Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерный систем и сетей

Кафедра электронных вычислительных машин

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

ПРОГРАММА ДЛЯ ПРОДАЖИ БИЛЕТОВ В КИНО

Руководители В.В. Марченко

Разработчик А.А. Науменко

МИНСК 2018

СОДЕРЖАНИЕ

ВВЕДЕНИЕ3

1 АНАЛИЗ ПРОТОТИПОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСВУ4

1.1 Анализ существующих аналогов4

1.2 Постановка задачи8

2 МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ9

2.1 Проектирование схемы базы данных9

2.2 Алгоритм реализации задачи11

2.3 Описание сред реализации задачи12

3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА14

3.1 Физическая структура базы данных14

3.2 Проектирование интерфейса 17

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ19

5 ТЕСТИРОВАНИЕ21

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 23

ЗАКЛЮЧЕНИЕ 26

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 27

ПРИЛОЖЕНИЕ А Текст программы 28

ПРИЛОЖЕНИЕ Б Диаграмма вариантов использования 43

ПРИЛОЖЕНИЕ В Диаграмма деятельности44

ПРИЛОЖЕНИЕ Г Структурная схема63

**ВВЕДЕНИЕ**

Темой данного курсового проекта является разработка программы для продажи билетов в кино.

Наличие автоматизированных касс-киосков помогает зрителю получить всю необходимую информацию прямо перед покупкой билета, самостоятельно выбрать удобное место приобретения билета, что сокращает очереди в кассах мест проведения мероприятий.

Целью курсового проекта является разработка автоматизированной системы продажи билетов на сеансы в кино на языке программирования C++. При создании программного средства нужно учитывать специфику и особенности деятельности по продажам и распространению билетов в сфере культуры и искусства на территории Беларуси.

Задачи курсового проекта включают разработку графического интерфейса, возможность предоставления клиенту необходимой информации, возможность ее редактирования, осуществление поиска в программе.

Первая глава курсовой работы посвящена предпроектной стадии, то есть изучению объекта автоматизации, обоснованию необходимости создания системы комплексной автоматизации, формированию альтернативных концепций автоматизации, а также выбору наиболее подходящей.

Объектом курсового проекта является автоматизация процесса, что позволяет повысить производительность, сократить время выполнения процесса, снизить вероятность ошибок.

Предмет курсового проекта – продажа билетов на сеансы в кино.

Во второй и третьей главах описано техническое проектирование системы.

В четвертой, пятой и шестой главах данной курсовой работы представлена рабочая документация по программному средству.

**1 АНАЛИЗ ПРОТОТИПОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ**

**1.1 Анализ существующих аналогов**

Автоматизация бизнес процессов – залог эффективного управления. Автоматизация бизнес процессов приводит к уменьшению рутинных операций, позволяет намного быстрее обслуживать клиентов, дает больше возможностей для контроля, бизнес-процессы становятся более «прозрачными». Существенно улучшается работа по планированию закупок и поставок и другие плюсы. Все это в свою очередь существенно увеличивает рост прибыли, товарооборота и выручки, сокращает издержки. Уменьшение рутинных операций способствует значительно способствует сокращению затрат на персонал.

В настоящее время можно проследить тенденции к расширению рынка предоставления всевозможных развлекательных услуг. К данной среде относятся и кинотеатры. Кинотеатр – общественное здание или его часть, оборудованные для публичной демонстрации кинофильмов. Главное помещение кинотеатра – зрительный зал с экраном большого размера и системой воспроизведения звука, состоящей из нескольких громкоговорителей, обеспечивающих объемный звук.

Количество кинотеатров неумолимо увеличивается как в больших городах, население которых переваливает за миллион, так и в городах поменьше.

Сегодня используются автоматизированные системы, которые  представляют собой комплекс программ, обеспечивающих учет билетов на киносеансы. Они разработаны для увеличения объема продаваемых билетов, производительности кассиров, уменьшения и облегчения труда бухгалтеров и менеджеров.

В век информационных технологий с каждым годом количество автоматизированных продуктов возрастает. В связи с этим можно привести несколько аналогов программных средств, предоставляющих услуги бронирования билетов.

Билетная система «Чудобилет», которая представлена на рисунке 1.1, обладает следующими функциями:

## возможность работы с клубными картами;

* On-line продажи и бронирования билетов;
* управление ценами мест и продажей билетов;
* создание афиши.



Рисунок 1.1 - Билетная система «Чудобилет»

Автоматизация касс на платформе "1С:Предприятие 8" представлена на рисунке 1.2. Она обладает следующими возможностями:

* предоставление информации о сеансах;
* возможность работы со схемами залов;
* управление ценами мест и продажей билетов;
* печать билетов на бланках строгой отчетности;
* учёт бланков строгой отчетности;
* ведение учёта билетов и мест, переданных на реализацию другим распространителям;
* специализированная отчётность (накладные на реализацию билетов, накладные на возврат);
* широкие возможности обмена информацией с другими системами, с интернет-сайтами;
* масштабируемость и управляемость (широкие возможности адаптации программы под конкретные задачи).

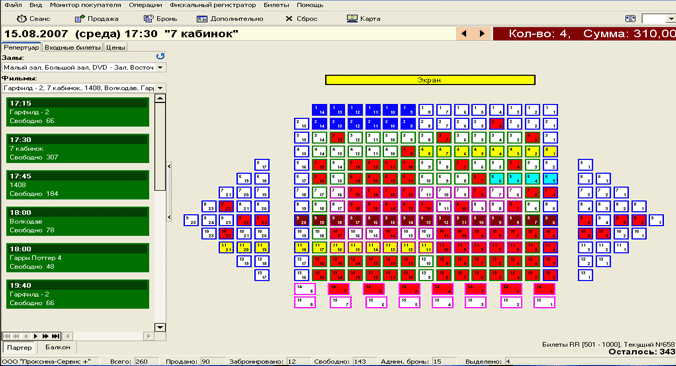


Рисунок 1.2 - Автоматизация касс на платформе «1С:Предприятие 8»

Система "Премьера" представлена на рисунке 1.3.

Возможности программы:

* Internet-справочная по расписанию сеансов и наличию мест;
* бронирование билетов через Internet;
* взаимодействие кассы и менеджерского компьютера через Internet;
* создание схемы зрительного зала;
* построение отчетов.

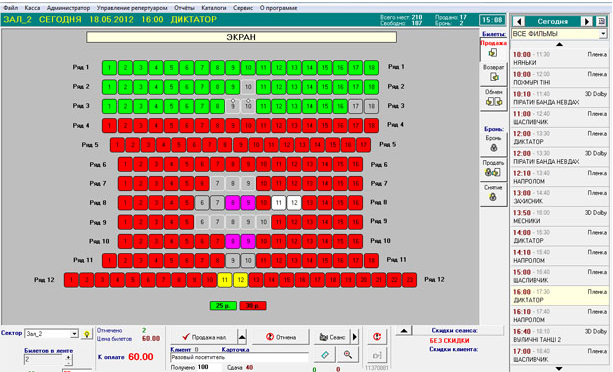


Рисунок 1.3 - Система «Премьера»

Рассмотрев аналоги разрабатываемого программного продукта, можно сказать, что разрабатываемый продукт включает в себя множество достоинств аналогов и предоставит простой интерфейс с широкими возможностями для пользователя. Из недостатков системы можно выделить отсутствие возможности бронирования билетов в On-line режиме.

Как и любая компьютерная программа, данное программное средство обрабатывает входные данные и выводит результат в виде выходных данных.

Входными данными разрабатываемой ПС являются:

* запросы пользователя: на поиск, бронирование, редактирование информации, просмотр справочной информации;
* критерии поиска (название, дата);
* информация о бронировании (название, время, дата начала спектакля, , забронированное место, стоимость билета);

Выходными данными разрабатываемого ПС являются:

* информация отображаемая на экран по запросу пользователя;
* результаты поиска;
* изменённые данные.

Постоянные данные:

* БД;
* схема зрительного зала.

В данном курсовом проекте поставлена задача создания программы для продажи билетов в кино. Поэтому при её выполнении было проведено исследование данной предметной области. В рамках данного курсового проекта реализуется проектирование и разработка базы данных по предметной области. Полученная БД не будет отражать всю бизнес-логику данной предметной области, а ограничиться лишь её частью.

**1.2 Постановка задачи**

Целью данного курсового проекта является создание программы для хранения и обработки данных о совершенных заказах, которое позволяет:

* предоставить информацию клиенту о фильме, сеансах, свободных местах в зале;
* бронировать места;
* редактировать информацию;
* осуществлять поиск информации в программе по критериям;

Разрабатываемое ПС должно быть обеспечено простой формой интерфейса, содержать краткие описания элементов меню, быть простой в использовании. Необходимо разработать базу данных, в которую будет заносится и редактироваться информация о сеансах и фильмах.

**2 МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ**

**2.1 Проектирование схемы базы данных**

На этапе проектирования схемы базы данных необходимо разработать логическую модель для физической реализации базы данных. Удачная разработка базы данных обеспечивает простоту ее поддержания. Данные следует сохранять в таблицах, причем каждая таблица должна содержать информацию одного типа.

Исходя из описания предметной области организацию данных можно представить набором следующих отношений:

* фильмы;
* сеансы;
* билет;
* место;
* состояние;
* возрастное ограничение;
* жанры;
* страна;
* режиссер;
* актеры.

Далее следует идентифицировать связи.

