МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПЕУБЛИКИ БЕЛАРУСЬ

Учреждение образования   
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1-40 05 01 Информационные системы и технологии

Направление специальности 1-40 01 02 03 Информационные системы

и технологии (издательско-полиграфический комплекс)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«База данных «Кинотеатр» с применением резервного копирования и восстановления данных»

Выполнил студент Демидкевич А.С.

(Ф.И.О.)

Руководитель проекта ассистент Колмаков М.В.

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

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

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

Консультант: ассистент Колмаков М.В.

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

Нормоконтролер: ассистент Колмаков М.В.

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

Курсовой проект защищен с оценкой

Минск 2021

**Оглавление**

[Введение 3](#_Toc71096987)

[1 Постановка задачи 4](#_Toc71096988)

[2 Разработка модели базы данных 5](#_Toc71096989)

[3 Разработка необходимых объектов 6](#_Toc71096990)

[4 Описание процедур импорта и экспорта 9](#_Toc71096991)

[5 Тестирование производительности 12](#_Toc71096992)

[6 Описание технологии и ее применения в базе данных 13](#_Toc71096993)

[7 Руководство пользователя 14](#_Toc71096994)

[Заключение 25](#_Toc71096995)

[Cписок источников 26](#_Toc71096996)

[Приложение А 27](#_Toc71096997)

[Приложение Б 30](#_Toc71096998)

[Приложение В 32](#_Toc71096999)

# **Введение**

Любая организация нуждается в своевременном доступе к информации. Ценность информации в современном мире очень высока. Роль распорядителей информации в современном мире чаще всего выполняют базы данных. Базы данных обеспечивают надежное хранение информации, в структурированном виде и своевременный доступ к ней. Практически любая современная организация нуждается в базе данных, удовлетворяющей те или иные потребности по хранению, управлению и администрированию данных.

За последние несколько лет наблюдается тенденция к усложнению структур данных. Простые виды информации, представимой в форме чисел и текстовых строк, не утратив своей значимости, дополняются сегодня многочисленными мультимедийными документами, графическими образами, хронологическими рядами, процедурными, или активными, данными и мириадами прочих сложных информационных форм.

На сегодняшний день на рынке представлено множество технологий доступа к данным и серверов баз данных, каждое, из которых имеет свои отличительные черты. Современные приложения обработки данных ориентированы на работу с большим количеством пользователей, на их удаленность от места расположения основного сервера БД.

Темой данного курсового проекта является разработка программы «Кинотеатр».

Приложение «Кинотеатр» имеет доступ к таблицам в базе данных и предназначено для взаимодействия с источником данных. Взаимодействие подразумевает получение данных, их представление в определенном формате для просмотра пользователем, редактирование в соответствии с реализованными в программе бизнес-алгоритмами и возврат обработанных данных обратно в базу данных.

База данных «Кинотеатр» существенно упрощает работу сотрудников кинотеатра, а также предоставляет возможность администратору своевременно вносить необходимые изменения. Данный программный продукт позволяет автоматизировать работу кассы кинотеатра.

# **1 Постановка задачи**

Целью данной курсовой работы является создание программного обеспечения для автоматизации процесса администрирования сеансов кинотеатра и продажи билетов.

Задача проекта: совершенствование практических навыков в использовании и разработке современных информационных систем, разработка архитектуры приложения, Создание базы данных и интерфейса, выполнение тестирования готового продукта.

Функционально должны быть выполнены следующие задачи:

* разработка приложения для взаимодействия с БД;
* регистрация персонала;
* обновление списка фильмов;
* обновление списка сеансов;
* продажа и возврат билетов.

Должны быть выполнены следующие требования:

− доступ к данным должен осуществляться только через соответствующие процедуры;

− должен быть проведен импорт данных из XML файлов, экспорт данных в формат XML;

− необходимо протестировать производительность базы данных на таблице, содержащей не менее 100 000 строк, и внести изменения в структуру в случае необходимости. Необходимо проанализировать планы запросов к таблице;

− применить технологию базы данных согласно выбранной теме: подробно описать применяемые системные пакеты, утилиты или технологии; показать применение указанной технологии в базе данных.

Десктоп приложение должно быть реализовано с использованием языка С#. Технология для реализации – WPF. База данных приложения должно взаимодействовать с Oracle Database 12с.

Разрабатываемое приложение должно содержать форму авторизации. Следует разработать два уровня доступа: уровень доступа для администратора и уровень доступа для работников кассы. Администратор имеет возможность добавлять сотрудников, сеансы и фильмы.

# **2 Разработка модели базы данных**

Первым этапом курсовой работы будет создание логически взаимосвязанных таблиц. Чтобы составить визуальную взаимосвязанную структуры нашей базы данных, нам необходимо продумать, какая информация будет храниться в этих таблицах, после этого создать связи с помощью первичных и внешних ключей.

Диаграмма базы данных, спроектированной в ходе разработки приложения представлена на рисунке 2.1.

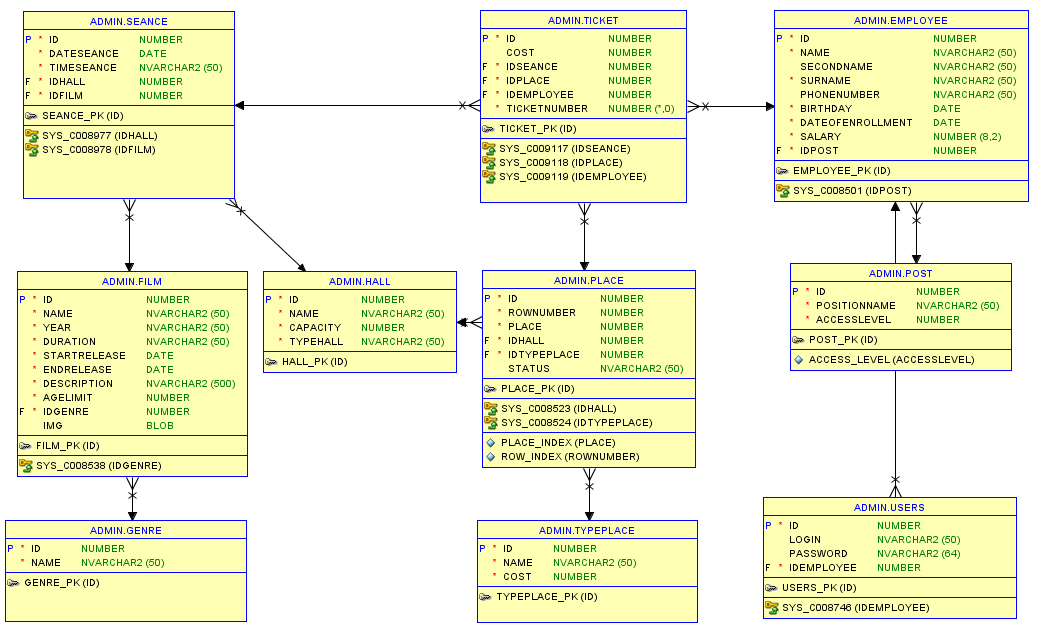


Рисунок 2.1 – Диаграмма базы данных

База данных состоит из следующих таблиц:

* Ticket;
* Seance;
* Film;
* Genre;
* Employee;
* Place;
* Hall;
* TypePlace;
* Post;
* User.

# **3 Разработка необходимых объектов**

В ходе проектирования базы данных было создано 10 таблиц, связанных между собой внешними ключами.

Таблица «Ticket» хранит список проданных билетов на данный момент на доступные сеансы.

Поля таблицы:

* ID – идентификатор билета, первичный ключ;
* Cost – стоимость билета;
* IDSeance – идентификатор сеанса, внешний ключ для связи с таблицей «Seance»;
* IDPlace – идентификатор места, внешний ключ для связи с таблицей «Place»;
* IDEmployee – идентификатор сотрудника, внешний ключ для связи с таблицей «Employee»;
* TicketNumber – номер билета.

Таблица «Employee» содержит список всех сотрудников, зарегистрированных в базе на данный момент.

Поля таблицы:

* ID – идентификатор сотрудника, первичный ключ.
* Name – имя сотрудника;
* SecondName – отчество;
* Surname – фамилия;
* PhoneNumber – номер телефона;
* Birthday – день рождения;
* DateOfEnrollment – дата зачисления;
* Salary – зарплата;
* IDPost – идентификатор должности, внешний ключ для связи с таблицей «Post».

Таблица «Post» содержит перечень должностей.

Поля таблицы:

* ID – идентификатор должности, первичный ключ;
* PositionName – название должности;
* AccessLevel – уровень доступа;

Таблица «User» содержит логин и пароль сотрудников, которые имеют доступ к системе.

Поля таблицы:

* ID – идентификатор аккаунта, первичный ключ;
* Login – логин сотрудника;
* Password – пароль сотрудника;
* IDEmployee – идентификатор сотрудника, внешний ключ для вязи с таблицей «Employee».

Таблица «Seance» содержит сеансы фильмов, которые сейчас в прокате в кинотеатре.

Поля таблицы:

* ID – идентификатор сеанса, первичный ключ;
* DateSeance – дата сеанса;
* TimeSeance – время сеанса;
* IDHall – идентификатор зала, внешний ключ для связи с таблицей «Hall»;
* IDFilm – идентификатор фильма, внешний ключ для связи с таблицей «Film».

