмах;
- InsertDirector – процедура для добавления информации о режиссёрах;
- InsertGenre – процедура для добавления информации о жанрах;
- InsertHall – процедура для добавления информации о залах;
- InsertMovie – процедура для добавления информации о фильмах;
- InsertPurchase – процедура для покупки билетов;
- InsertSession – процедура для добавления информации о сеансах;
- InsertTicket – процедура для добавления информации о купленных билетах;
- SelectFreeSeats – процедура для вывода информации о свободных местах;
- SelectPrice – процедура для вывода информации о стоимости заказа;
- UpdateCinema – процедура для обновления информации о кинотеатрах;
- UpdateMovie – процедура для обновления информации о фильмах.

Все скрипты хранимых процедур приложены в отдельных файлах в корне директории прилагаемого диска.

## 4. Описание процедур импорта и экспорта данных

База данных обычно имеет не самостоятельную ценность, является частью информационной системы. Независимо от того, сколько звеньев имеет 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 вариантной части объекта в виде XML.

XML — расширяемый язык разметки. XML разрабатывался как язык с простым формальным синтаксисом, удобный для создания и обработки документов программами и одновременно удобный для чтения и создания документов человеком. Язык называется расширяемым, поскольку он не фиксирует разметку, используемую в документах: разработчик волен создать разметку в соответствии с потребностями конкретной области, будучи ограниченным лишь синтаксическими правилами языка.

### 4.1. 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. Пример данного скрипта представлена в Приложении Г.

### 4.2. Процедура экспорта данных в XML-файл

Для работы процедуры экспорта данных в XML-файл необходимо изменить параметры конфигурации dbo при помощи скрипта, представленного на рисунке 4.2.1.

```
exec master.dbo.sp_configure 'show advanced options', 1;  
RECONFIGURE;  
exec master.dbo.sp_configure 'xp_cmdshell', 1;  
RECONFIGURE;
```

Рисунок 4.2.1. — Скрипт разрешения выполнения команд в cmd

Скрипт процедуры экспорта в XML-файл представлен в Приложении Д.

## 5. Технология полнотекстового поиска

Полнотекстовый поиск — автоматизированный поиск документов, при котором поиск ведётся не по именам документов, а по их содержанию, всему или существенной части.

Многие СУБД поддерживают методы полнотекстового поиска (Fulltext search), которые позволяют очень быстро находить нужную информацию в больших объемах текста.

В отличие от оператора LIKE, такой тип поиска предусматривает создание соответствующего полнотекстового индекса, который представляет собой своеобразный словарь упоминаний слов в полях. Под словом обычно понимается совокупность из не менее 3-х пробельных символов (но это может быть изменено). В зависимости от данных словаря может быть вычислена релевантность — сравнительная мера соответствия запроса найденной информации.

Все технологии полнотекстового поиска работают по одному принципу. На основе текстовых данных строится индекс, который способен очень быстро искать соответствия по ключевым словам.

Обычно сервис поиска состоит из двух компонент. Поисковик и индексатор. Индексатор получает текст на вход, делает обработку текста (вырезание окончаний, незначимых слов и т.п.) и сохраняет все в индексе. Устройство такого индекса позволяет проводить по нему очень быстрый поиск.

Чтобы работать с полнотекстовым поиском, сначала нам нужно создать полнотекстовой каталог. Далее нужно создать соответствующий индекс. Он называется *FULLTEXT*, и может быть наложен на поля CHAR, VARCHAR и TEXT. Затем можно создать полнотекстовые запросы.

```
CREATE FULLTEXT CATALOG MovieCatalog
with accent_sensitivity = on
as default
authorization dbo
go
ALTER FULLTEXT CATALOG TestCatalog
REBUILD WITH ACCENT_SENSITIVITY=OFF
GO
DROP FULLTEXT CATALOG MovieCatalog
GO
CREATE FULLTEXT INDEX ON MOVIE(PLOT)
KEY INDEX PK_Movie ON(MovieCatalog)
WITH (CHANGE_TRACKING AUTO)
```

Рисунок 4.1. — Создание полнотекстового каталога и индекса.

Скрипт процедуры полнотекстового поиска представлен в Приложении Е.

## 6. Тестирование

### 6.1 Тестирование производительности базы данных

Для тестирования производительности была взята за основу таблица CINEMA, так как она заполнена гораздо больше всех остальных таблиц.

Изначально таблица CINEMA была заполнена на 100000 строк. После этого был применён SELECT-запрос к данной таблице и при помощи стандартных средств IDE MS SQL Server Management Studio оценена цена выборки к таблице. Результат данной оценки запроса приведён на рисунке 6.1.

Clustered Index Scan (Clustered)	
Просмотр всего кластеризованного индекса или его части.	
Физическая операция	Clustered Index Scan
Логическая операция	Clustered Index Scan
Предполагаемый режим выполнения	Row
Хранилище	RowStore
Предполагаемая стоимость операций ввода-вывода	1,31572
Предполагаемая стоимость оператора	1,42588 (100%)
Предполагаемая стоимость ЦП	0,110158
Предполагаемая стоимость поддерева	1,42588
Предполагаемое количество выполнений	1
Предполагаемое количество строк	100001
Приблизительное число считываемых строк	100001
Предполагаемый размер строки	8165 B
Отсортировано	False
Идентификатор узла	0
Объект	
[Cinema].[dbo].[CINEMA].[PK_CINEMA_3214EC2784C87A6A]	
Список выходных столбцов	
[Cinema].[dbo].[CINEMA].ID; [Cinema].[dbo].[CINEMA].NAME; [Cinema].[dbo].[CINEMA].ADDRESS; [Cinema].[dbo].[CINEMA].WEBSITE; [Cinema].[dbo].[CINEMA].REGION; [Cinema].[dbo].[CINEMA].NUMBER_OF_HALLS; [Cinema].[dbo].[CINEMA].TICKET_OFFICE	

Рисунок 6.1 – Оценка запроса к таблице CINEMA без некластеризованного индекса

После проведения первоначальной оценки был построен некластеризованный индекс к таблице CINEMA и проведена оценка такого же SELECT-запроса к таблице CINEMA. Результаты, полученные во время оценки, представлены на рисунке 6.2.

Просмотр индекса (NonClustered)	
Просмотр всего некластеризованного индекса или его части.	
Физическая операция	Просмотр индекса
Логическая операция	Index Scan
Предполагаемый режим выполнения	Row
Хранилище	RowStore
Предполагаемая стоимость операций ввода-вывода	0,332014
Предполагаемая стоимость оператора	0,442174 (2%)
Предполагаемая стоимость ЦП	0,11016
Предполагаемая стоимость поддерева	0,442174
Предполагаемое количество выполнений	1
Предполагаемое количество строк	100003
Приблизительное число считываемых строк	100003
Предполагаемый размер строки	45 B
Отсортировано	False
Идентификатор узла	2
Объект	
[Cinema].[dbo].[CINEMA].[INDEX_CINEMA]	
Список выходных столбцов	
[Cinema].[dbo].[CINEMA].ID; [Cinema].[dbo].[CINEMA].NAME	

Рисунок 6.2 – Оценка запроса к таблице с построенным некластеризованным индексом

По результатам проведённых оценок до и после построения некластеризованного индекса, можно сделать вывод, что после создания индекса получили:

- стоимость по параметру “Предполагаемая стоимость операций ввода-вывода” в 4 раза меньше;
- стоимость по параметру “Предполагаемая стоимость оператора” в 3 раза;
- стоимость по параметру “Предполагаемая стоимость поддерева” в 3 раза.