Связь – это некоторая ассоциация между двумя отношениями. Связи позволяют по одному отношению находить другие отношения, связанные с ним. Они бывают следующих типов:

* один-к-одному;
* один-ко-многим;
* многие-к-одному;
* много-ко-многим.

Связь типа один-к-одному означает, что один экземпляр одного отношения связан с одним экземпляром второго отношения.

Связь типа один-ко-многим означает, что один экземпляр одного отношения связан с несколькими экземплярами второго отношения. Это наиболее часто используемый тип связи. Первое отношение (со стороны «один») называется родительским, второе (со стороны «много») – дочерним.

Связь типа один-ко-многим означает, что несколько экземпляров одного отношения связан с одним экземпляром второго отношения.

Связь типа много-ко-многим означает, что каждый экземпляр одного отношения может быть связан с несколькими экземплярами второго отношения, и каждый экземпляр второго отношения может быть связан с несколькими экземплярами первого отношения.

Далее при проектировании схемы базы данных следует определить атрибуты отношений.

Отношение «Фильм» имеет следующий набор атрибутов:

* название;
* длительность;
* описание.

Отношение «Сеансы» имеет следующие атрибуты:

* дата начала сеанса;
* время начала сеанса.

Отношение «Билеты» имеет следующие атрибуты:

* информация о забронированном месте.

Отношение «Возрастное ограничение» имеет следующие атрибуты: возрастное ограничение.

Отношение «Жанр» имеет следующий атрибут: жанр.

Отношение «Страна» имеет следующий атрибут: страна.

Отношение «Режиссер» имеет следующие атрибуты: имя.

Отношение «Актеры» имеет следующий атрибут: имя.

Схема БД представлена на рисунке 2.1.

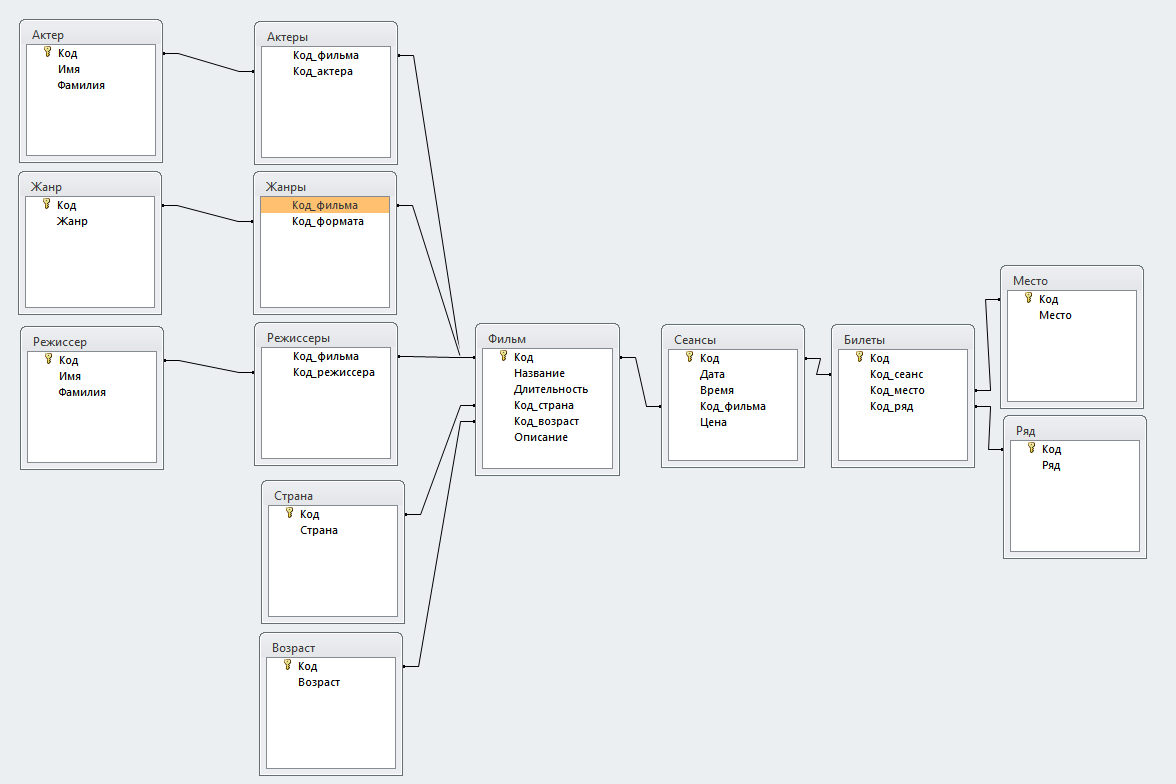


Рисунок 2.1 – схема БД

**2.2 Алгоритм реализации задачи**

Разрабатываемое программное средство предназначено для учета билетов на киносеансы.

В зависимости от прав доступа к программному средству будет реализовано два режима работы. В режиме пользователя возможны функции просмотра информации о фильме, сеансе, количестве занятых мест, схеме зрительного зала, брони билета. В режиме администратора будет предоставлен доступ ко всей информации и возможность ее редактирования.

Алгоритм работы программы дял продажи билетов представлен в приложении В.

Проектирование информационных систем, включающих в себя базы данных, осуществляется на физическом и логическом уровнях. Решение проблем проектирования на физическом уровне во многом зависит от используемой системы управления базами данных.

**2.3 Описание сред реализации задачи**

Базы данных – совокупность данных, организованная по определенным правилам, предусматривающая общие принципы описания, хранения, манипулирования данными, независимыми от прикладных программ.

СУБД – совокупность программ, предназначенных для управления БД и возможности получения пользователями необходимой информации из базы. В задачи СУБД входят:

* формирование и поддержание БД;
* обработка информации;
* прием запросов;
* предоставление информации пользователям;
* обеспечение целостности и реорганизации ценностей БД;
* организация совместной работы пользователей.

Для организации базы данных данного курсого проекта был выбран Microsoft Access. Microsoft Access — [реляционная система управления базами](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D1%8F%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%A1%D0%A3%D0%91%D0%94) данных корпорации [Microsoft](https://ru.wikipedia.org/wiki/Microsoft). Входит в состав пакета Microsoft Office. Имеет широкий спектр функций, включая связанные запросы, связь с внешними таблицами и [базами данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85). Благодаря встроенному языку [VBA](https://ru.wikipedia.org/wiki/VBA), в самом Access можно писать приложения, работающие с базами данных. С помощью Access можно создавать удобные веб-приложения для работы с базами данных, которые значительно облегчают ведение работы. Данные автоматически сохраняются в базе данных SQL, поэтому они надежно защищены. В приложениях автоматически создается привлекательный, удобный и единообразный интерфейс.

Для разработки интерфейса и программной составляющей была выбрана среда разработки Qt и язык программирования C++.

**Qt** —  это **кроссплатформенный фреймворк**  для разработки ПО на языке программирования **C++**. **Qt** полностью объектно-ориентированная, кросс-платформенная. Дает возможность разрабатывать платформо-независимое ПО, написанный код можно компилировать для **Linux, Windows, Mac OS X** и других операционных систем. Включает в себя множество классов для работы с сетью, базами данных, классы-контейнеры, а также для создания графического интерфейса и множество других.

Приложение разрабатывается на языке программирования C++, который широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования. Область его применения включает создание [операционных систем](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), разнообразных прикладных программ, [драйверов](https://ru.wikipedia.org/wiki/%D0%94%D1%80%D0%B0%D0%B9%D0%B2%D0%B5%D1%80)устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также развлекательных приложений.

**3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

**3.1 Физическая структура базы данных**

Реляционная база данных – это совокупность отношений, содержащих всю информацию, которая должна храниться в БД. Пользователи могут воспринимать такую базу данных как совокупность таблиц. При проектировке таблиц, необходимо руководствоваться следующими основными принципами:

* не должно быть повторений и между таблицами;
* каждая таблица должна содержать информацию только на одну тему;
* каждое поле должно быть связано с темой таблицы;
* не рекомендуется включать в таблицу данные, которые являются результатом выражения;
* в таблице должна присутствовать вся необходимая информация;
* информацию следует разбивать на наименьшие логические единицы.
* каждая таблица состоит из однотипных строк и имеет уникальное имя;
* строки имеют фиксированное число полей (столбцов) и значений (множественные поля и повторяющиеся группы недопустимы). Иначе говоря, в каждой позиции таблицы на пересечении строки и столбца всегда имеется в точности одно значение или ничего;
* строки таблицы обязательно отличаются друг от друга хотя бы единственным значением, что позволяет однозначно идентифицировать любую строку такой таблицы;
* столбцам таблицы однозначно присваиваются имена, и в каждом из них размещаются однородные значения данных (даты, фамилии, целые числа или денежные суммы);
* полное информационное содержание базы данных представляется в виде явных значений данных и такой метод представления является единственным. В частности, не существует каких-либо специальных "связей" или указателей, соединяющих одну таблицу с другой;
* при выполнении операций с таблицей ее строки и столбцы можно обрабатывать в любом порядке безотносительно к их информационному содержанию. Этому способствует наличие имен таблиц и их столбцов, а также возможность выделения любой их строки или любого набора строк с указанными признаками.