Таблица «Film» содержит список всех фильмов, внесенных в базу.

Поля таблицы:

* ID – идентификатор фильма, первичный ключ;
* Name – название фильма;
* Year – год выпуска фильма;
* Duration – продолжительность фильма;
* StartRelease – дата начала проката фильма;
* Endrelease – дата окончания проката фильма;
* Description – описание фильма;
* AgeLimit – возрастное ограничение;
* IDGenre – идентификатор жанра, внешний ключ для связи с таблицей «Genre»;
* IMG – афиша фильма.

Таблица «Genre» содержит список всех жанров.

Поля таблицы:

* ID – идентификатор жанра, первичный ключ;
* Name – название жанра.

Таблица «Hall» включает список залов кинотеатра.

Поля таблицы:

* ID – идентификатор зала, первичный ключ;
* Name – название зала;
* Capacity – общее число мест в зале;
* TypeHall – тип зала.

Таблица «Place» содержит перечень всех мест в кинотеатре, распределенных по залам.

Поля таблицы:

* ID – идентификатор места, первичный ключ;
* RawNumber – номер ряда;
* Place – номер места в ряду;
* IDHall – идентификатор зала, внешний ключ для связи с таблицей «Hall»;
* IDTypePlace – идентификатор типа места, внешний ключ для связи с таблицей «Place»;
* Status – статус места: «Занято», «Свободно».

Таблица «TypePlace» включает в себя список всех типов мест.

Поля таблицы:

* ID – идентификатор типа места, первичный ключ;
* Name – название типа места;
* Cost – стоимость места данного типа.

Помимо таблиц в базе данных были созданы объекты: хранимые процедуры для выполнения основных действий с данными в базе данных, представления, последовательности, пользователи, планировщик задач.

В каждой из процедур, для избежания исключительных ситуаций, происходит обработка исключений с выводом сообщения об ошибке.

Для разграничения прав доступа было создана два пользователя. Один является администратором базы данных. От его имени создавались объекты базы данных. Администратор имеет возможность добавлять сотрудников, сеансы и фильмы. Второй пользователь имеет ограниченные права на выполнение процедур, которые ему выдал администратор.

Хранимая процедура «addEmployeeAndUser» предназначена для добавления новых сотрудников и регистрации их в системе. При создании новой учетной записи происходит шифрация пароля пользователя. Для удаления сотрудника из базы используется процедура «deleteEmployee».

После добавления сотрудника или сеанса, есть возможно редактировать данные. Для этого используются процедуры «updateEmployee» и «updateSeance» соответственно.

Для оформления продажи билета используется процедура «SellTicket», которая производит ряд проверок на возможность продажи данного билета. После продажи билета, доступ к месту, указанному в билете, будет закрыт. Также после покупки билета его можно вернуть, тем самым позволив продавать билеты на это место снова. За это отвечает процедура «deleteTicket».

Также имеется ряд однотипных процедур для добавления и удаления сеансов и фильмов. Для обеспечения удобного доступа к данным используются представления, которые хранят все необходимые поля и процедуры типа «Get».

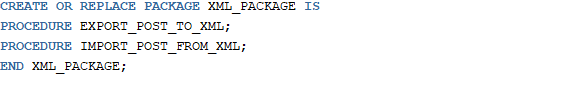
Для автоматической очистки списка фильмов, дата окончания проката которых уже прошла, используется планировщик задач, который выполняет процедуру «deleteFilmScheduler». Данный планировщик убирает из списка фильмов те, прокат которых завершился вчера. Обновление планировщика происходит ежеминутно.

Также была создана последовательность для нумерации проданных билетов, которая каждому проданному билету ставит номер на единицу больше предыдущего. Максимальное значение последовательности равно общему количеству билетов, которые можно продать за один день работы кинотеатра.

# **4 Описание процедур импорта и экспорта**

XML — это формат данных, используемый для обмена данными в форме, которая может быть легко использована и распространена. Часто возникает необходимость импортировать и экспортировать XML-файлы в Oracle.

В данном курсовом проекте используются пакеты DBMS\_XMLPARSER и DBMS\_XSLPROCESSOR для импорта XML, и DBMS\_XMLDOM для экспорта данных в XML формат. Для работы с файлами так же использовался пакет DBMS\_LOB. В данном курсовом проекте функции экспорта, импорта используются для таблицы Post. Пример создания процедуры экспорта можно посмотреть на рисунке 4.1.



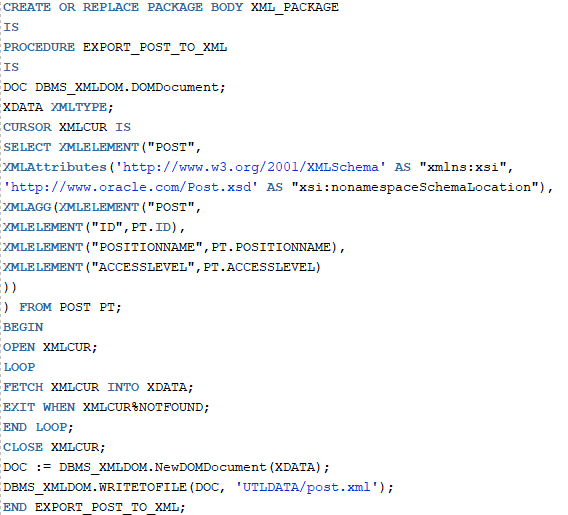
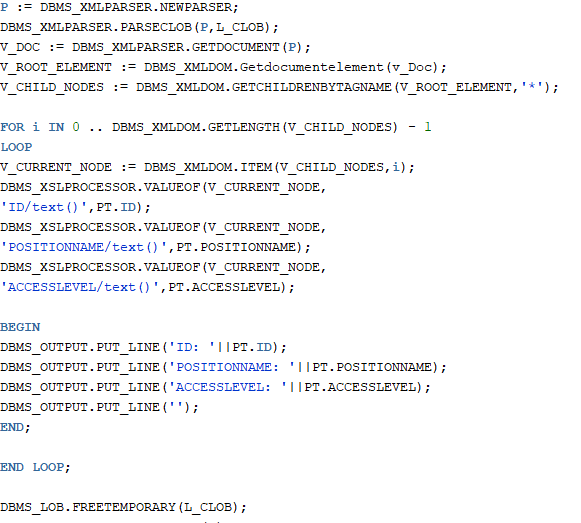
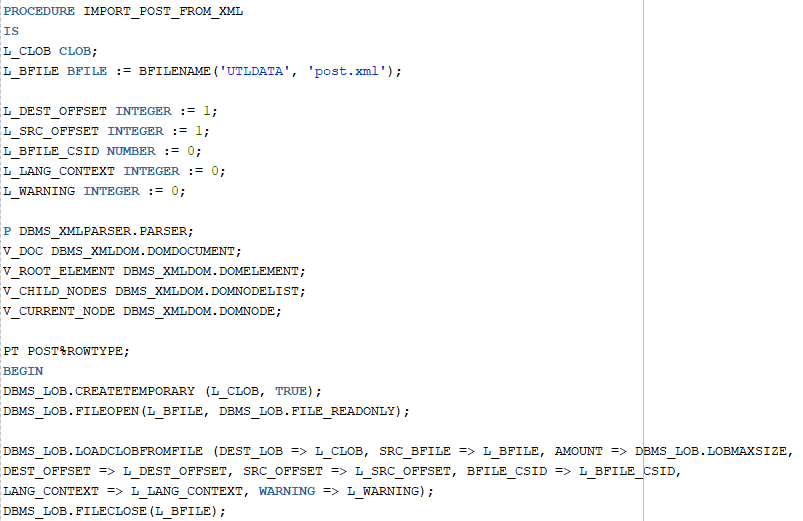
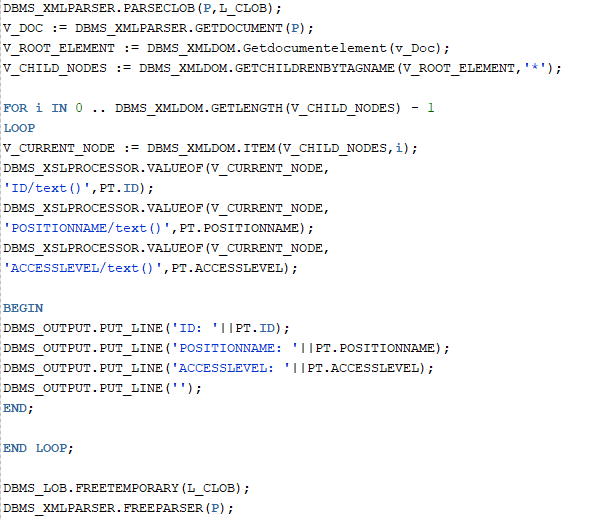


Рисунок 4.1 – Процедура экспорта в XML файл

Для импортирования данных из XML файла в базу данных используется процедура, код которой приведен на рисунке 4.2. Для импорта данных использовались пакеты DBMS\_XMLPARSER и DBMS\_XSLPROCESSOR.