Таким образом, постройка индекса к таблице была более чем оправдана, так как мы получили прирост производительности в 3 и более раза, в зависимости от параметра, характеризующего запрос.

## 6.2 Тестирование приложения.

При разработке приложения были учтены возможные исключительные ситуации.

Если пользователь забудет ввести одно из полей Логин или Пароль, то будет выведено окно с сообщением «Введите логин и пароль!».

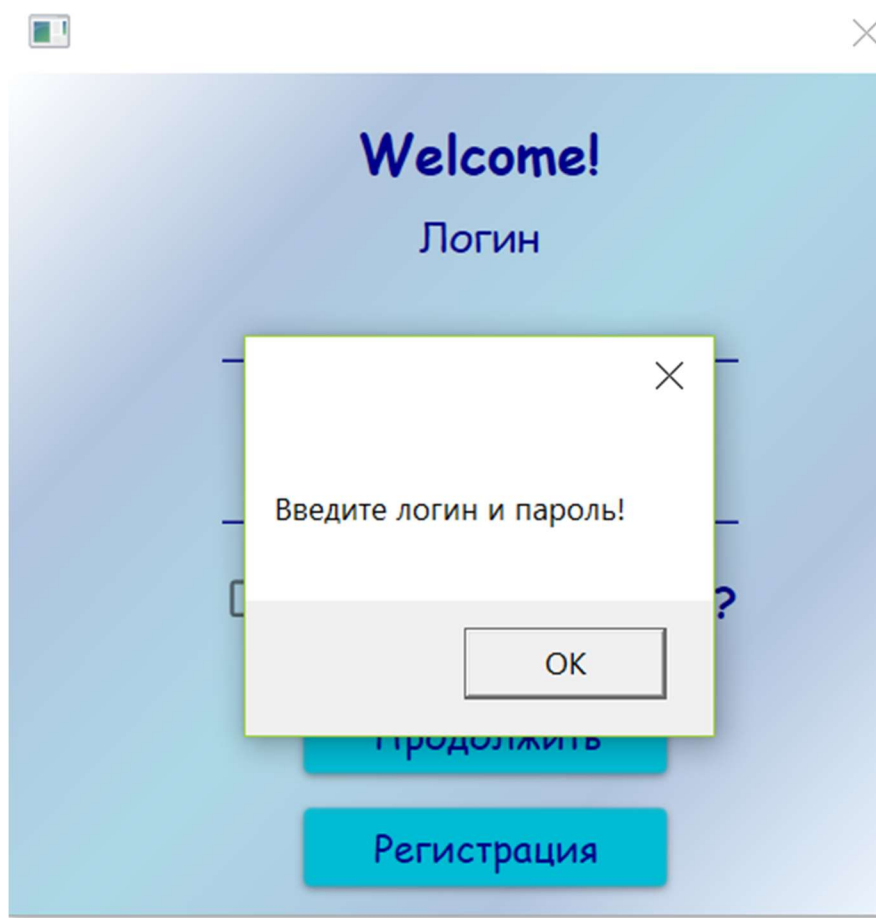


Рисунок 6.2.1. Пустое поле Логина и Пароля.

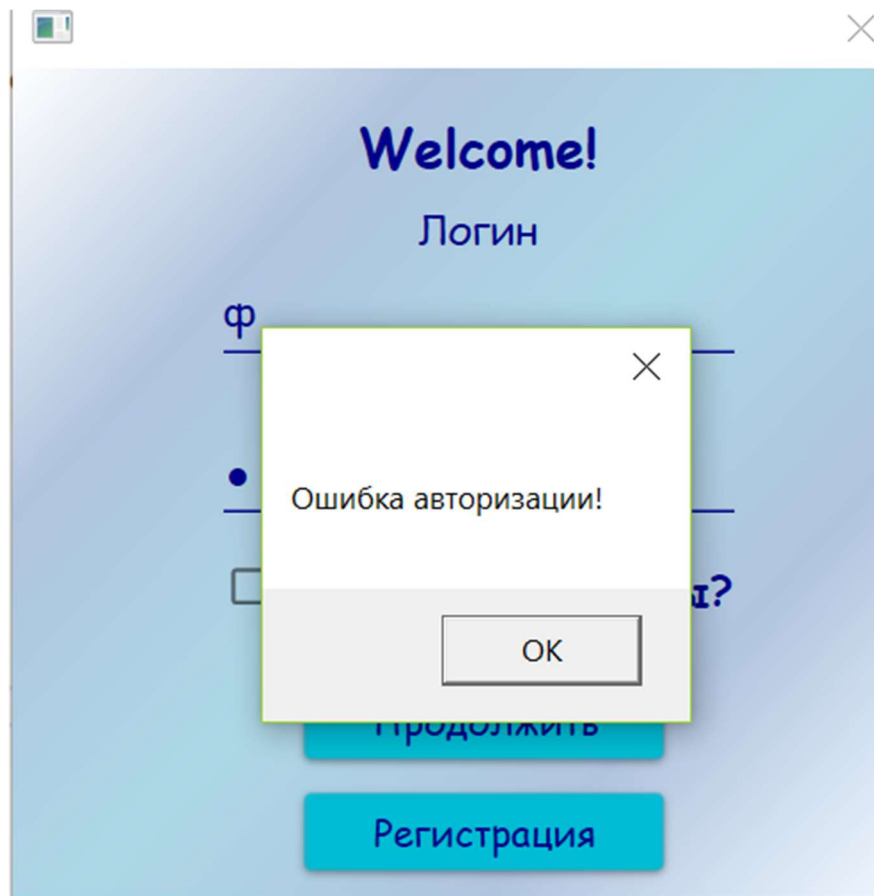


Рисунок 6.2.2. Веден неправильный Логин или Пароль.