В таблицах данные распределяются по столбцам (которые называют полями) и строкам (которые называют записями). Все данные, содержащиеся в поле таблицы, должны иметь один и тот же тип. Каждое поле таблицы характеризуется наименованием, типом и шириной поля. При задании типа дан­ных поля можно также указать размер, формат и другие параметры, влияющие на отображение значения поля и точность числовых данных. Основные типы данных:

* текстовый. Текст или числа не требующие проведения расчётов;
* МЕМО. Поле этого типа предназначено для хранения небольших текстовых данных (до 64000 символов). Поле этого типа не может быть ключевым или проиндексированным;
* числовой. Этот тип данных содержит множество подтипов. От выбора подтипа (размера) зависит точность вычислений;
* счётчик. Уникальные, последовательно возрастающие числа, автоматически вводящиеся при добавлении новой записи в таблицу;
* логический. Логические значения, а так же поля, которые могут содержать одно из двух возможных значений;
* денежный. Денежные значения и числовые данные, используемые в математических вычислениях;
* Дата/Время. Дата и время хранятся в специальном фиксированном формате;
* поле объекта OLE. Включает звукозапись, Рис. и прочие типы данных. Поле этого типа не может быть ключевым или проиндексированным;
* гиперсвязь. Содержит адреса Web-страниц.

Используемые типы данных продемонстрированы в таблице 3.1:

Таблица 3.1 – типы данных в таблицах

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Поле | Тип данных | | Поле | Тип данных | |
| Таблица «Фильм» | | | Таблица «Билет» | | |
| идентификатор фильма | | int | идентификатор билета | | int |
| название | | varchar | идентификатор сеанса | | int |
| длительность | | time | идентификатор места | | int |
| дата начала проката | | date | идентификатор ряда | | int |
| идентификатор страны | | int | Таблица «Возрастное ограничение» | | |
| идентификатор возрастного ограничения | | int | идентификатор возрастного ограничения | | int |
| описание | | МЕМО | возрастное ограничение | | varchar |
| Таблица «Жанр» | | |
| Таблица «Ряд» | | | идентификатор жанра | | int |
| идентификатор ряда | | int | жанр | | varchar |
| номер ряда | | int | Таблица «Страна» | | |
|  | |  | идентификатор производителя | | int |
| Таблица «Сеансы» | | | страна | | varchar |
| идентификатор фильма | | int | Таблица «Режиссер» | | |
| дата начала сеанса | | date | идентификатор продюсера | | int |
| время начала сеанса | | time | имя | | varchar |
| Таблица «Место» | | | фамилия | | varchar |
| идентификатор места | | int | Таблица «Жанры» | | |
| номер места | | int | идентификатор формата | | int |
| идентификатор фильма | | int |
| Таблица «Актеры» | | | Таблица «Актер» | | |
| идентификатор актера | | int | идентификатор актера | | int |
| идентификатор фильма | | varchar | актер | | varchar |

Разрабатываемая в данном курсовом проекте база данных для сотрудника кинотеатра, позволяет автоматизировать весь процесс редактирования данных. При появлении новых задач разработчик может в кратчайшие сроки реализовать их в базе данных, путем добавления строк, столбцов и целых таблиц.

**3.2 Проектирование интерфейса**

Интерфейс программы будет реализован на основе диалоговых окон.

Диалоговые окна – это окна, предназначенные для открытия, создания, печати и закрытия документов, окна отображения и настройки всевозможных параметров.

Выбор действия

Схема зрительного зала

Бронь билета\ Просмотр занятых мест

Рисунок 3.1 – Интерфейс формы «Бронь билета»

В разрабатываемом программном средстве будут реализованы формы, в которых имеются таблиыцы, предназначенные для отображения содержимого базы данных, рисунок 3.2.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Номер билета | Название фильма | Время начала | Дата начала | Место\Ряд | Стоимость билета |
|  |  |  |  |  |  |

Рисунок 3.2 – Таблица с информацией

Для разработки потребовались следующие компоненты:

* pushbutton – элемент управления "кнопка", реагирующий на событие Click;
* textEdit – элемент управления, который может использоваться для ввода информации;
* label – возвращает или задает текстовое содержимое;
* pictureBox – используется для отображения рисунка;
* comboBox – предназначен для выбора значения, с раскрывающимся списком, который можно отображать и скрывать нажатием принадлежащей этому элементу управления кнопки со стрелкой;
* radioButton – позволяет пользователю выбрать единственный вариант из группы доступных, когда используется вместе с другими элементами управления;

Элементом для отображения табличных данных используется tableView.

**4** **РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ**

В разрабатываемом программном средстве существует 7 классов: «addfilm», «addschedule», «authorization», «editing, «main», «movie».

Класс «Movie»:

* void query (QString question,QSqlQueryModel \*setquery) – запрос на поиск;
* void Movie::recieveData(QString str) – получение информации с родительской формы, заполнение таблиц и полей данными из БД;
* void Movie::on\_tableView\_clicked(const QModelIndex &index) – отображение купленных билетов в зависимости от выбранного сеанса;
* void Movie::on\_pushButton\_clicked() – бронь билета;

Класс «Authorization»:

* QStringList fun\_file\_enter ( QString filename) – считывание с файла логина и пароля;
* void fun\_file\_in\_out (QString filename, bool check)– проверка на правильность ввода логина и пароля;
* void Authorization::on\_pushButton\_clicked() – вход в режим администратора;
* void Authorization::on\_pushButton\_2\_clicked()– выход из режима администратора.

Класс «Authorization»:

* void query (QString question,QSqlQueryModel \*setquery) – запрос на поиск;
* MainWindow::MainWindow(QWidget \*parent):QMainWindow(parent), ui(new Ui::MainWindow) – инициализация класса, извлечение начальной информации из БД;
* void MainWindow::onButtonSend() – передача информации другой форме;
* void MainWindow::on\_pushButton\_clicked() – открытие формы со схемой зрительного зала;
* void MainWindow::on\_pushButton\_2\_clicked() – форма редактирования БД;
* void MainWindow::on\_pushButton\_3\_clicked() – выход;
* void MainWindow::on\_pushButton\_6\_clicked() – обновление данных;
* void MainWindow::on\_pushButton\_5\_clicked() – поиск сеансов по дате;
* void MainWindow::on\_pushButton\_7\_clicked() – переход к форме для авторизации.

Класс «addfilm»:

* QString find\_key (QString question)- нахождение значения ключа;
* void addfilm::on\_pushButton\_clicked()– добавление фильма;

Класс «addschedule»:

* QString find\_key (QString question)- нахождение значения ключа;
* void addschedule::on\_pushButton\_4\_clicked()– добавление сеанса;
* addschedule::addschedule(QWidget \*parent): QDialog(parent), ui(new Ui::addschedule) – инициализация класса.

Класс «editing»:

* QString find\_key (QString question)- нахождение значения ключа;
* void editing::on\_pushButton\_clicked() – отображение всех фильмов;
* void editing::on\_pushButton\_2\_clicked()–отображение всех сеансов;
* void editing::on\_pushButton\_3\_clicked()–отображение всех купленных билетов;
* void editing::on\_pushButton\_4\_clicked() – отображение новых форм;
* void editing::on\_pushButton\_5\_clicked() – удаление данных;

**5 ТЕСТИРОВАНИЕ**

Тестирование программного обеспечения – процесс исследования, испытания [программного продукта](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), имеющий две различные цели:

* продемонстрировать разработчикам и заказчикам, что программа соответствует требованиям;
* выявить ситуации, в которых поведение программы является неправильным, нежелательным или не соответствующим спецификации.

Уровни тестирования

* [модульное тестирование](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) – тестируется минимально возможный для тестирования компонент, например, отдельный класс или функция.
* [интеграционное тестирование](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) – тестируются интерфейсы между компонентами, подсистемами или системами.
* [системное тестирование](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%BD%D0%BE%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) – тестируется интегрированная система на её соответствие [требованиям](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B5%D0%B1%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%BC%D1%83_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8E).
* альфа-тестирование – имитация реальной работы с системой штатными [разработчиками](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%81%D1%82), либо реальная работа с системой потенциальными пользователями/заказчиком. Чаще всего альфа-тестирование проводится на ранней стадии разработки продукта, но в некоторых случаях может применяться для законченного продукта в качестве внутреннего приёмочного тестирования.
* [бета-тестирование](https://ru.wikipedia.org/wiki/%D0%91%D0%B5%D1%82%D0%B0-%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) – в некоторых случаях выполняется распространение предварительной версии для некоторой большей группы лиц с тем, чтобы убедиться, что продукт содержит достаточно мало ошибок. Иногда бета-тестирование выполняется для того, чтобы получить обратную связь о продукте от его будущих пользователей.

Таблица 4.1 – Ход тестирования

|  |  |  |
| --- | --- | --- |
| Ход тестирования | Ожидание | Результат |
| Удаление фильма | Изменение БД | БД изменена |
| Добавление фильма | Изменение БД | БД изменена |
| Осуществление поиска | Вывод необходимой информации | Вывод необходимой информации |
| Изменение БД | Изменения информации в программе | Соответствующие изменения в программе |
| Добавление нового билета | Изменение БД, Изменение схемы зрительного зала | БД и схема зрительного зала изменена |
| Удаление билета | Изменение БД, Изменение схемы зрительного зала | БД и схема зрительного зала изменена |
| Изменение данных пользователем | Запрет на редактирование | Вывод предупреждения |
| Авторизация | Доступ ко всей информации | Доступ ко всей информации |
| Ввод некорректных данных | Запрет на добавление | Вывод предупреждения |
| Поиск с пустыми полями | Запрет на поиск | Вывод предупреждения |
| Добавление с пустыми полями | Запрет на добавление | Вывод предупреждения |

В ходе тестирования программы для продажи билетов ошибок не найдено.