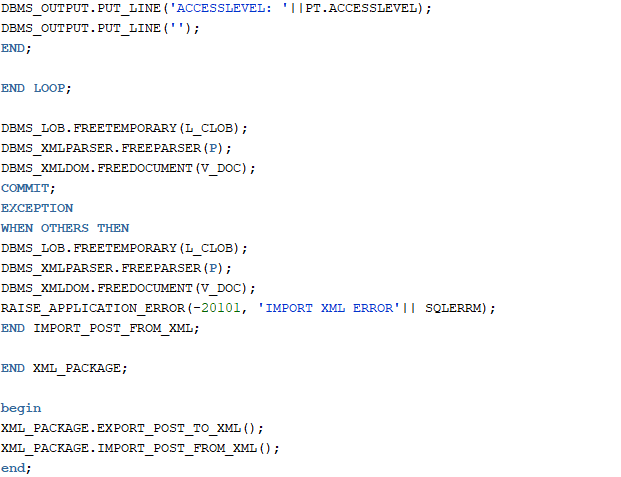


Рисунок 4.2 – Процедура импорта из XML файла

# **5 Тестирование производительности**

Для тестирования производительности разработанного приложения были проведены тесты использование памяти и использование ЦП.

Тестирование использование памяти проводится с целью обнаружить проблему утечки памяти.

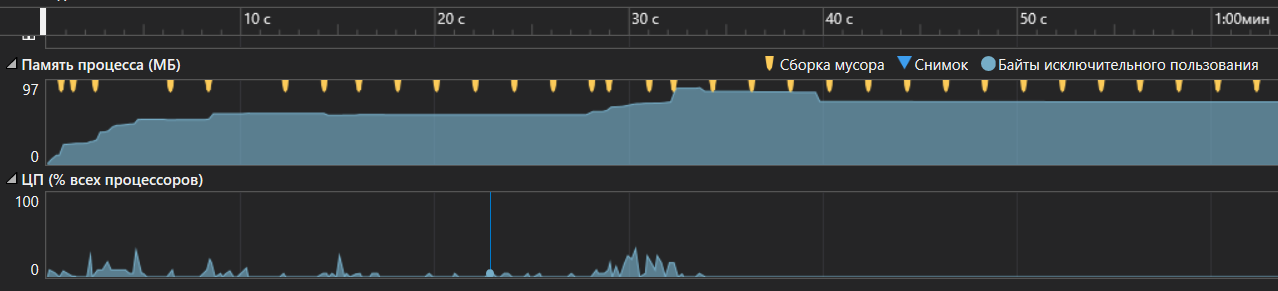
****

Рисунок 5.1 – Тестирование памяти

Как показано на рисунке 5.1 тестирование приложения на проблему утечку памяти пройдено успешно. Приложения в процессе своей работы использует 97 МБ памяти.

Тестирование использование ЦП проводится с целью нахождения степени загруженности центрального процессора при работе программы.

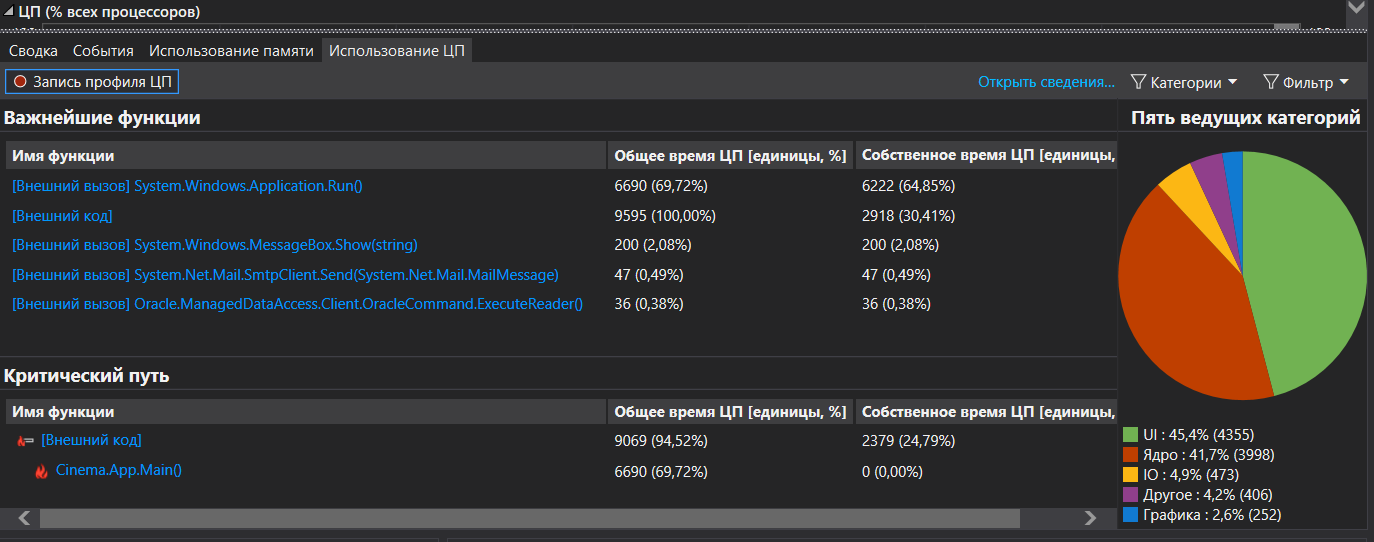


Рисунок 5.2 – Тестирование ЦП

Как показано на рисунке 5.2 тестирование приложения на загруженность процессора основная нагрузка на приложение (69%) при запуске приложения. Стоит заметить, что нагрузка на процессор не постоянная, а в среднем 0.5-0.9 секунды. С учетом того, что все страницы генерируются из базы данных, данный результат можно считать хорошим.

# **6 Описание технологии и ее применения в базе данных**

В процессе разработки проекта была разработана технология: резервное копирование и восстановление данных.

Резервное копирование – процесс создания копии данных на носителе, предназначенном для восстановления данных в оригинальном или новом месте их расположения в случае их повреждения или разрушения. Традиционный пользовательский метод резервного копирования состоит в применении команд операционной системы для копирования необходимых файлов в другое место и/или на ленточное устройство. В случае применения утилиты RMAN резервное копирование файлов базы данных Oracle выполняться внутри базы данных посредством самого сервера баз данных.

RMAN умеет делать резервные копии и копии образов файлов данных, управляющих файлов, архивных журналов повторного выполнения, файлов SPFILE. Утилита RMAN восстанавливает файлы данных из резервных копий (backup) и применяет необходимые архивные журналы повторного выполнения для приведения базы данных в актуальное состояние.

Существует два вида резервного копирования: «холодное» и «горячее».

Из «холодной» резервной копии можно восстановить только то состояние базы данных, которое было в момент останова; транзакции, сделанные после рестарта базы, в «холодную» резервную копию не попадут.

«Горячее» резервное копирование подразумевает копирование файлов базы данных без остановки базы. Поскольку запись данных в базу и чтение файлов БД никак не синхронизированы, программа резервного копирования может прочитать некорректную страницу, в которой половина будет от старой версии страницы, а другая половина – от новой.

В курсовом проекте используется «холодное» резервное копирование. Рассмотрим последовательность команд для создания резервных копий базы данных с помощью утилиты RMAN:

* $rman – запуск утилиты RMAN;
* connect target "admin/Password1@//localhost:1521/RestorePDB.be.by as sysdba" – подключение к целевой базе данных;
* shutdown immediate – немедленная остановка базы данных;
* startup mount – запуск базы данных в режиме MOUNT;
* backup database tag "full\_database\_datafiles" – создание резервной копии файлов данных;
* backup current controlfile tag "full\_database\_controlfile" – создание резервной копии управляющего файла.

Восстановление данных:

* shutdown immediate;
* startup nomount;
* restore database – восстановление файлов базы данных из резервной копии;
* recover database – восстановление данных;
* startup mount;
* alter pluggable database open – запуск базы данных.

# **7 Руководство пользователя**

При запуске приложения появляется окно авторизации (рисунок 7.1).

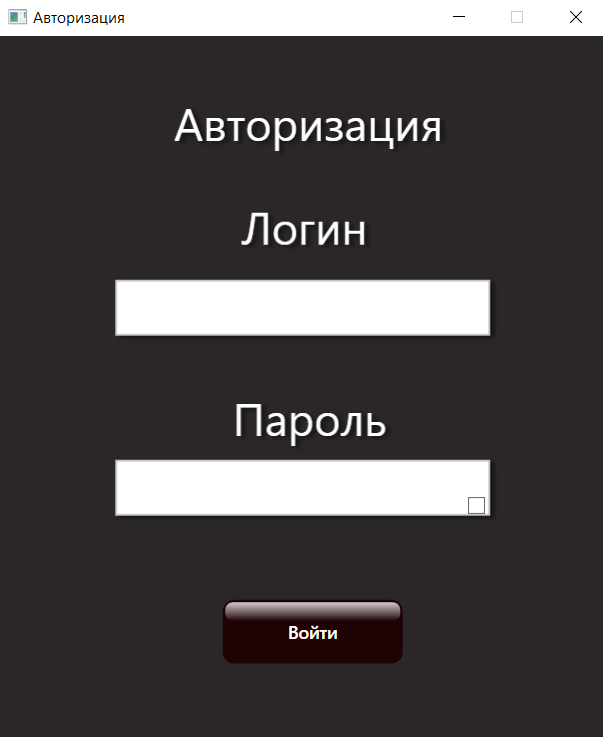


Рисунок 7.1 – Окно авторизации

Можно войти от имени администратора или от имени сотрудника кассы. При входе от имени администратора открывается окно администратора, где явно указывается ФИО сотрудника (рисунок 7.2).