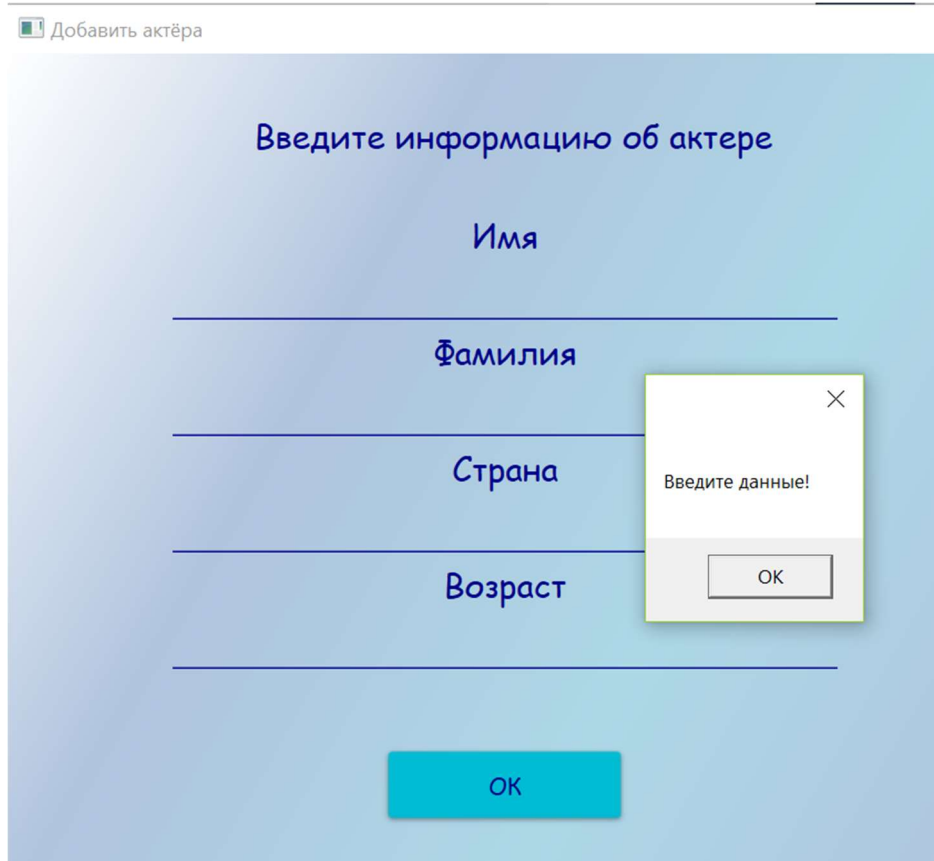


Рисунок 6.2.3. Не введены данные.

## 7. Руководство пользователя

### 7.1. Руководство пользователя администратора

По умолчанию в базе данных создан один администратор, который может добавлять, изменять и удалять информацию в базе данных.

Главное окно содержит кнопки для добавления, обновления, удаления данных. На рисунке 7.1.1 показано общее для всех окно управления.

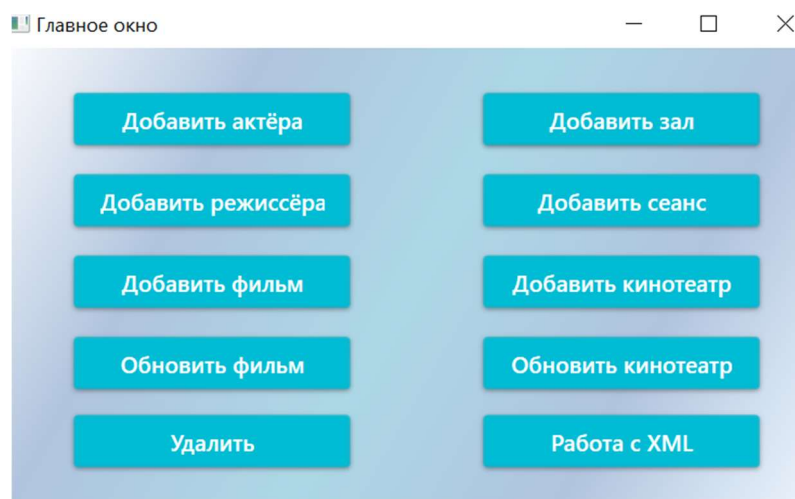


Рисунок 7.1.1. Окно управления

В окне добавления фильма есть поля для ввода информации о фильме и кнопка для добавления постера из файловой системы, а также кнопка для выхода в главное окно.

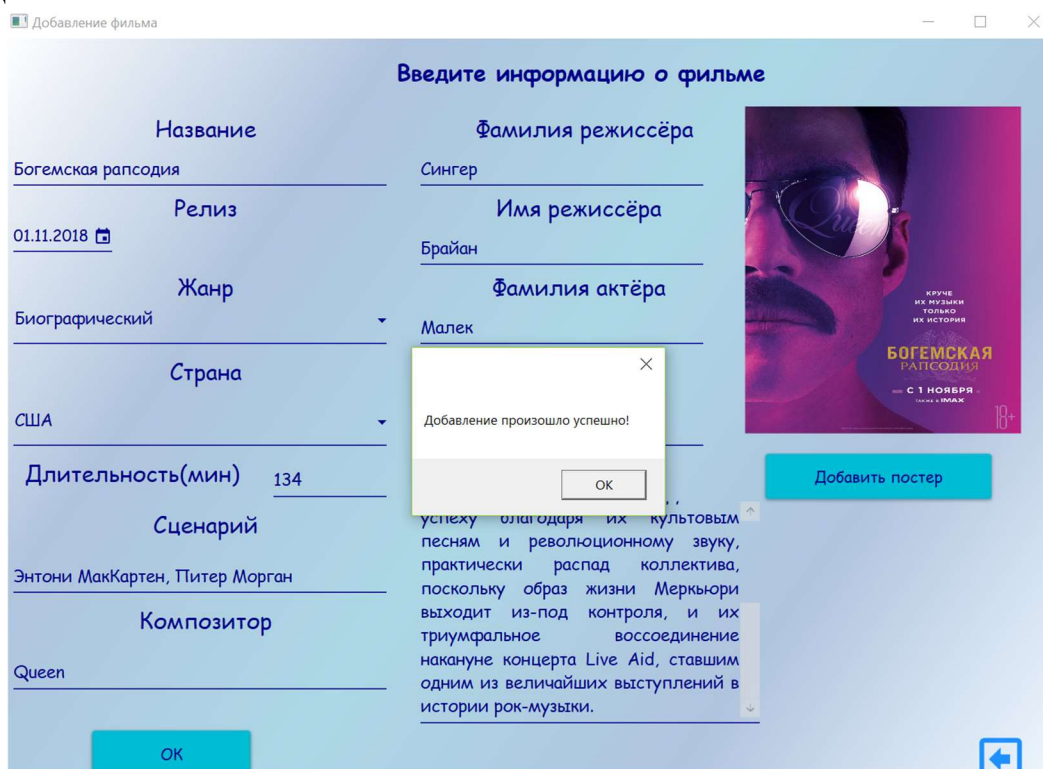


Рисунок 7.1.2. Окно добавления фильма



Окно Удалить предназначено для удаления информации о фильме или кинотеатре.

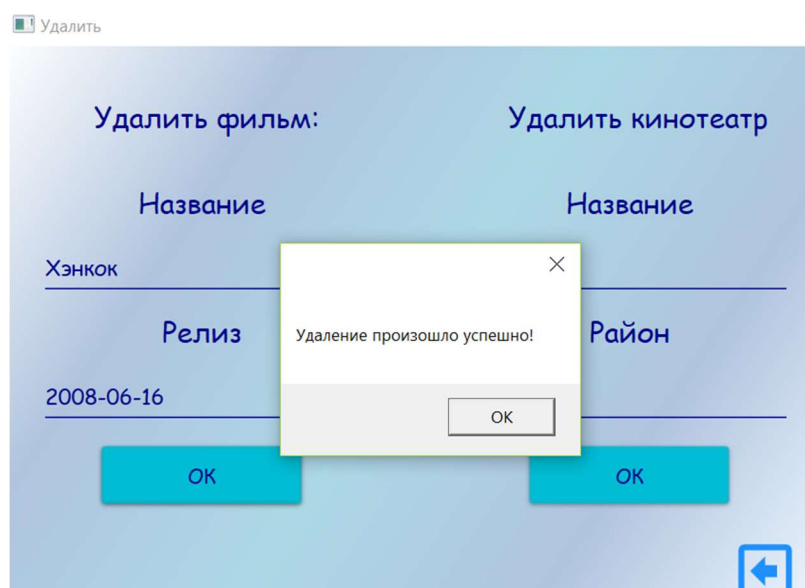


Рисунок 7.1.3. Окно Удалить.