**6. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

Чтобы запустить программное средство необходимо запустить «project.exe». При его открытии откроется окно, которое представлено на рисунке 5.1.

Начало работы осуществляется с открытия главной формы, на котором пользователю предоставляется информация о текущих фильмах. Чтобы выбрать интересующий фильм на определенный день можно использовать критерии поиска. Для этого введите дату и нажмите и нажмите на кнопку "Выбрать дату". Для покупки билета нажмите на соответствующую кнопку.

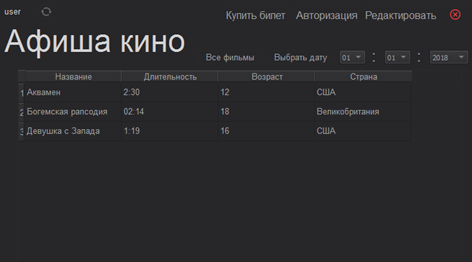


Рисунок 5.1 – Главное меню

На форме покупки билета, показанной на рисунке 5.2 можно просмотреть количество купленных на определенный сеанс. Если желаете купить введите в поле «ряд» и «место» соответствующую информацию и нажмите на кнопку "Купить".

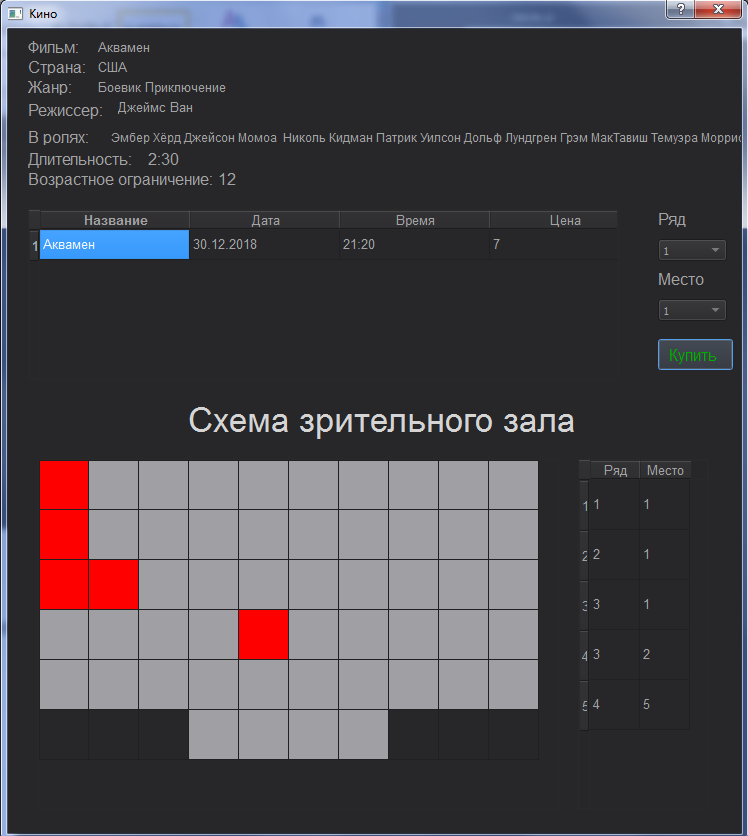


Рисунок 5.2 – Форма покупки билета

Для авторизации необходимо ввести логин и пароль, как проиллюстрировано на рисунке 5.3, после чего открываются возможности добавлять и редактировать данные.

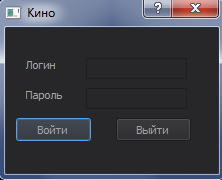


Рисунок 5.3 – Авторизация пользователя

Для манипуляции данными администратором нажмите на главной форме кнопку «Редактировать», после чего откроется форма, продемонстрированная на рисунке 5.4, необходимый пункт и заполните пустые поля данными для добавления.

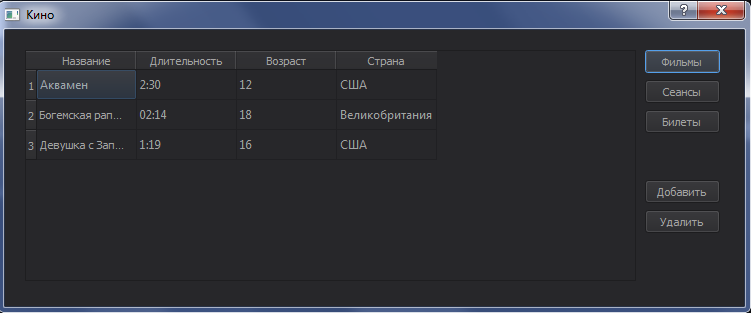


Рисунок 5.4 – Редактирование данных

Чтобы удалить данные из базы данных, необходимо выделить любое поле и нажать кнопку "Удалить".

Для добавления фильма или сеанса нажмите на кнопку «Добавить», рисунок 5.5.

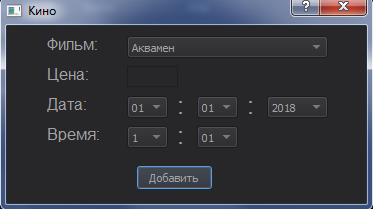


Рисунок 5.5 – Добавление сеанса

**ЗАКЛЮЧЕНИЕ**

Целью курсового проекта является разработка автоматизированной системы продажи билетов на сеансы в кино.

Результатом выполнения курсового проекта стало разработанное приложение, позволяющее автоматизировать операции сотрудника кинотеатра и увеличить эффективность работы персонала.

Во время создания программного средства были изучены и укреплены навыки работы с языком С++, СУБД Microsoft SQL Server и средой разработки Qt. Также были разобраны основы системного программирования и основы объектно-ориентированного подхода к программированию.

Основные преимущества курсового проекта:

* реализована возможность брони билетов;
* существует два режима работы: пользователь, администратор;
* возможность редактировать информацию;
* простой интерфейс;
* высокая надежность автоматизированной системы.

В процессе решения поставленных задач курсовой работы использовались прикладные системы программирования и необходимые методы решения заданий.

По окончанию курсового проекта можно сказать, что были изучены и закреплены следующие задачи:

* ставить задачу и разрабатывать алгоритм её решения;
* использовать прикладные системы программирования;
* разрабатывать основные программные документы;
* работать с современными системами программирования, включая объектно-ориентированные.

При разработке программного средства, поставленная задача выполнена полностью.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Голицина, О. Л., Попов, И. И. Microsoft SQL Server 2008. Основы T-SQL: Учебное пособие / Голицина О. Л. – М.: ФОРУМ: ИНФРА, 2005. – 340 c.
2. Нортроп Тони, Уилдермьюс Шон, Райан Билл. Основы разработки приложений на платформе Microsoft .Net Framework. Учебный курс Microsoft / Пер. с англ. – М.: «Русская редакция», СПб.: «Питер», 2007. – 514 c.
3. Павловская, Т. А. C. Программирование на языке высокого уровня: Учебник для вузов / Павловская Т. А. – СПб.: БХВ-Петербург. 2007. – 245 c.
4. Павловская, Т. А. С/С++ Программирование на языке высокого уровня / Павловская Т. А. – СПб.: Питер, 2005. – 261 c.
5. Попов, И. И., Партыка Т. Л. Языки программирования: Учебное пособие/ Попов И. И., Партыка Т. Л. – М.: ФОРУМ: ИНФРА-М, 2008. – 452 c.
6. Ржеуцкая, С.Ю. Базы данных. Язык SQL / Ржеуцкая С.Ю – М.: Эксмо, 2008. – 251 c.

**ПРИЛОЖЕНИЕ А**

**Текст программы**

#ifndef MAINWINDOW\_H //maiwindow.h

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QSqlDatabase>

#include "movie.h"

#include <QtSql>

namespace Ui {class MainWindow;}

class QSqlTableModel;

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = nullptr);

QSqlDatabase mDatabase;

QSqlQueryModel \*setquery = new QSqlQueryModel;

QString question;

~*MainWindow*();

signals:

void sendData(QString str);

private slots:

void onButtonSend();

private slots:

void on\_pushButton\_clicked();

void on\_pushButton\_2\_clicked();

void on\_pushButton\_3\_clicked();

void on\_tableView\_doubleClicked(const QModelIndex &index);

void on\_pushButton\_6\_clicked();

void on\_pushButton\_5\_clicked();

void on\_pushButton\_7\_clicked();

private:

Ui::MainWindow \*ui;

QSqlTableModel \*mModel;

Movie \*moive;

};

#endif

#ifndef EDITING\_H

#define EDITING\_H

#include <QDialog>

namespace Ui {class editing;}

class editing : public QDialog

{

Q\_OBJECT

public:

explicit editing(QWidget \*parent = nullptr);

~*editing*();

private slots:

void on\_pushButton\_clicked();

void on\_pushButton\_2\_clicked();

void on\_pushButton\_3\_clicked();

void on\_pushButton\_4\_clicked();

void on\_pushButton\_5\_clicked();

private:

Ui::editing \*ui;

};

#endif

#ifndef ADDFILM\_H//addfilm.h

#define ADDFILM\_H

#include <QDialog>

#include <QtSql>

namespace Ui {class addfilm;}

class addfilm : public QDialog

{

Q\_OBJECT

public:

explicit addfilm(QWidget \*parent = nullptr);

QString question;

QSqlQueryModel \*setquery = new QSqlQueryModel;

~*addfilm*();

private slots:

void on\_pushButton\_clicked();

private:

Ui::addfilm \*ui;

};