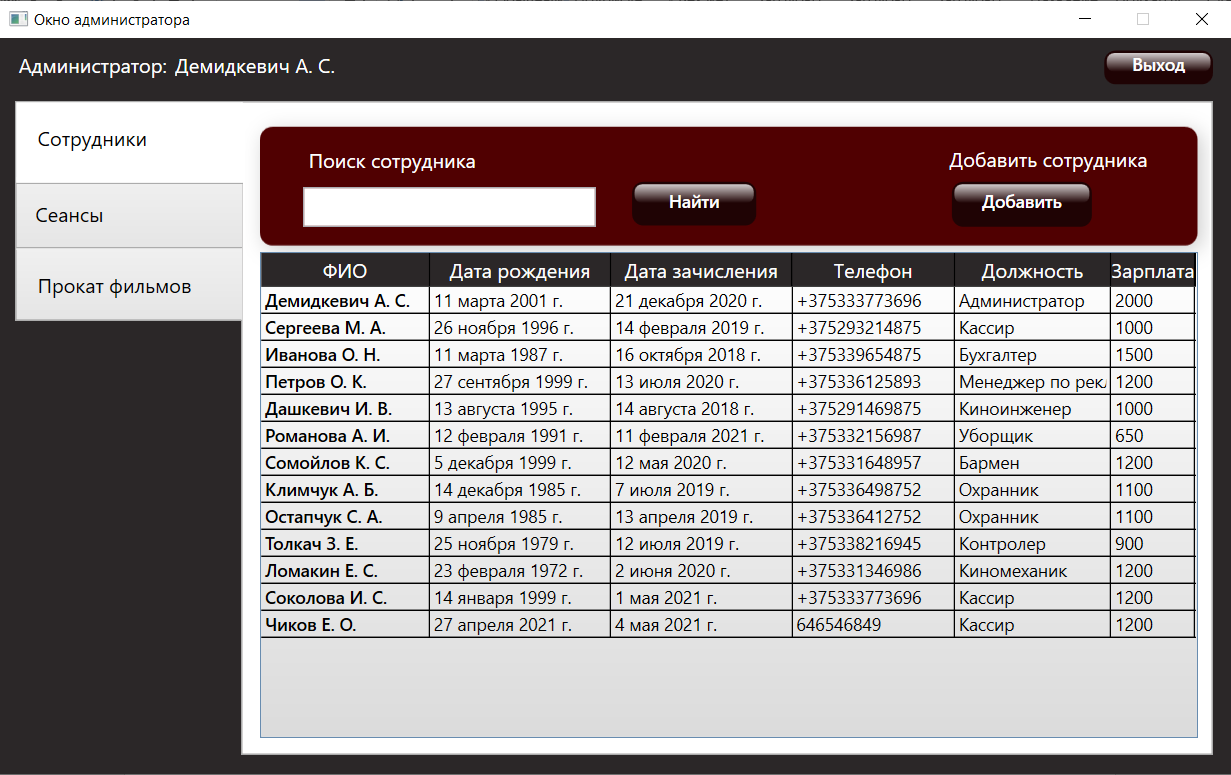


Рисунок 7.2 – Окно администратора

В данном окне есть три вкладки: сотрудники, сеансы и прокат фильмов. На вкладке сотрудники содержится список всех сотрудников кинотеатра. Также здесь есть поиск сотрудника по ФИО (рисунок 7.3).

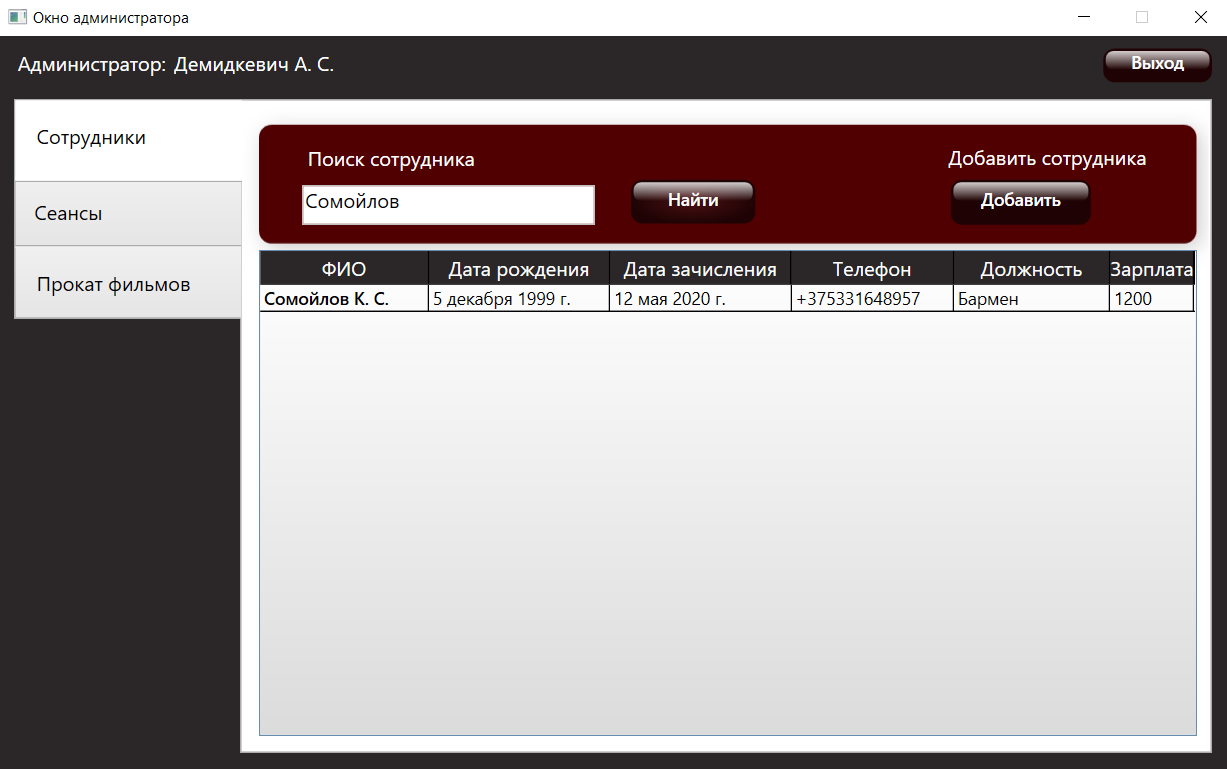


Рисунок 7.3 – Поиск сотрудников

На этом окне есть кнопка «Добавить». Нажав на неё появляется окно добавления сотрудника (рисунок 7.4).

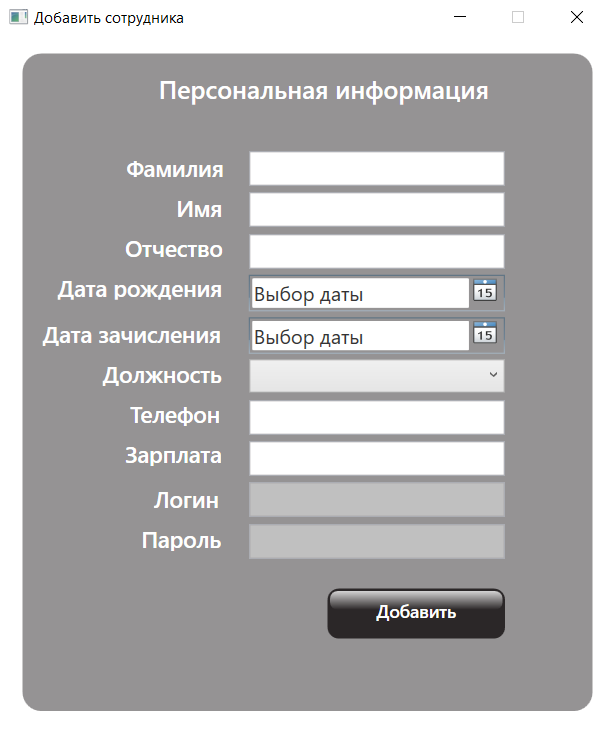


Рисунок 7.4 – Окно добавления сотрудника

В данном окне необходимо заполнить всю информацию и нажать кнопку «Добавить». Поля «Логин» и «Пароль» доступны только для администраторов и кассиров (рисунок 7.5 и рисунок 7.6).