Окно Обновить кинотеатр предназначено для обновления записей о кинотеатрах.

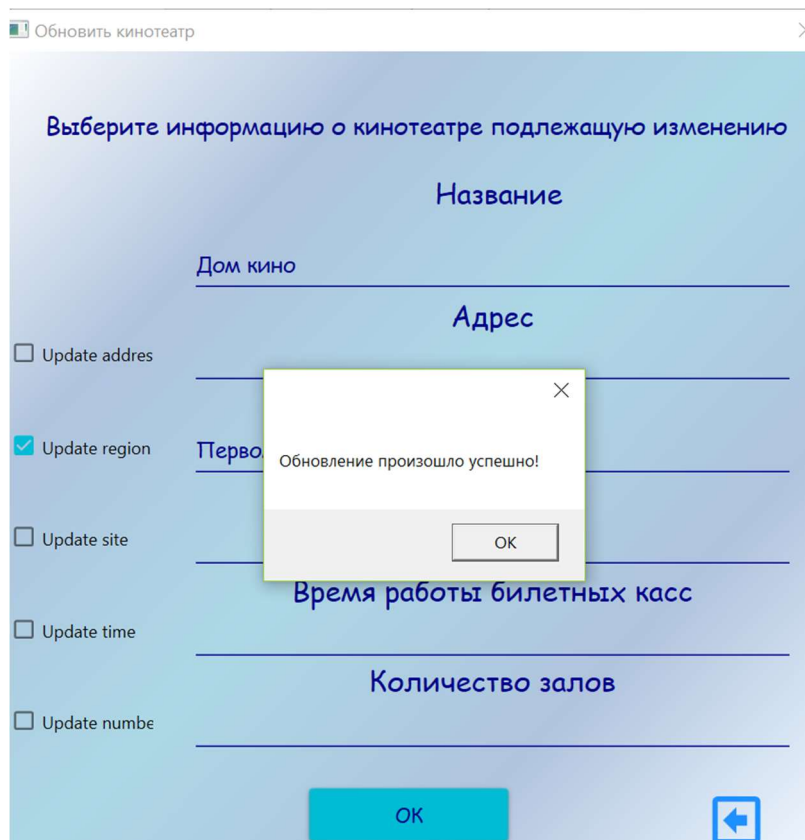


Рисунок 7.1.4. Окно Обновить кинотеатр.

Окно XML предназначено для импорта и экспорта файлов xml.



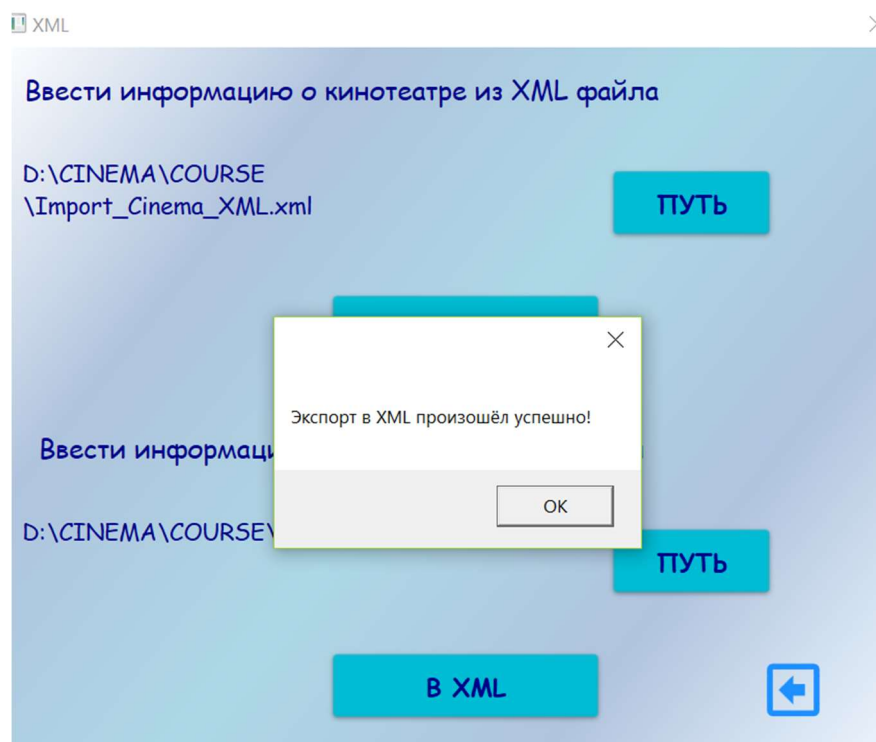


Рисунок 7.1.5. Окно XML.

## 7.2 Руководство пользователя приложением для клиента

Для входа в приложение необходимо авторизоваться. Если пользователь желает заказать билеты, то он должен выбрать пункт «Приобрести билеты». После этого появится диалоговое окно с паролем для покупки, который пользователь должен запомнить.

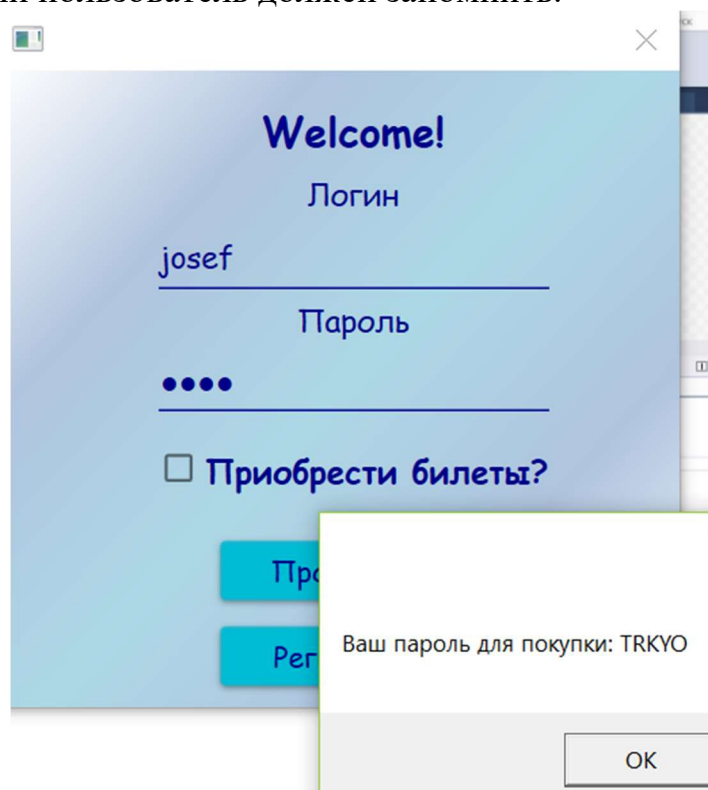


Рисунок 7.2.1. Окно авторизации.