#endif

#ifndef ADDSCHEDULE\_H//addschedule.h

#define ADDSCHEDULE\_H

#include <QDialog>

#include <QtSql>

namespace Ui {class addschedule;}

class addschedule : public QDialog

{

Q\_OBJECT

public:

explicit addschedule(QWidget \*parent = nullptr);

QSqlQueryModel \*setquery = new QSqlQueryModel;

QString question;

~*addschedule*();

private slots:

void on\_pushButton\_4\_clicked();

private:

Ui::addschedule \*ui;

};

#endif

#ifndef MOVIE\_H // movie.h

#define MOVIE\_H

#include <QDialog>

#include <QtSql>

namespace Ui {class Movie;}

class Movie : public QDialog

{

Q\_OBJECT

public slots:

void recieveData(QString str);

public:

explicit Movie(QWidget \*parent = nullptr);

~*Movie*();

QSqlQueryModel \*setquery = new QSqlQueryModel;

QString question;

private slots:

void on\_tableView\_clicked(const QModelIndex &index);

void on\_pushButton\_clicked();

private:

Ui::Movie \*ui;

};

#endif

#include "addfilm.h"

#include "ui\_addfilm.h"

#include <QSqlTableModel>

#include <QSqlError>

#include <QMessageBox>

#include <QCoreApplication>

#include <QtSql>

#include <QUrl>

#include <QWidget>

#include <QFileDialog>

#include <QDir>

#include <QGraphicsDropShadowEffect>

#include <QStandardItem>

#include <QDialog>

#include "mainwindow.h"

addfilm::addfilm(QWidget \*parent) :

QDialog(parent),

ui(new Ui::addfilm)

{

ui->setupUi(this);

}

addfilm::~*addfilm*()

{

delete ui;

}

QString find\_key\_3 (QString question)

{

QSqlQuery query;

query.exec(question);

if ( ! query.isActive() )

{}

QString key="";

while (query.next())

key = query.value(0).toString();

return key;

}

void addfilm::on\_pushButton\_clicked()

{

if ((ui->textEdit->toPlainText())=="" && (ui->textEdit\_2->toPlainText())=="" && (ui->textEdit\_3->toPlainText())==""&& (ui->textEdit\_4->toPlainText())==""&& (ui->textEdit\_5->toPlainText())==""&& (ui->textEdit\_6->toPlainText())=="") QMessageBox::about(this,"Сообщение","Не все поля заполнены !");

else

{

QString fkey ="";

QString country="";

QString age = "";

QString film\_name = ui->textEdit->toPlainText();

question = "SELECT Код FROM Фильм where Название = '"+film\_name+"'";

fkey = find\_key\_3 ( question);

if (fkey != ""){QMessageBox::warning(this, "Ошибка","Данный фильм уже есть");}

else

{

question = "SELECT Страна.Код FROM Страна where Страна = '"+ui->textEdit\_2->toPlainText()+"'";

country = find\_key\_3 ( question);

if ( country != "")

{

question = "SELECT Страна.Код FROM Страна where Страна.Страна = '"+ui->textEdit\_2->toPlainText()+"'";

country=find\_key\_3(question);

}

else

{

setquery->setQuery("insert into Страна(Страна) values('" + ui->textEdit\_2->toPlainText() + "')");

country = find\_key\_3(question);

}

QString temp = ui->comboBox->currentText();

question = "SELECT Возраст.Код FROM Возраст where Возраст.Возраст = '"+temp+"'";

age = find\_key\_3(question);

QString time = ui->comboBox\_3->currentText()+":"+ui->comboBox\_2->currentText();

setquery->setQuery("insert into Фильм(Название,Длительность,Код\_страна,Код\_возраст,Описание) values('" + ui->textEdit->toPlainText() + "','" + time + "','" + country + "','" + age + "','" + ui->textEdit\_6->toPlainText() + "')");

question = "SELECT Код FROM Фильм where Название = '"+ ui->textEdit->toPlainText()+"'";

fkey = find\_key\_3(question);

QString key= ui->textEdit\_3->toPlainText();

QStringList list = key.split(',');

for (int i=0;i<list.count();i++)

{

QString addkey ="";

question = "SELECT Жанр.Код FROM Жанр where Жанр = '"+list[i]+"'";

addkey = find\_key\_3(question);

if (addkey=="")

{

setquery->setQuery("insert into Жанр(Жанр) values('" + list[i] + "')");

addkey = find\_key\_3(question);

setquery->setQuery("insert into Жанры(Код\_фильма,Код\_формата) values('" + fkey + "','" + addkey + "')");

}

else

{

setquery->setQuery("insert into Жанры(Код\_фильма,Код\_формата) values('" + fkey + "','" + addkey + "')");

}

}

key= ui->textEdit\_4->toPlainText();

list = key.split(',');

for (int i=0;i<list.count();i++)

{

QString addkey ="";

question = "SELECT Режиссер.Код FROM Режиссер where Имя = '"+list[i]+"'";

addkey = find\_key\_3(question);

if (addkey=="")

{

setquery->setQuery("insert into Режиссер(Имя) values('" + list[i] + "')");

addkey = find\_key\_3(question);

setquery->setQuery("insert into Режиссеры(Код\_фильма,Код\_режиссера) values('" + fkey + "','" + addkey + "')");

}

else

{

setquery->setQuery("insert into Режиссеры(Код\_фильма,Код\_режиссера) values('" + fkey + "','" + addkey + "')");

}

}

key= ui->textEdit\_5->toPlainText();

list = key.split(',');

for (int i=0;i<list.count();i++)

{

QString addkey ="";

question = "SELECT Актер.Код FROM Актер where Имя = '"+list[i]+"'";

addkey = find\_key\_3(question);

if (addkey=="")

{

setquery->setQuery("insert into Актер(Имя) values('" + list[i] + "')");

addkey = find\_key\_3(question);

setquery->setQuery("insert into Актеры(Код\_фильма,Код\_актера) values('" + fkey + "','" + addkey + "')");

}

else

{

setquery->setQuery("insert into Актеры(Код\_фильма,Код\_актера) values('" + fkey + "','" + addkey + "')");

}

}

QMessageBox::about(this,"Сообщение","Фильм добавлен");

}

}

}

#include "addschedule.h"

#include "ui\_addschedule.h"

#include <QSqlTableModel>

#include <QSqlError>

#include <QMessageBox>

#include <QCoreApplication>

#include <QtSql>

#include <QUrl>

#include <QWidget>

#include <QFileDialog>

#include <QDir>

#include <QGraphicsDropShadowEffect>

#include <QStandardItem>

#include <QDialog>

void query\_3 (QString question,QSqlQueryModel \*setquery)

{

setquery->setQuery(question);

}

addschedule::addschedule(QWidget \*parent): QDialog(parent), ui(new Ui::addschedule)

{

ui->setupUi(this);

question = "SELECT Название FROM Фильм";

query\_3 ( question,setquery);

ui->tableView->*setModel*(setquery);

for (int i=0; i<ui->tableView->model()->*rowCount*();i++)

ui->comboBox\_6->addItem ( ui->tableView->model()->*data*((ui->tableView->model()->*index*(i,0))).toString());

}

addschedule::~*addschedule*()

{

delete ui;

}

void addschedule::on\_pushButton\_4\_clicked()

{

if (ui->textEdit->toPlainText()=="")

{

QMessageBox::about(this,"Сообщение","Не все поля заполнены !");

}

else

{

QString strings = ui->comboBox\_6->currentText();

question = "SELECT Код FROM Фильм where Название = '" + strings + "'";

query\_3 ( question,setquery);

ui->tableView->*setModel*(setquery);

QString key\_film = ui->tableView->model()->*data*((ui->tableView->model()->*index*(0,0))).toString();

QString date = ui->comboBox\_2->currentText()+"."+ui->comboBox->currentText()+"."+ui->comboBox\_3->currentText();

QString time = ui->comboBox\_5->currentText()+":"+ui->comboBox\_4->currentText();

QString text = ui->textEdit->toPlainText();

QRegExp rx( "([a-zA-ZА-Яа-я#\_.]+)" );

QString check="";

int pos =0;

while ((pos = rx.indexIn(text, pos)) != -1)

{

check += rx.cap(1);

pos += rx.matchedLength();

}

if (check == "" || check == " " )

{

question = "insert into Сеансы(Дата,Время,Код\_фильма,Цена) values('" + date + "','" + time + "','" + key\_film + "','" + ui->textEdit->toPlainText() + "')";

query\_3 ( question,setquery);

QMessageBox::about(this,"Сообщение","Операция прошла успешно");

}

else

{

QMessageBox::critical(this,"Error","Введите цену в формате :'2,3'");

}

}

}

#include "authorization.h"

#include "ui\_authorization.h"

#include <QSqlTableModel>

#include <QSqlError>

#include <QMessageBox>

#include <QCoreApplication>

#include <QtSql>

#include <QUrl>

#include <QWidget>

#include <QFileDialog>

#include <QDir>

#include <QGraphicsDropShadowEffect>

#include <QStandardItem>

#include <QDialog>

Authorization::Authorization(QWidget \*parent): QDialog(parent),ui(new Ui::Authorization)

{

ui->setupUi(this);

}

Authorization::~*Authorization*()

{

delete ui;

}

QStringList fun\_file\_enter ( QString filename)

{

QFile file(filename);

QStringList strings;

if (file.*open*(QIODevice::ReadOnly | QIODevice::Text))

{

QTextStream in(&file);

while (!in.atEnd()) {

strings += in.readLine().split(";");