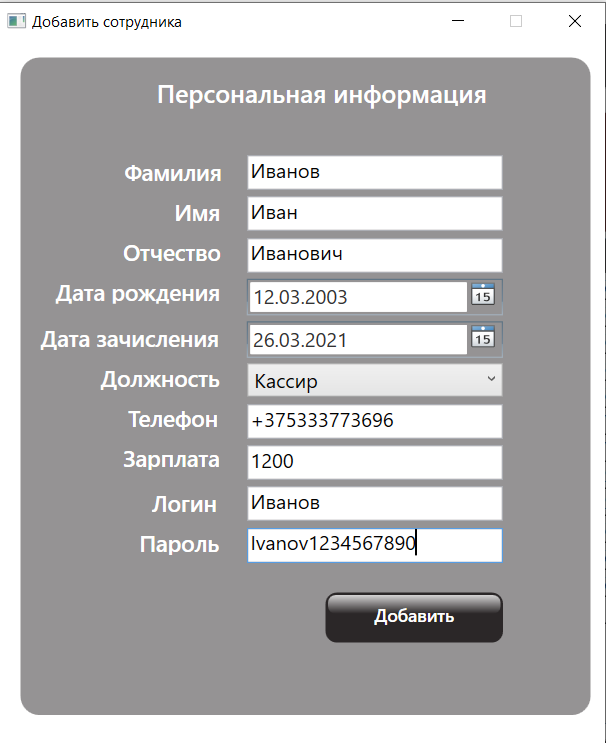


Рисунок 7.5 – Регистрация сотрудника с высоким уровнем доступа

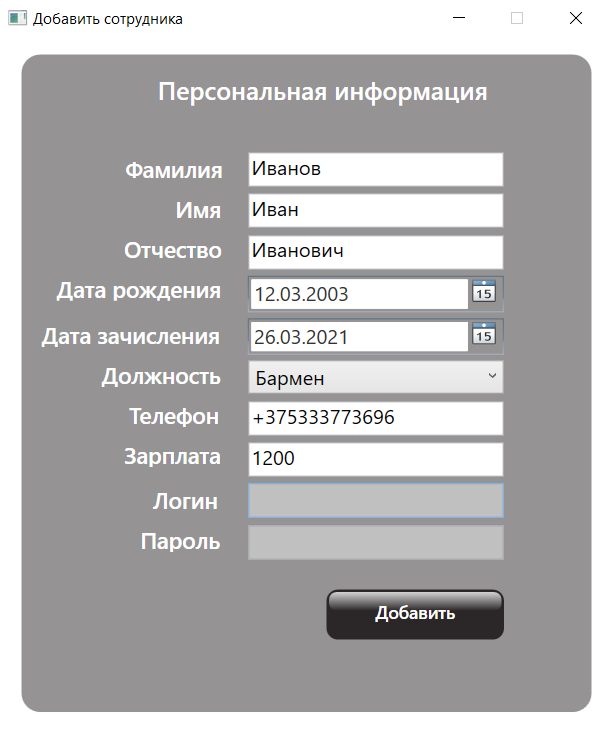


Рисунок 7.6 – Регистрация сотрудника с низким уровнем доступа

Сотрудник будет успешно добавлен в базу данных и отобразится в списке на вкладке «Сотрудники» (рисунок 7.7).

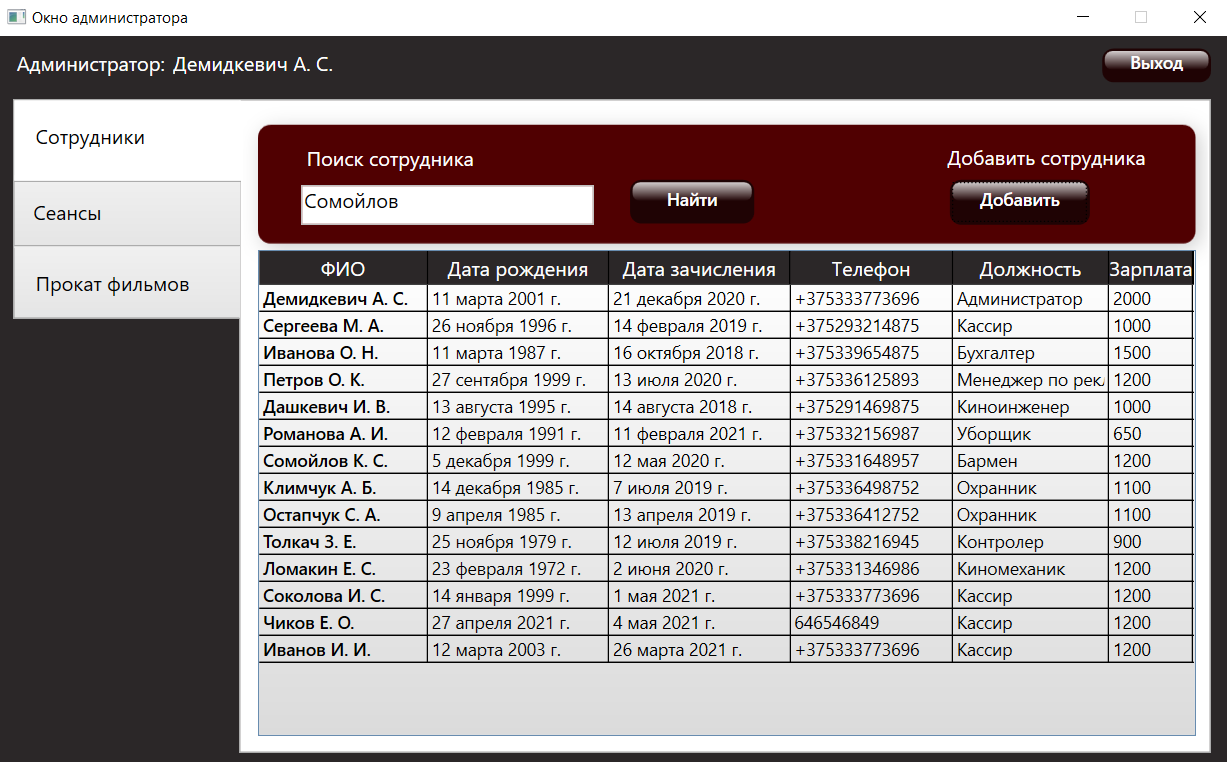


Рисунок 7.7 – Отображение добавленного сотрудника

После добавления сотрудника его данные можно редактировать, дважды щелкнув по соответствующей записи в таблице (рисунок 7.8).

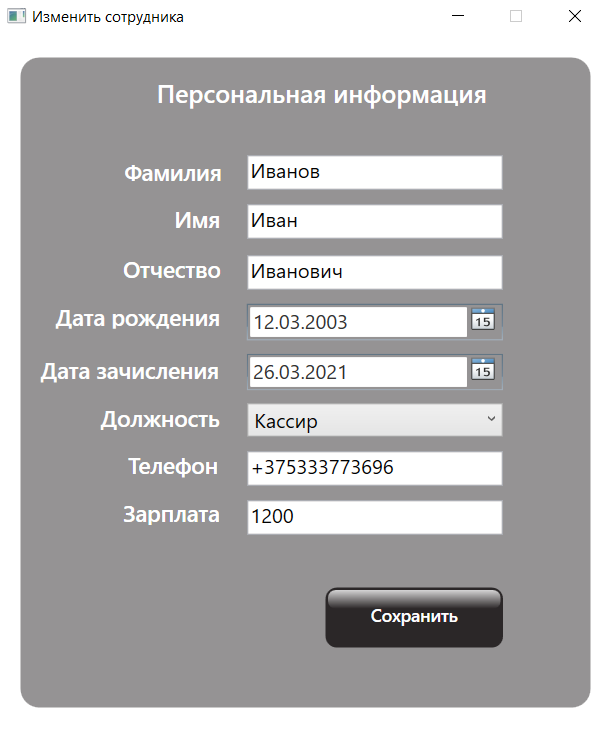


Рисунок 7.8 – Изменение данных сотрудника

После внесения поправок можно сохранить изменения нажав на кнопку «Сохранить». Также можно удалять сотрудников из списка с помощью контекстного меню, которое вызывается кликом правой кнопкой мыши по соответствующей записи (рисунок 7.9).