Окно Регистрация предназначено для регистрации новых пользователей.

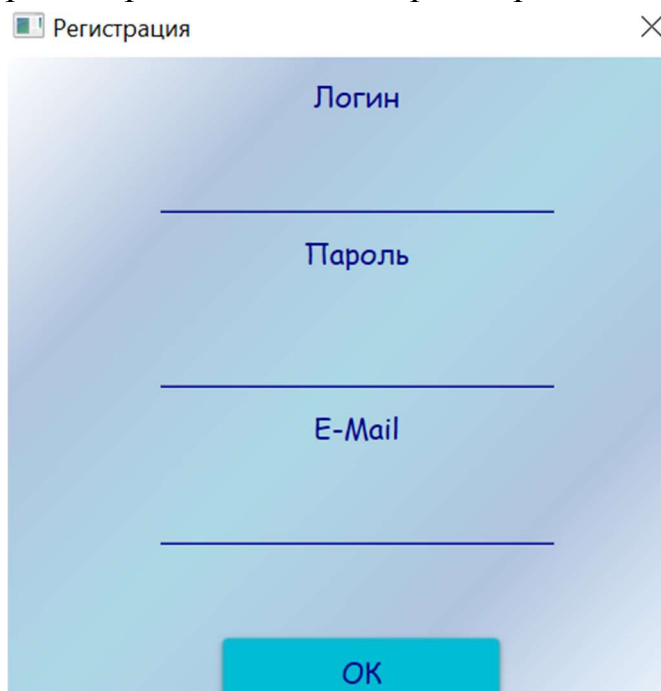


Рисунок 7.2.2. Окно Регистрация.

Главное окно приложения содержит кнопки “Сеансы”, “Заказ билетов”, “Кинотеатры”, “Фильмы”, “Полнотекстовой поиск”, “Актёры”.

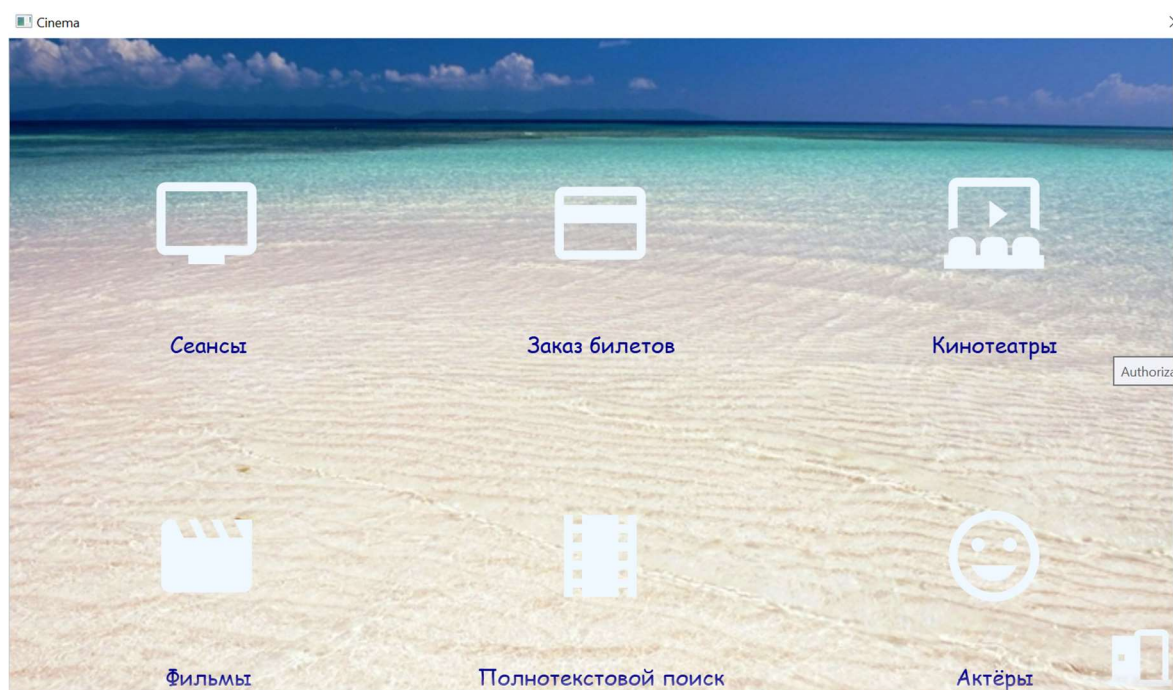


Рисунок 7.2.3. Главное окно.

Окно Сеансы позволяет просмотреть список сеансов, отсортировать их по дате.

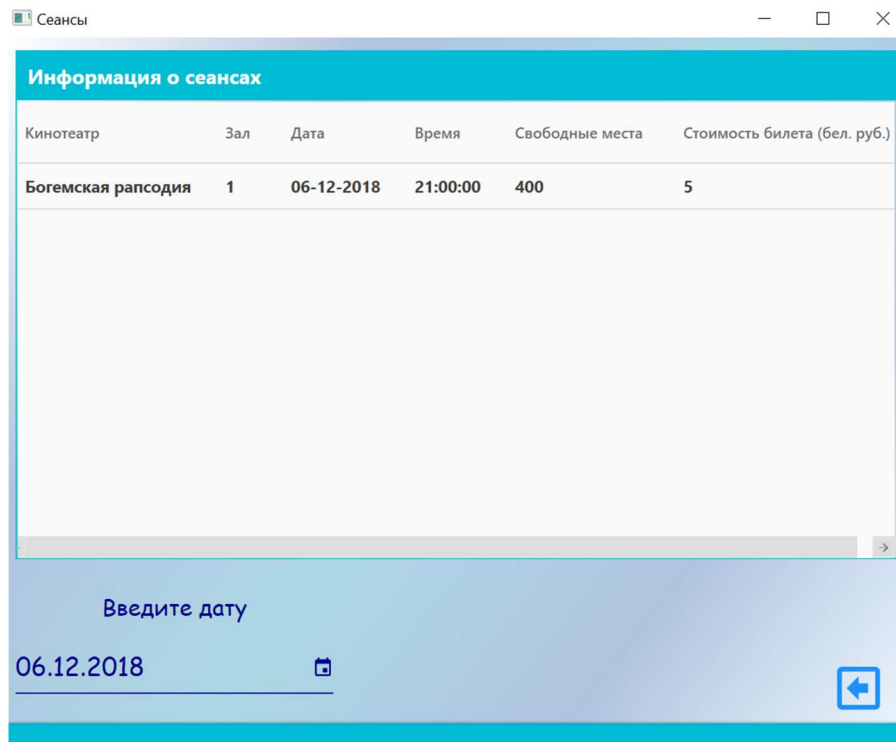


Рисунок 7.2.4. Окно Сеансы.