}

}

return strings;

}

void fun\_file\_in\_out (QString filename, bool check)

{

QFile file( filename );

if ( file.*open*(QIODevice::ReadWrite) )

{

QTextStream stream( &file );

if (check == false )

stream << "user" << endl;

else

stream << "admin" << endl;

}

}

void Authorization::on\_pushButton\_clicked()

{

QString filename = "enter.txt";

QStringList strings = fun\_file\_enter(filename);

filename="who.txt";

bool check = false;

if (strings[0]==ui->textEdit->toPlainText()&&strings[1]==ui->textEdit\_2->toPlainText())

{

check=true;

fun\_file\_in\_out( filename,check);

QMessageBox::about(this,"Сообщение","Вы вошли");

}

else

{

QMessageBox::about(this,"Сообщение","Неверный логин или пароль");

}

}

void Authorization::on\_pushButton\_2\_clicked()

{

QString filename = "who.txt";

bool check = false;

fun\_file\_in\_out( filename,check);

QMessageBox::about(this,"Сообщение","Вы вошли как пользователь");

}

#include "editing.h"

#include "ui\_editing.h"

#include "mainwindow.h"

#include <QSqlTableModel>

#include <QSqlError>

#include <QMessageBox>

#include <QCoreApplication>

#include <QtSql>

#include <QUrl>

#include <QWidget>

#include <QFileDialog>

#include <QDir>

#include <QGraphicsDropShadowEffect>

#include <QStandardItem>

#include "addfilm.h"

#include "addschedule.h"

int choose = 1;

editing::editing(QWidget \*parent):QDialog(parent), ui(new Ui::editing)

{

ui->setupUi(this);

QSqlQueryModel \*setquery1 = new QSqlQueryModel;

setquery1->setQuery("SELECT Название,Длительность,Возраст,Страна FROM Фильм,Возраст,Страна where Фильм.Код\_возраст=Возраст.Код and Фильм.Код\_страна=Страна.Код");

ui->tableView->*setModel*(setquery1);

}

editing::~*editing*()

{

delete ui;

}

QString find\_key (QString question)

{

QSqlQuery query;

query.exec(question);

if ( ! query.isActive() )

{}

QString key="";

while (query.next())

key = query.value(0).toString();

return key;

}

void editing::on\_pushButton\_clicked()

{

choose = 1;

QSqlQueryModel \*setquery1 = new QSqlQueryModel;

setquery1->setQuery("SELECT Название,Длительность,Возраст,Страна FROM Фильм,Возраст,Страна where Фильм.Код\_возраст=Возраст.Код and Фильм.Код\_страна=Страна.Код");

ui->tableView->*setModel*(setquery1);

}

void editing::on\_pushButton\_2\_clicked()

{

choose = 2;

QSqlQueryModel \*setquery1 = new QSqlQueryModel;

setquery1->setQuery("SELECT DISTINCT Название, Дата, Время,Сеансы.Код FROM Фильм,Сеансы,Билеты where Фильм.Код = Сеансы.Код\_фильма");

ui->tableView->*setModel*(setquery1);

}

void editing::on\_pushButton\_3\_clicked()

{

choose = 3;

QSqlQueryModel \*setquery1 = new QSqlQueryModel;

setquery1->setQuery("SELECT DISTINCT Название, Дата, Время,Ряд,Место FROM Место,Ряд,Билеты,Сеансы,Фильм where Фильм.Код = Сеансы.Код\_фильма and Сеансы.Код = Билеты.Код\_сеанс and Билеты.Код\_место = Место.Код and Билеты.Код\_ряд=Ряд.Код");

ui->tableView->*setModel*(setquery1);

}

void editing::on\_pushButton\_4\_clicked()

{

switch (choose)

{

case 1:

{

addfilm enter;

enter.setModal(true);

enter.*exec*();

}

break;

case 2:

{

addschedule enter;

enter.setModal(true);

enter.*exec*();

}

break;

case 3:

QMessageBox::about(this,"Сообщение","Для выполнения этой операции перейдите на соответствующую форму");

break;

}

}

void editing::on\_pushButton\_5\_clicked()

{

QModelIndex currentDiscount = ui->tableView->currentIndex();

QString question;

QSqlQueryModel \*setquery4 = new QSqlQueryModel;

switch (choose)

{

case 1:

{

question = "SELECT Код FROM Фильм where Название = '" + (ui->tableView->model()->*data*(ui->tableView->model()->*index*(currentDiscount.row(),0),0)).toString() + "'";

QString key\_film = find\_key(question);

question = "SELECT Сеансы.Код FROM Фильм,Сеансы where Название = '" + (ui->tableView->model()->*data*(ui->tableView->model()->*index*(currentDiscount.row(),0),0)).toString() + "' and Фильм.Код = Сеансы.Код\_фильма";

QString key\_seans = find\_key(question);

setquery4->setQuery("delete from Билеты where Билеты.Код\_сеанс = " + key\_seans +"");

setquery4->setQuery("delete from Сеансы where Сеансы.Код = " + key\_seans +"");

setquery4->setQuery("delete from Актеры where Актеры.Код\_фильма = " +key\_film +"");

setquery4->setQuery("delete from Жанры where Жанры.Код\_фильма =" + key\_film +"");

setquery4->setQuery("delete from Режиссеры where Режиссеры.Код\_фильма = " + key\_film +"");

setquery4->setQuery("delete from Фильм where Название = '" + (ui->tableView->model()->*data*(ui->tableView->model()->*index*(currentDiscount.row(),0),0)).toString() +"'");

}

break;

case 2:

{

question = "SELECT Код FROM Фильм where Название = '" + (ui->tableView->model()->*data*(ui->tableView->model()->*index*(currentDiscount.row(),0),0)).toString() + "'";

QString key\_film = find\_key(question);

question = "SELECT Сеансы.Код FROM Фильм,Сеансы where Название = '" + (ui->tableView->model()->*data*(ui->tableView->model()->*index*(currentDiscount.row(),0),0)).toString() + "' and Фильм.Код = Сеансы.Код\_фильма";

QString key\_seans = find\_key(question);

setquery4->setQuery("delete from Билеты where Билеты.Код\_сеанс = " + key\_seans +"");

setquery4->setQuery("delete from Сеансы where Сеансы.Код = " + key\_seans +"");

}

break;

case 3:

{

question = "SELECT Код FROM Фильм where Название = '" + (ui->tableView->model()->*data*(ui->tableView->model()->*index*(currentDiscount.row(),0),0)).toString() + "'";

QString key\_film = find\_key(question);

question = "SELECT Сеансы.Код FROM Фильм,Сеансы where Название = '" + (ui->tableView->model()->*data*(ui->tableView->model()->*index*(currentDiscount.row(),0),0)).toString() + "' and Фильм.Код = Сеансы.Код\_фильма";

QString key\_seans = find\_key(question);

QString col = (ui->tableView->model()->*data*(ui->tableView->model()->*index*(currentDiscount.row(),4),0)).toString();

QString row = (ui->tableView->model()->*data*(ui->tableView->model()->*index*(currentDiscount.row(),3),0)).toString();

question = "SELECT Билеты.Код FROM Билеты where Билеты.Код\_сеанс = " + key\_seans + " and Билеты.Код\_место " + (ui->tableView->model()->*data*(ui->tableView->model()->*index*(currentDiscount.row(),0),4)).toString() +"and Билеты.Код\_ряд = " + (ui->tableView->model()->*data*(ui->tableView->model()->*index*(currentDiscount.row(),3),0)).toString() +" ";

QString key\_tic = find\_key(question);

setquery4->setQuery("delete from Билеты where Билеты.Код\_сеанс = " + key\_seans + " and Билеты.Код\_место " + (ui->tableView->model()->*data*(ui->tableView->model()->*index*(currentDiscount.row(),0),4)).toString() +"and Билеты.Код\_ряд = " + (ui->tableView->model()->*data*(ui->tableView->model()->*index*(currentDiscount.row(),3),0)).toString() +"");

}

break;

}

}

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

a.setStyle("fusion");

MainWindow w;

w.show();

return a.exec();

}

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "authorization.h"

#include "movie.h"

#include "editing.h"

#include <QSqlTableModel>

#include <QSqlError>

#include <QMessageBox>

#include <QCoreApplication>

#include <QtSql>

#include <QUrl>

#include <QWidget>

#include <QFileDialog>

#include <QDir>

#include <QGraphicsDropShadowEffect>

#define ACCESS "DRIVER={Microsoft Access Driver (\*.mdb)};FIL={MS Access};DBQ=DB.mdb"

void query (QString question,QSqlQueryModel \*setquery)

{

setquery->setQuery(question);

}

MainWindow::MainWindow(QWidget \*parent):QMainWindow(parent), ui(new Ui::MainWindow)

{

ui->setupUi(this);

moive = new Movie();

connect(ui->pushButton\_4, SIGNAL(clicked()), moive, SLOT(show()));

connect(ui->pushButton\_4, SIGNAL(clicked()), this, SLOT(onButtonSend()));

connect(this, SIGNAL(sendData(QString)),moive, SLOT(recieveData(QString)));

this->setWindowFlags(Qt::FramelessWindowHint);

this->setAttribute(Qt::WA\_TranslucentBackground);

QGraphicsDropShadowEffect \*shadowEffect = new QGraphicsDropShadowEffect(this);

shadowEffect->setBlurRadius(9);

shadowEffect->setOffset(0);

mDatabase = QSqlDatabase::addDatabase("QODBC");

mDatabase.setDatabaseName(ACCESS);

if (!mDatabase.open())