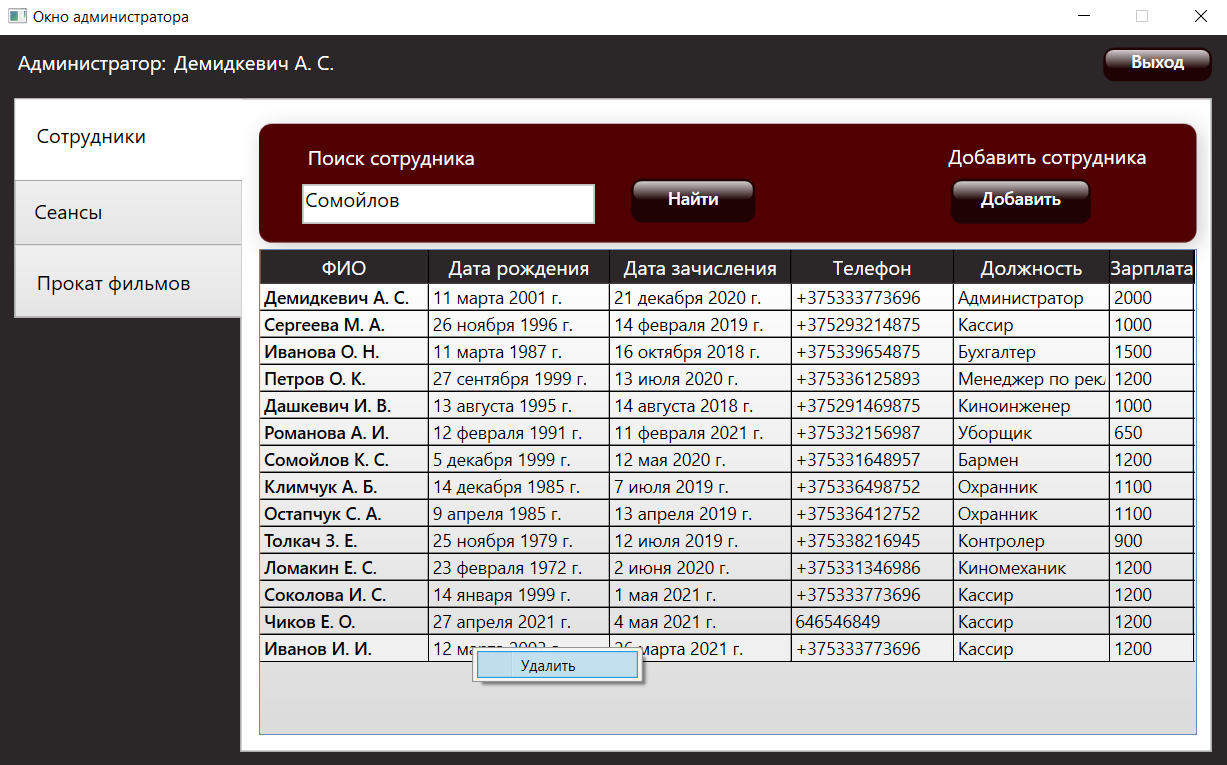


Рисунок 7.9 – Удаление сотрудника

На вкладке «Сеансы» отображается список текущих сеансов. Здесь есть поиск сеансов по нескольким критериям: дата, время и зал (рисунок 7.10).

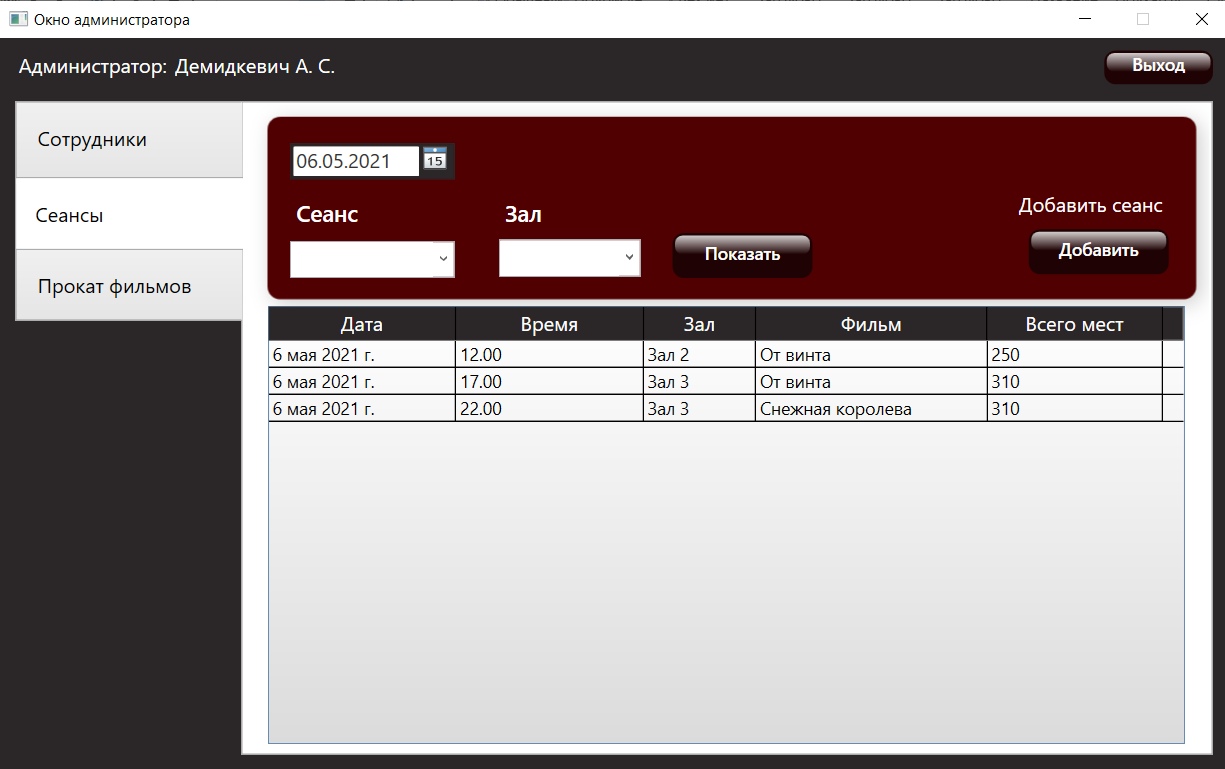


Рисунок 7.10 – Выбор сеансов по дате

Сеансы как и сотрудников можно добавлять, изменять и удалять (рисунок 7.11, 7.12, 7.13).

Данные в таблицах будут изменены динамически.

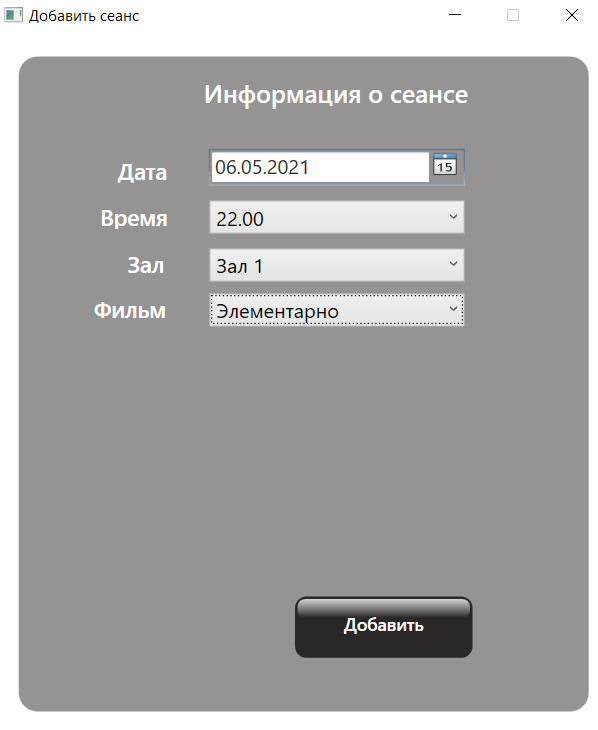


Рисунок 7.11 – Добавление сеанса

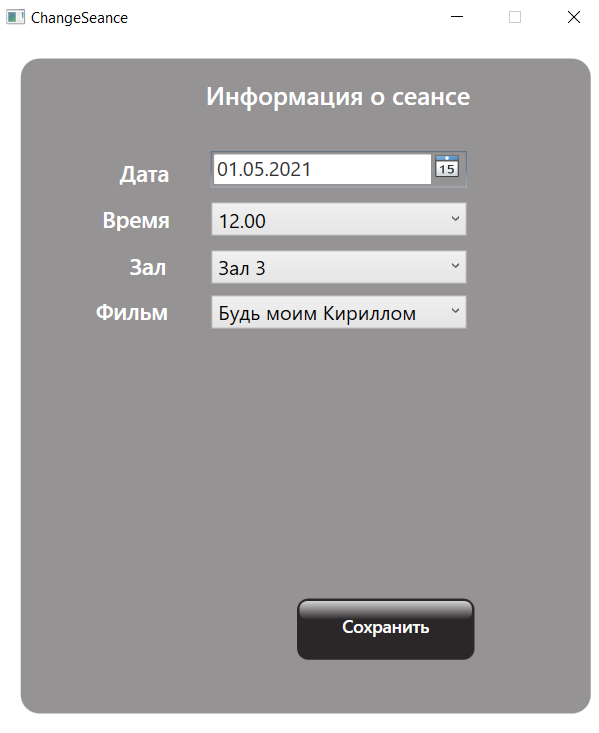


Рисунок 7.12 – Изменение сеанса

Удаление сеанса по контекстному меню, которое вызывается кликом правой кнопкой мыши.