Окно Полнотекстовой поиск позволяет просмотреть список фильмов, найденных по введённой фразе.

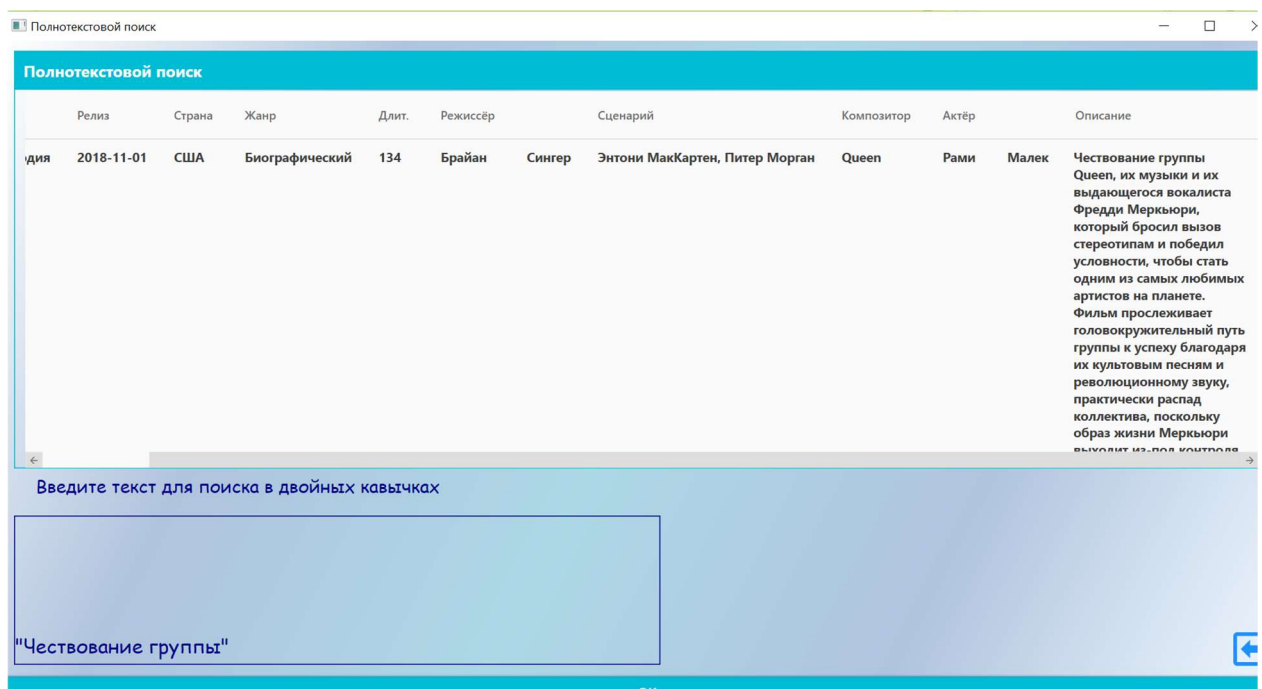



Рисунок 7.2.5. Окно Полнотекстовой поиск.

Окно Актёры позволяет просмотреть список актёров по введённым имени и фамилии. При установке в обоих полях нулей, вы получите весь список актёров.

Актёры

Информация об актёрах

Имя	Фамилия	Страна	Возраст
Рами	Малек	США	37



Введите имя актёра

Введите фамилию актёра

Рами

Малек

←

Рисунок 7.2.6. Окно Актёры.

Окно Заказать билеты позволяет купить билеты на определённый сеанс. Можно заказать максимум 10 билетов. Для покупки необходимо ввести пароль покупки, полученный в окне авторизации. Также можно узнать количество свободных мест и стоимость заказа.

Заказать билеты

Заказ билетов

Фильм

Богемская рапсодия

Кинотеатр

Дом кино

Зал

1

Дата сеанса

06.12.2018

Время

21:00

Пароль для покупки

FPBСХ

Места

Свободно

400

мест

Цена

Цена

20

руб.

Заказать

Выберите ряд и место (максимум 10 билетов)

Ряд	Место
12	6
12	5
5	7
5	8

←

Рисунок 7.2.7. Окно Заказать билеты.

## **Заключение**

В данном курсовом проекте была разработана база данных для кинотеатров. Также было разработано приложение для взаимодействия с базой данных. Помимо этого, была разобрана технология полнотекстового поиска.

В соответствии с полученным результатом, можно сказать, что разработанная программа работает верно, а требования технического задания выполнены в полном объеме.

### **Список использованной литературы**

1. Блинова Е.А. Курс лекций по базам данных / Е.А. Блинова
2. Пацей, Н.В. Технология разработки программного обеспечения / Н.В. Пацей. – Минск: БГТУ, 2016. – 129 с.
3. ProfessorWeb .NET & Web Programming [Электронный ресурс] – Режим доступа: <https://professorweb.ru> – Дата доступа: 30.11.2017.
4. Надёжное обслуживание баз MS SQL Server [Электронный ресурс] – Режим доступа: <https://habrahabr.ru/company/softlab/blog/266033/> - Дата доступа: 01.12.2017.

## Приложение А

```
--Genre
create table GENRE(
ID int primary key IDENTITY(1, 1),
NAME nvarchar(30) unique not null);

--Country
create table COUNTRY(
ID varchar(10) primary key not null,
NAME nvarchar(70) unique not null);

--Director
create table DIRECTOR(
ID uniqueidentifier primary key rowguidcol DEFAULT NEWSEQUENTIALID(),
NAME nvarchar(20) not null,
SURNAME nvarchar(20),
COUNTRY_ID varchar(10) foreign key references COUNTRY(ID),
AGE int,
IMAGE varbinary(max));

--Actor
create table ACTOR(
ID uniqueidentifier primary key rowguidcol DEFAULT NEWSEQUENTIALID(),
NAME nvarchar(20) not null,
SURNAME nvarchar(20),
COUNTRY_ID varchar(10) foreign key references Country(ID),
AGE int,
IMAGE varbinary(max));

--Movie
create table MOVIE(
ID uniqueidentifier rowguidcol DEFAULT NEWSEQUENTIALID(),
NAME nvarchar(max) not null,
RELEASE date,
COUNTRY_ID varchar(10) foreign key references COUNTRY(ID),
GENRE_ID int foreign key references GENRE(ID),
RUNNING_TIME int,
DIRECTOR_ID uniqueidentifier foreign key references DIRECTOR(ID),
SCREENPLAY nvarchar(50),
COMPOSER nvarchar(50),
ACTOR_ID uniqueidentifier foreign key references ACTOR(ID),
PLOT nvarchar(max),
IMAGE varbinary(max),
CONSTRAINT PK_Movie PRIMARY KEY CLUSTERED(ID));

--Cinema
create table CINEMA(
ID int primary key IDENTITY(1, 1),
NAME nvarchar(30) not null,
ADDRESS nvarchar(max),
WEBSITE nvarchar(max),
REGION nvarchar(30),
NUMBER_OF_HALLS int,
TICKET_OFFICE nvarchar(30));
```