{

QMessageBox::critical(this,"Error",mDatabase.lastError().text());

return;

}

question = "SELECT Название,Длительность,Возраст,Страна FROM Фильм,Возраст,Страна where Фильм.Код\_возраст=Возраст.Код and Фильм.Код\_страна=Страна.Код";

query (question,setquery);

ui->tableView->*setModel*(setquery);

}

void MainWindow::onButtonSend()

{

QStringList list;

QModelIndex currentDiscount = ui->tableView->currentIndex();

QString temp = (ui->tableView->model()->*data*(ui->tableView->model()->*index*(currentDiscount.row(),0),0)).toString();

if (temp==""){QMessageBox::warning(this, tr("Database Error"),"Выберите фильм"); moive ->close();}

else {emit sendData(temp);}

}

MainWindow::~*MainWindow*()

{

delete ui;

}

void MainWindow::on\_pushButton\_clicked()

{

Authorization enter;

enter.setModal(true);

enter.*exec*();

}

void MainWindow::on\_pushButton\_2\_clicked()

{

if (ui->label\_2->text()=="user")

{

QMessageBox::about(this,"Сообщение","Необходимо зайти в систему");

}

else

{

editing enter;

enter.setModal(true);

enter.*exec*();

}

}

void MainWindow::on\_pushButton\_3\_clicked()

{

qApp->closeAllWindows();

qApp->quit();

}

void MainWindow::on\_tableView\_doubleClicked(const QModelIndex &index)

{

}

void MainWindow::on\_pushButton\_6\_clicked()

{

question = ("SELECT Название,Длительность,Возраст,Страна FROM Фильм,Возраст,Страна where Фильм.Код\_возраст=Возраст.Код and Фильм.Код\_страна=Страна.Код");

query (question,setquery);

ui->tableView->*setModel*(setquery);

}

void MainWindow::on\_pushButton\_5\_clicked()

{

QString date =ui->comboBox\_2->currentText()+'.'+ui->comboBox->currentText()+'.'+ui->comboBox\_3->currentText();

question = ("SELECT Название,Длительность,Возраст,Страна FROM Фильм,Возраст,Страна,Сеансы where Фильм.Код\_возраст=Возраст.Код and Фильм.Код\_страна=Страна.Код and Фильм.Код=Сеансы.Код\_фильма and Сеансы.Дата = '" + date + "'");

query (question,setquery);

ui->tableView->*setModel*(setquery);

}

void MainWindow::on\_pushButton\_7\_clicked()

{

QFile file("who.txt");

QString strings;

if (file.*open*(QIODevice::ReadOnly | QIODevice::Text))

{

QTextStream in(&file);

while (!in.atEnd())

{

strings += in.readLine();

}

}

ui->label\_2->setText(strings);

question = "SELECT Название,Длительность,Возраст,Страна FROM Фильм,Возраст,Страна where Фильм.Код\_возраст=Возраст.Код and Фильм.Код\_страна=Страна.Код";

query (question,setquery);

ui->tableView->*setModel*(setquery);

}

#include "movie.h"

#include "ui\_movie.h"

#include <QSqlTableModel>

#include <QSqlError>

#include <QMessageBox>

#include <QCoreApplication>

#include <QtSql>

#include <QUrl>

#include <QWidget>

#include <QFileDialog>

#include <QDir>

#include <QGraphicsDropShadowEffect>

#include <QStandardItem>

void query\_2 (QString question,QSqlQueryModel \*setquery)

{

setquery->setQuery(question);

}

Movie::Movie(QWidget \*parent):QDialog(parent),ui(new Ui::Movie)

{

ui->setupUi(this);

}

void Movie::recieveData(QString str)

{

ui->label->setText(str);

question = "SELECT DISTINCT Длительность,Возраст,Страна FROM Фильм,Возраст,Страна where Фильм.Код\_возраст=Возраст.Код and Фильм.Код\_страна=Страна.Код and Фильм.Название='"+ui->label->text()+"' ";

query\_2 (question,setquery);

ui->tableView->*setModel*(setquery);

ui->label\_11->setText( ui->tableView->model()->*data*((ui->tableView->model()->*index*(0,2))).toString());

ui->label\_14->setText( ui->tableView->model()->*data*((ui->tableView->model()->*index*(0,1))).toString());

ui->label\_8->setText( ui->tableView->model()->*data*((ui->tableView->model()->*index*(0,0))).toString());

question = "SELECT Жанр FROM Фильм,Жанр,Жанры where Фильм.Название='"+ui->label->text()+"' and Фильм.Код=Жанры.Код\_фильма and Жанры.Код\_формата = Жанр.Код";

query\_2 (question,setquery);

ui->tableView->*setModel*(setquery);

QString temp;

for (int i=0 ; i< ui->tableView->model()->*rowCount*();i++)

{

temp+=ui->tableView->model()->*data*((ui->tableView->model()->*index*(i,0))).toString();

temp+=' ';

}

ui->label\_7->setText( temp );

question = "SELECT Имя FROM Фильм,Актер,Актеры where Фильм.Название='"+ui->label->text()+"' and Фильм.Код=Актеры.Код\_фильма and Актеры.Код\_актера = Актер.Код";

query\_2 (question,setquery);

ui->tableView->*setModel*(setquery);

temp="";

for (int i=0 ; i< ui->tableView->model()->*rowCount*();i++)

{

temp+=ui->tableView->model()->*data*((ui->tableView->model()->*index*(i,0))).toString();

temp+=' ';

}

ui->label\_10->setText( temp );

question = "SELECT Имя FROM Фильм,Режиссер,Режиссеры where Фильм.Название='"+ui->label->text()+"' and Фильм.Код=Режиссеры.Код\_фильма and Режиссеры.Код\_режиссера = Режиссер.Код";

query\_2 (question,setquery);

ui->tableView->*setModel*(setquery);

temp="";

for (int i=0 ; i< ui->tableView->model()->*rowCount*();i++)

{

temp+=ui->tableView->model()->*data*((ui->tableView->model()->*index*(i,0))).toString();

temp+=' ';

}

ui->label\_12->setText( temp );

question = "SELECT DISTINCT Название, Дата, Время,Цена,Сеансы.Код FROM Фильм,Сеансы,Билеты where Фильм.Название='"+ui->label->text()+"' and Фильм.Код=Сеансы.Код\_фильма";

query\_2 (question,setquery);

ui->tableView->*setModel*(setquery);

QStandardItemModel \*model = new QStandardItemModel;

QStandardItem \*item;

for (int i=0; i<6;i++)

for (int j=0; j<10;j++)

{

item = new QStandardItem();

model->setItem(i,j,item);

}

ui->tableView\_2->*setModel*(model);

ui->tableView\_2->resizeRowsToContents();

ui->tableView\_2->resizeColumnsToContents();

for (int i=0; i<5;i++)

for (int j=0; j<10;j++)

{

ui->tableView\_2->model()->*setData*(ui->tableView\_2->model()->*index*(i,j),QColor(Qt::gray),Qt::BackgroundColorRole);

}

for (int temp2=3;temp2<7;temp2++)

{

ui->tableView\_2->model()->*setData*(ui->tableView\_2->model()->*index*(5,temp2),QColor(Qt::gray),Qt::BackgroundColorRole);

}

}

Movie::~*Movie*()

{

delete ui;

}

void Movie::on\_tableView\_clicked(const QModelIndex &index)

{

for (int i=0; i<5;i++)

for (int j=0; j<10;j++)

{

ui->tableView\_2->model()->*setData*(ui->tableView\_2->model()->*index*(i,j),QColor(Qt::gray),Qt::BackgroundColorRole);

}

for (int temp2=3;temp2<7;temp2++)

{

ui->tableView\_2->model()->*setData*(ui->tableView\_2->model()->*index*(5,temp2),QColor(Qt::gray),Qt::BackgroundColorRole);

}

QModelIndex currentDiscount = ui->tableView->currentIndex();

QString name = (ui->tableView->model()->*data*(ui->tableView->model()->*index*(currentDiscount.row(),0),0)).toString();

QString date = (ui->tableView->model()->*data*(ui->tableView->model()->*index*(currentDiscount.row(),1),0)).toString();

QString time = (ui->tableView->model()->*data*(ui->tableView->model()->*index*(currentDiscount.row(),2),0)).toString();

QSqlQueryModel \*setquery2 = new QSqlQueryModel;

setquery2->setQuery("SELECT DISTINCT Ряд,Место FROM Место,Ряд,Билеты,Сеансы,Фильм where Фильм.Название = '"+name+"'and Сеансы.Дата='"+date+"'and Сеансы.Время='"+time+"'and Билеты.Код\_место=Место.Код and Билеты.Код\_ряд=Ряд.Код and Сеансы.Код=Билеты.Код\_сеанс");

ui->tableView\_3->*setModel*(setquery2);

for (int i=0 ; i< ui->tableView\_3->model()->*rowCount*();i++)

{

int row =ui->tableView\_3->model()->*data*((ui->tableView\_3->model()->*index*(i,0))).toInt()-1;

int col =ui->tableView\_3->model()->*data*((ui->tableView\_3->model()->*index*(i,1))).toInt()-1;

ui->tableView\_2->model()->*setData*(ui->tableView\_2->model()->*index*(row,col),QColor(Qt::red),Qt::BackgroundColorRole);

}

}

void Movie::on\_pushButton\_clicked()