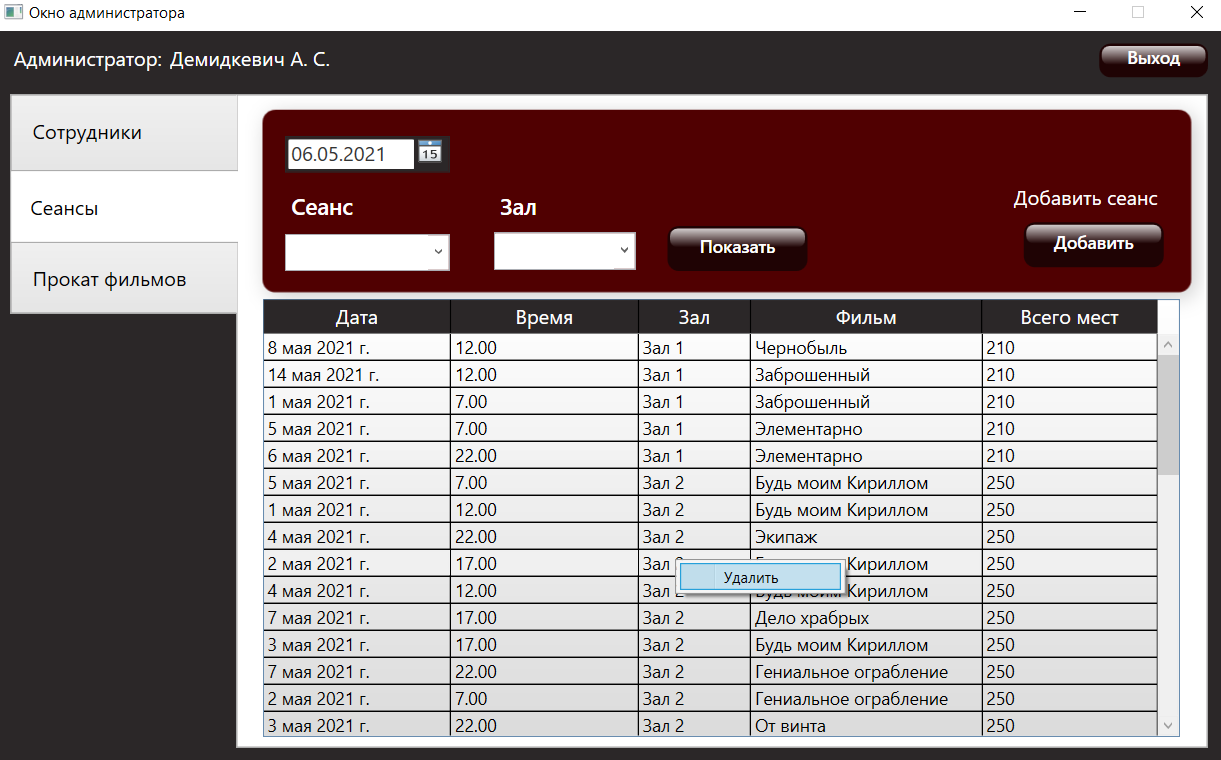


Рисунок 7.13 – Удаление сеанса

На вкладке «Прокат фильмов» находится список фильмов, которые можно отсортировать по датам проката. Также можно добавить новый фильм, заполнив все поля в окне добавления и загрузив афишу фильма (рисунок 7.14).

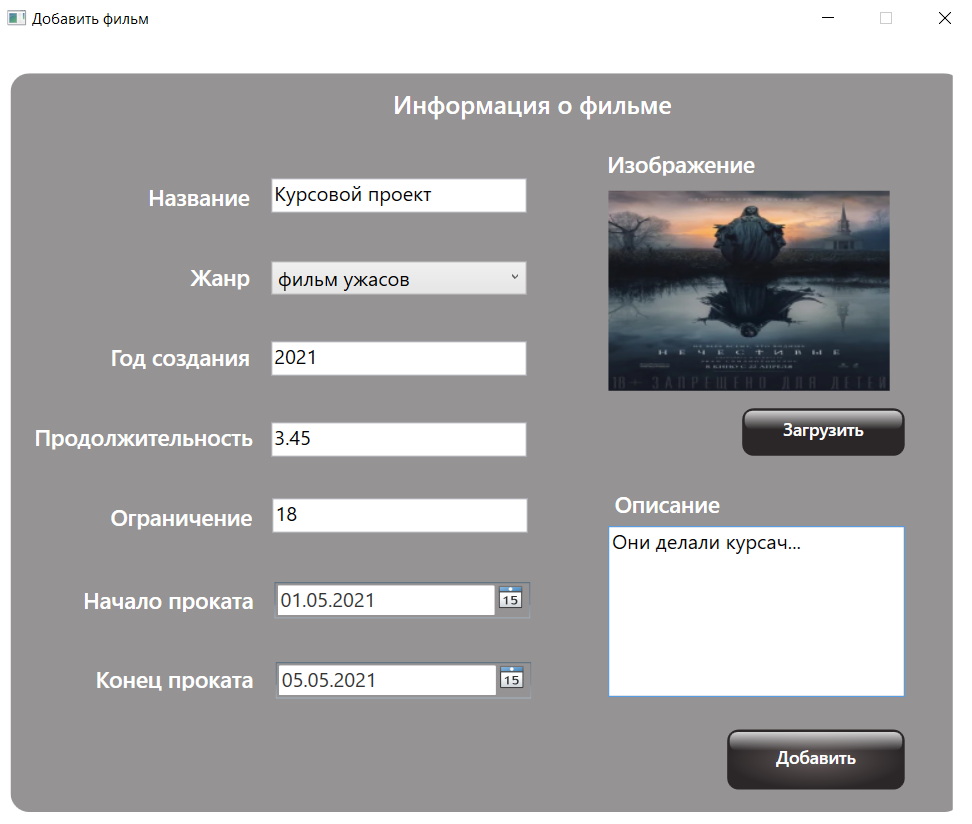


Рисунок 7.14 – Добавление фильма

По двойному нажатию по соответствующей записи в таблице откроется окно с информацией о данном фильме (рисунок 7.15).

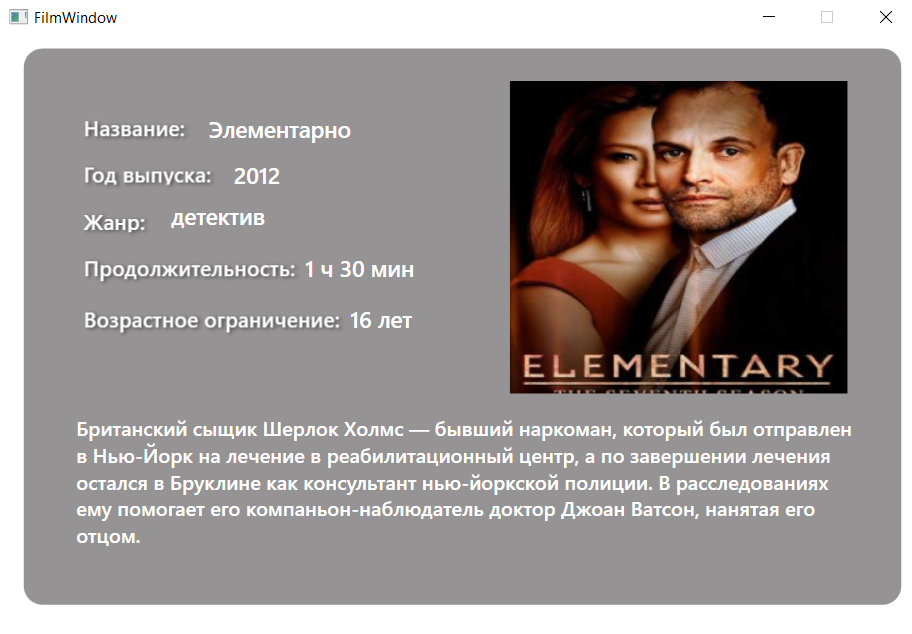


Рисунок 7.15 – Окно информации о фильме

Если зайти в приложение от лица кассира, то появится окно с двумя вкладками: «Продажа билетов» и «Проданные билеты» (рисунок 7.16).



Рисунок 7.16 – Окно кассира

На вкладке «Продажа билетов» оформляется билет на конкретный сеанс согласно дате, времени и залу. В зависимости от выбранного зала предлагается выбрать ряд и место. Так как 3 зала разной вместимости, то количество рядов для каждого зала своё. Заполнив все поля нажимаем кнопку «Продать». Информация о проданном билете выводится в соответствующее окно справа (рисунок 7.17).