--Hall

```
create table HALL(  
ID int primary key IDENTITY(1, 1),  
NAME NVARCHAR(30),  
CINEMA_ID int foreign key references CINEMA(ID),  
ROWS int not null,  
SEATS int not null);
```

--Session

```
create table SESSION(  
ID int primary key IDENTITY(1, 1),  
MOVIE_ID uniqueidentifier foreign key references MOVIE(ID),  
HALL_ID int foreign key references HALL(ID),  
DATE date not null,  
TIME time(7) not null,  
COST int not null,  
FREESEATS int);
```

--Users

```
create table USERS(  
ID uniqueidentifier primary key rowguidcol DEFAULT NEWSEQUENTIALID(),  
LOGIN nvarchar(50) not null,  
PASSWORD nvarchar(max) not null,  
EMAIL nvarchar(50) not null);
```

--Purchase

```
create table PURCHASE(  
ID int primary key IDENTITY(1, 1),  
USER_ID uniqueidentifier foreign key references USERS(ID),  
DATE smalldatetime);  
ALTER TABLE PURCHASE ADD PRICE INT;
```

--Tickets

```
create table TICKETS(  
ID int primary key IDENTITY(1, 1),  
SESSION_ID int foreign key references SESSION(ID),  
PURCHASE_ID int foreign key references PURCHASE(ID),  
ROW int not null,  
SEAT int not null);
```



## Приложение Б

```
USE master;
```

```
CREATE LOGIN ADMINISTRATOR_LOG  
  WITH PASSWORD = '1111',  
        DEFAULT_DATABASE = [Cinema],  
        DEFAULT_LANGUAGE = [русский];
```

```
CREATE USER ADMINISTRATOR  
  FOR LOGIN ADMINISTRATOR_LOG  
  WITH DEFAULT_SCHEMA = [dbo];
```

```
GRANT EXECUTE TO ADMINISTRATOR;
```

```
CREATE LOGIN CLIENT_LOG  
  WITH PASSWORD = '2222',  
        DEFAULT_DATABASE = [Cinema],  
        DEFAULT_LANGUAGE = [русский];
```

```
CREATE USER CLIENT  
  FOR LOGIN CLIENT_LOG  
  WITH DEFAULT_SCHEMA = [dbo];
```

```
GRANT EXECUTE ON OBJECT::Authorisation TO CLIENT;  
GRANT EXECUTE ON OBJECT::FullTextSearch TO CLIENT;  
GRANT EXECUTE ON OBJECT::GetActorInfo TO CLIENT;  
GRANT EXECUTE ON OBJECT::GetCinemaInfo TO CLIENT;  
GRANT EXECUTE ON OBJECT::GetMovieInfo TO CLIENT;  
GRANT EXECUTE ON OBJECT::GetSessionInfo TO CLIENT;  
GRANT EXECUTE ON OBJECT::InsertPurchase TO CLIENT;  
GRANT EXECUTE ON OBJECT::InsertTicket TO CLIENT;  
GRANT EXECUTE ON OBJECT::Registration TO CLIENT;  
GRANT EXECUTE ON OBJECT::SelectFreeSeats TO CLIENT;  
GRANT EXECUTE ON OBJECT::SelectPrice TO CLIENT;
```

## Приложение В

-----Авторизация пользователя по логину.

CREATE PROCEDURE Authorisation

    @login nvarchar(50),

    @password nvarchar(max),

    @rc int output

AS BEGIN

    set @rc = 0;

    IF EXISTS(SELECT ID FROM USERS WHERE LOGIN like @login and PASSWORD  
like @password)

    BEGIN

        set @rc = 1;

    END

END;

-----Создание покупки

CREATE PROCEDURE InsertPurchase

    @login nvarchar(50),

    @password nvarchar(max),

    @date date,

    @rc nvarchar(10) = " output

AS BEGIN

    DECLARE @user\_id uniqueidentifier;

    SET @user\_id = (SELECT ID FROM USERS WHERE LOGIN = @login and  
PASSWORD = @password);

    BEGIN

        SET @rc = (select char((rand()\*25 + 65))+char((rand()\*25 + 65))+char((rand()\*25 +  
65))+char((rand()\*25 + 65))+char((rand()\*25 + 65)));

        INSERT INTO PURCHASE(USER\_ID, DATE, UNIQUE\_PASSWORD)

        VALUES (@user\_id, @date, @rc);

        -- SET @rc = (select left(RAND(),5));

    END

END;

-----Добавление сеанса

CREATE PROCEDURE InsertSession

    @movie nvarchar(max),

    @cinema nvarchar(30),

    @hall nvarchar(30),

    @date date,

    @time time,

    @cost int,

    @rc int output

AS BEGIN

    SET @rc = 0;

    DECLARE @movie\_id uniqueidentifier;

    SET @movie\_id = (SELECT ID FROM MOVIE WHERE NAME = @movie);

    DECLARE @cinema\_id int;

    SET @cinema\_id = (SELECT ID FROM CINEMA WHERE NAME = @cinema);

    DECLARE @hall\_id int;

    SET @hall\_id = (SELECT ID FROM HALL WHERE CINEMA\_ID = @cinema\_id);

    DECLARE @free\_seats int;

    SET @free\_seats = (SELECT SEATS \* ROWS FROM HALL WHERE NAME =  
@hall);

    BEGIN

        INSERT INTO SESSION(MOVIE\_ID, HALL\_ID, DATE, TIME, COST,  
FREESEATS)

```

VALUES (@movie_id, @hall_id, @date, @time, @cost, @free_seats);
SET @rc = 1;
END

END;

-----Просмотр свободных мест
CREATE PROCEDURE SelectFreeSeats
@movie nvarchar(max),
@hall nvarchar(30),
@cinema nvarchar(30),
@date date,
@time time,
@freeseats int = NULL output
AS BEGIN
    SET @freeseats = (SELECT S.FREESEATS FROM SESSION S INNER JOIN MOVIE M ON M.ID
= S.MOVIE_ID
    INNER JOIN HALL H ON H.ID = S.HALL_ID
    INNER JOIN CINEMA C ON C.ID= H.CINEMA_ID
    WHERE M.NAME = @movie AND H.NAME = @hall AND DATE = @date AND TIME = @time);
    PRINT @freeseats;
END;

-----Просмотр стоимости заказа билетов
CREATE PROCEDURE SelectPrice
@pass nvarchar(10),
@price int = NULL output
AS BEGIN
    SET @price = (SELECT PRICE FROM PURCHASE WHERE UNIQUE_PASSWORD = @pass);
    PRINT @price;
END;

-----Вставка данных о кинотеатре.

CREATE PROCEDURE InsertCinema
@name nvarchar(30),
@address nvarchar(max),
@website nvarchar(max),
@region nvarchar(30),
@halls int,
@ticketoffice nvarchar(30),
@rc int output
AS BEGIN
    SET @rc = 0;
    BEGIN
        INSERT INTO CINEMA (NAME, ADDRESS, WEBSITE, REGION,
NUMBER_OF_HALLS, TICKET_OFFICE)
VALUES (@name, @address, @website, @region, @halls, @ticketoffice);
        SET @rc = 1;
    END
END;

-----Просмотр информации о сеансах.
CREATE PROCEDURE GetSessionInfo
@date date = NULL
AS BEGIN

    IF (@date IS NOT NULL)
        SELECT M.NAME, H.NAME, S.DATE, S.TIME, S.FREESEATS, S.COST
        FROM SESSION S INNER JOIN MOVIE M ON M.ID = S.MOVIE_ID
        INNER JOIN HALL H ON H.ID = S.HALL_ID WHERE DATE = @date;