{

QModelIndex currentDiscount = ui->tableView->currentIndex();

if (ui->comboBox\_2->currentText().toInt()==6 && ui->comboBox->currentText().toInt()==1 || ui->comboBox\_2->currentText().toInt()==6 && ui->comboBox->currentText().toInt()==2 || ui->comboBox\_2->currentText().toInt()==6 && ui->comboBox->currentText().toInt()==3 || ui->comboBox\_2->currentText().toInt()==6 && ui->comboBox->currentText().toInt()==8 || ui->comboBox\_2->currentText().toInt()==6 && ui->comboBox->currentText().toInt()==9 ||ui->comboBox\_2->currentText().toInt()==6 && ui->comboBox->currentText().toInt()==10)

{

QMessageBox::critical(this,"Error","Такого места нет");

}

else

{

QSqlQueryModel \*setquery4 = new QSqlQueryModel;

bool chek = false;

for (int i =0 ; i<ui->tableView\_3->model()->*rowCount*();i++)

if ((ui->tableView\_3->model()->*data*((ui->tableView\_3->model()->*index*(i,0)))).toString()==ui->comboBox\_2->currentText()&&(ui->tableView\_3->model()->*data*((ui->tableView\_3->model()->*index*(i,1)))).toString()==ui->comboBox->currentText())

{

chek = true;

QMessageBox::about(this,"Сообщение","Билет уже куплен");

}

if (chek==false)

{

setquery4->setQuery("insert into Билеты(Код\_сеанс,Код\_место,Код\_ряд) values('" + (ui->tableView->model()->*data*(ui->tableView->model()->*index*(currentDiscount.row(),4),0)).toString() + "','" + ui->comboBox->currentText() + "','" + ui->comboBox\_2->currentText() + "')");

QMessageBox::about(this,"Сообщение","Билет куплен");

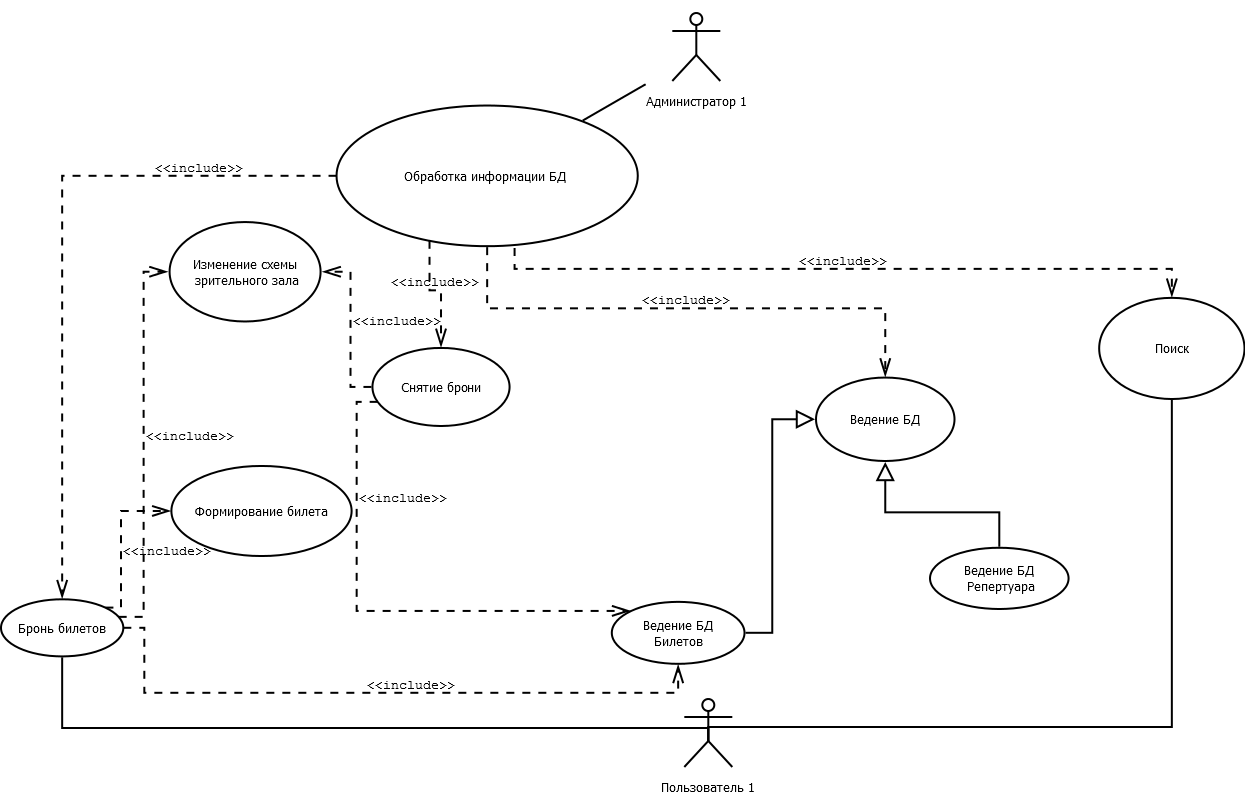
}

}

}

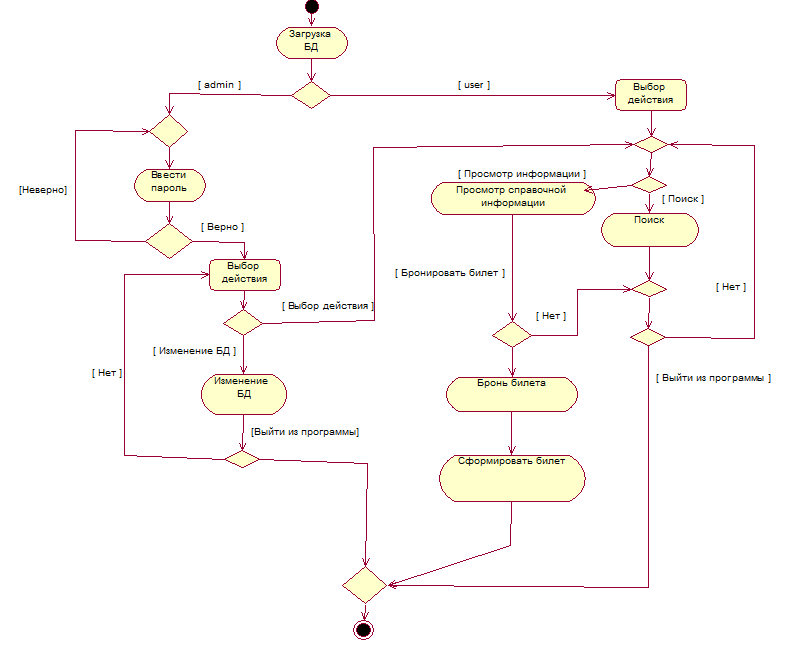
**ПРИЛОЖЕНИЕ Б**

**Диаграмма вариантов использования**



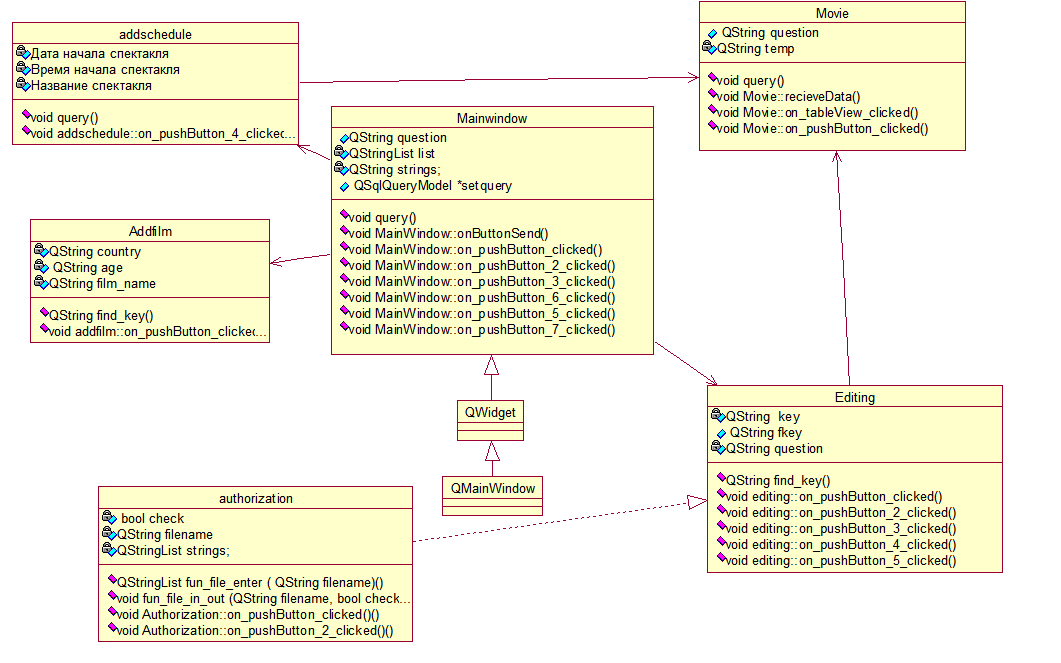
**ПРИЛОЖЕНИЕ В**

**Диаграмма деятельности**



**ПРИЛОЖЕНИЕ Г**

**Диаграмма классов**

****

**ПРИЛОЖЕНИЕ Д**

**Схема структурная**

Изм.

Лист

№ докум.

Подпись

Дата

Лист

Разраб.

27М

Провер.

Реценз.

Н. Контр.

Утверд.

Лит.

Листов