```

```

ELSE IF (@date IS NULL)
BEGIN
SELECT M.NAME, H.NAME, S.DATE, S.TIME, S.FREESEATS, S.COST
FROM SESSION S INNER JOIN MOVIE M ON M.ID = S.MOVIE_ID
INNER JOIN HALL H ON H.ID = S.HALL_ID;
END
END;

```

-----Обновление информации о кинотеатрах.

```

CREATE PROCEDURE UpdateCinema

```

```

    @check2 bit,

```

```

    @check3 bit,

```

```

    @check4 bit,

```

```

    @check5 bit,

```

```

    @check6 bit,

```

```

    @name nvarchar(30),

```

```

    @address nvarchar(max),

```

```

    @website nvarchar(max),

```

```

    @region nvarchar(30),

```

```

    @halls int,

```

```

    @ticketoffice nvarchar(30),

```

```

    @rc bit output

```

```

AS BEGIN

```

```

    SET @rc = 1;

```

```

    BEGIN TRY

```

```

        IF @check2 = 1

```

```

        BEGIN

```

```

            IF EXISTS (SELECT NAME FROM CINEMA)

```

```

                UPDATE CINEMA SET ADDRESS = @address WHERE NAME = @name;

```

```

            END

```

```

        IF @check3 = 1

```

```

        BEGIN

```

```

            IF EXISTS (SELECT NAME FROM CINEMA)

```

```

                UPDATE CINEMA SET REGION = @region WHERE NAME = @name;

```

```

            END

```

```

        IF @check4 = 1

```

```

        BEGIN

```

```

            IF EXISTS (SELECT NAME FROM CINEMA)

```

```

                UPDATE CINEMA SET WEBSITE = @website WHERE NAME = @name;

```

```

            END

```

```

        IF @check5 = 1

```

```

        BEGIN

```

```

            IF EXISTS (SELECT NAME FROM CINEMA)

```

```

                UPDATE CINEMA SET TICKET_OFFICE = @ticketoffice WHERE NAME =

```

```

    @name;

```

```

        END

```

```

        IF @check6 = 1

```

```

        BEGIN

```

```

            IF EXISTS (SELECT NAME FROM CINEMA)

```

```

                UPDATE CINEMA SET NUMBER_OF_HALLS = @halls WHERE NAME =

```

```

    @name;

```

```

        END

```

```

    END TRY

```

```

    BEGIN CATCH

```

```

        SET @rc = 0;

```

```

    END CATCH

```

```

END;

```

## Приложение Г

```
CREATE PROCEDURE ImportFromXML
@path nvarchar(256),
@rc bit output
AS BEGIN
    SET @rc = 0;
        SET NOCOUNT ON;
        SET XACT_ABORT ON;
        BEGIN TRAN
            declare @result table(x xml);
            declare @sql nvarchar(300);
            set @sql = 'SELECT CAST(REPLACE(CAST(x AS VARCHAR(MAX)),
"encoding="utf-16"', "encoding="utf-8'") AS XML)
                                FROM OPENROWSET(BULK '"+@path+"',
SINGLE_BLOB) AS T(x);
            INSERT INTO @result EXEC(@sql);
            declare @xml XML;
            set @xml = (SELECT TOP 1 x FROM @result);

            INSERT INTO CINEMA(NAME, ADDRESS, WEBSITE, REGION,
NUMBER_OF_HALLS, TICKET_OFFICE)
                SELECT
                    C3.value('name[1]', 'nvarChar(30)') As NAME,
                    C3.value('address[1]', 'nvarChar(max)') AS ADDRESS,
                    C3.value('website[1]', 'nvarChar(max)') AS WEBSITE,
                    C3.value('region[1]', 'nvarChar(30)') AS REGION,
                    C3.value('number_of_halls[1]', 'int') AS NUMBER_OF_HALLS,
                    C3.value('ticket_office[1]', 'nvarChar(30)') AS TICKET_OFFICE
                FROM @xml.nodes('dataroot/cinema') AS T3(C3);
            SET @rc = 1;
        COMMIT;
    END;
```

## Приложение Д

```
exec master.dbo.sp_configure 'show advanced options', 1;

RECONFIGURE;

exec master.dbo.sp_configure 'xp_cmdshell', 1;

RECONFIGURE;

GO

CREATE PROCEDURE ExportToXML

    @path nvarchar(256),

    @rc bit output

AS BEGIN

    set @rc = 0;

    BEGIN TRANSACTION

        declare @sql nvarchar(500);

        SET @sql = 'bcp.exe "SELECT [ID], [NAME], [ADDRESS], [WEBSITE], [REGION],
[NUMBER_OF_HALLS], [TICKET_OFFICE] FROM CINEMA FOR XML PATH("List"),
ROOT("Root") " queryout "' + @path + "' -d Cinema -w -T';

        --SET @sql = 'bcp.exe "SELECT [ID], [NAME], [ADDRESS],
[WEBSITE], [REGION], [NUMBER_OF_HALLS], [TICKET_OFFICE] FROM CINEMA FOR XML
PATH " queryout "' + @path + "' -d Cinema -w -T';

        EXEC xp_cmdshell @sql;

        set @rc = 1;

    COMMIT;

END;
```

## Приложение Е

```
CREATE FULLTEXT CATALOG MovieCatalog
    with accent_sensitivity = on
    as default
    authorization dbo
go
ALTER FULLTEXT CATALOG TestCatalog
    REBUILD WITH ACCENT_SENSITIVITY=OFF
GO
DROP FULLTEXT CATALOG MovieCatalog

GO
CREATE FULLTEXT INDEX ON MOVIE(PLOT)
    KEY INDEX PK_Movie ON(MovieCatalog)
    WITH (CHANGE_TRACKING AUTO)
GO
CREATE PROCEDURE FullTextSearch
    @plot nvarchar(4000)
AS BEGIN
    BEGIN
        SELECT    M.NAME,    M.RELEASE,    C.NAME[COUNTRY],    G.NAME[GENRE],
        M.RUNNING_TIME, D.SURNAME, D.NAME, M.SCREENPLAY, M.COMPOSER, A.SURNAME,
        A.NAME, M.PLOT, M.IMAGE, M.TRAILER
        FROM MOVIE M INNER JOIN COUNTRY C ON C.ID = M.COUNTRY_ID
        INNER JOIN GENRE G ON G.ID = M.GENRE_ID
        INNER JOIN DIRECTOR D ON D.ID = M.DIRECTOR_ID
        INNER JOIN ACTOR A ON A.ID = M.ACTOR_ID
        WHERE CONTAINS (PLOT, @plot);
    END
END;
```