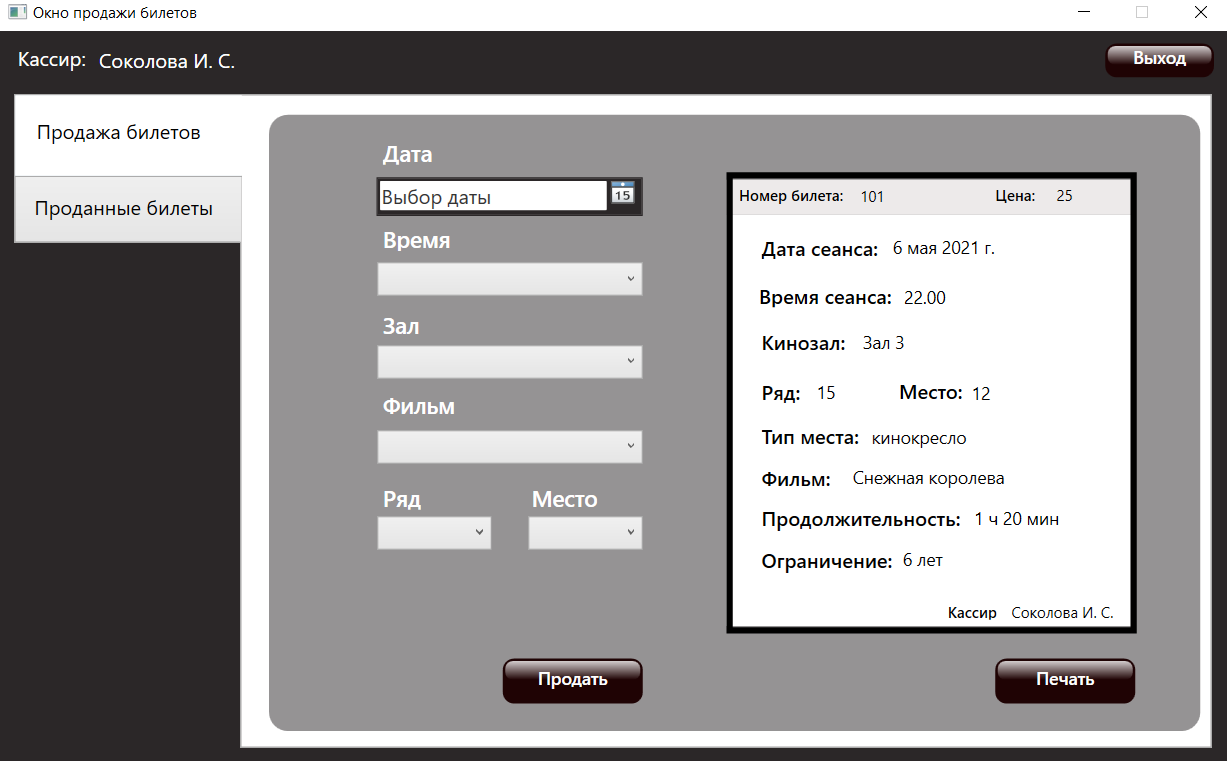


Рисунок 7.17 – Продажа билета

Данный билет можно отправить на печать или, в нашем случае, отправить на почту (рисунок 7.18 и 7.19).

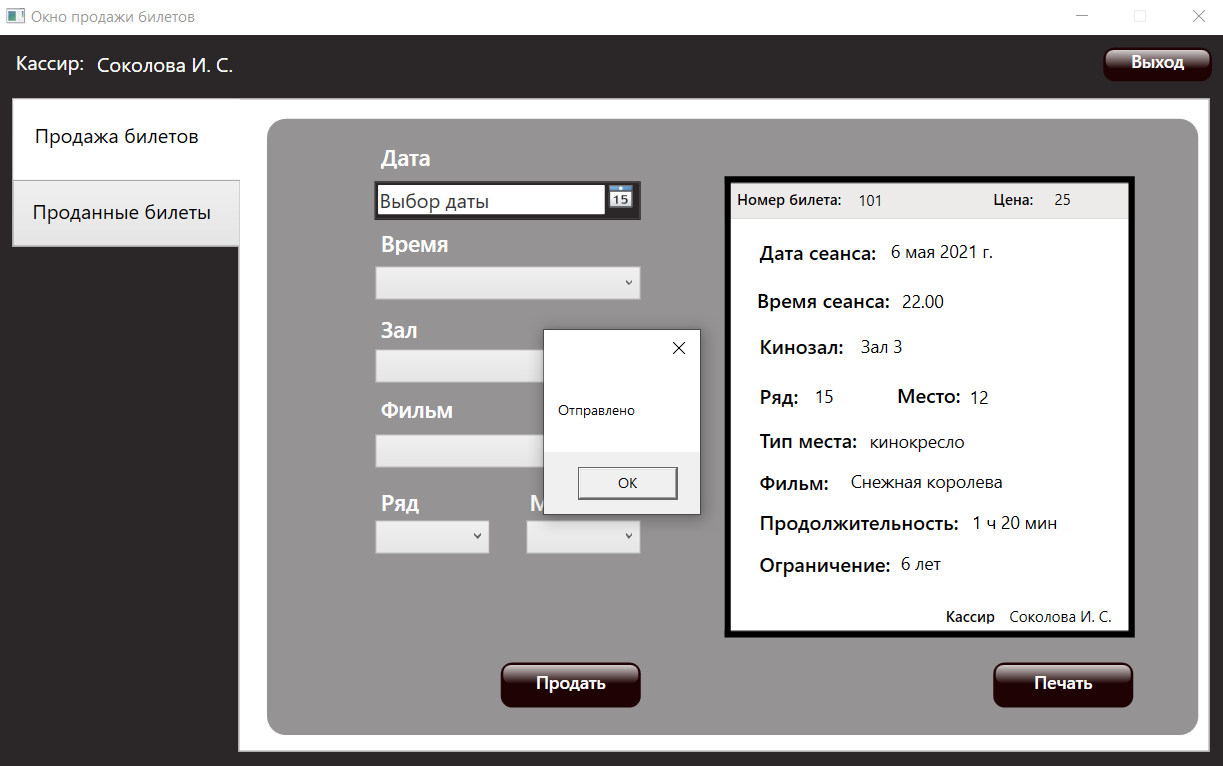


Рисунок 7.18 – Отправка билета

Ниже представлен снимок экрана электронной почты получателя билета.

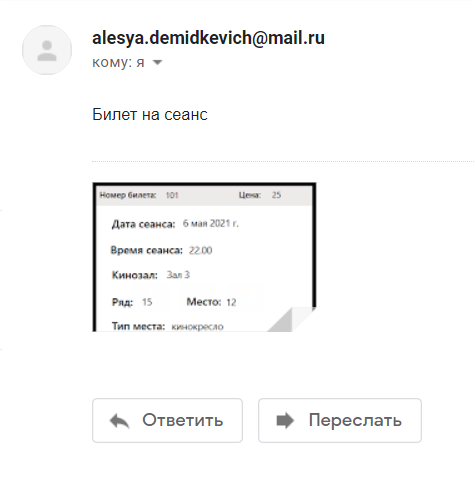


Рисунок 7.19 – Полученное сообщение

На вкладке «Проданные билеты» отображается список всех проданных билетов. Для того чтобы найти необходимый билет можно воспользоваться поиском, который осуществляется по номеру билета (рисунок 7.20).

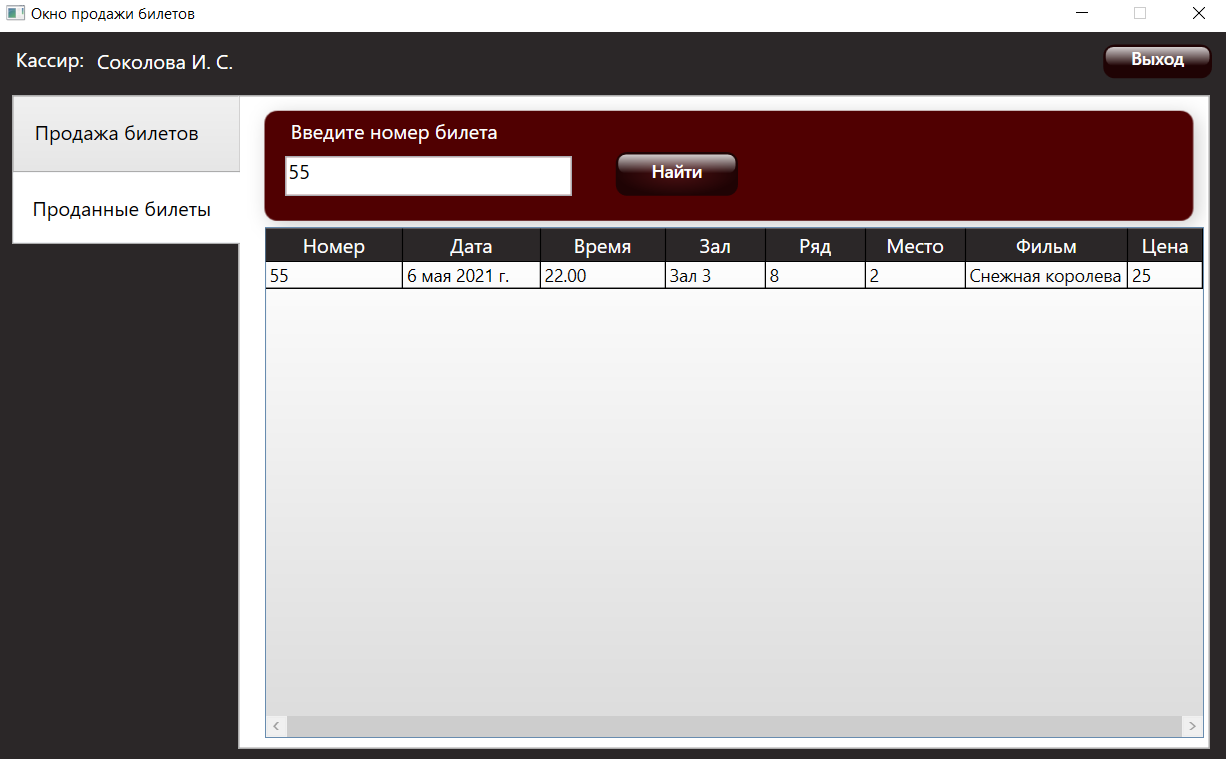


Рисунок 7.20 – Поиск билета по номеру

Проданный билет можно вернуть из контекстного меню и он снова станет доступным для продажи (рисунок 7.21).

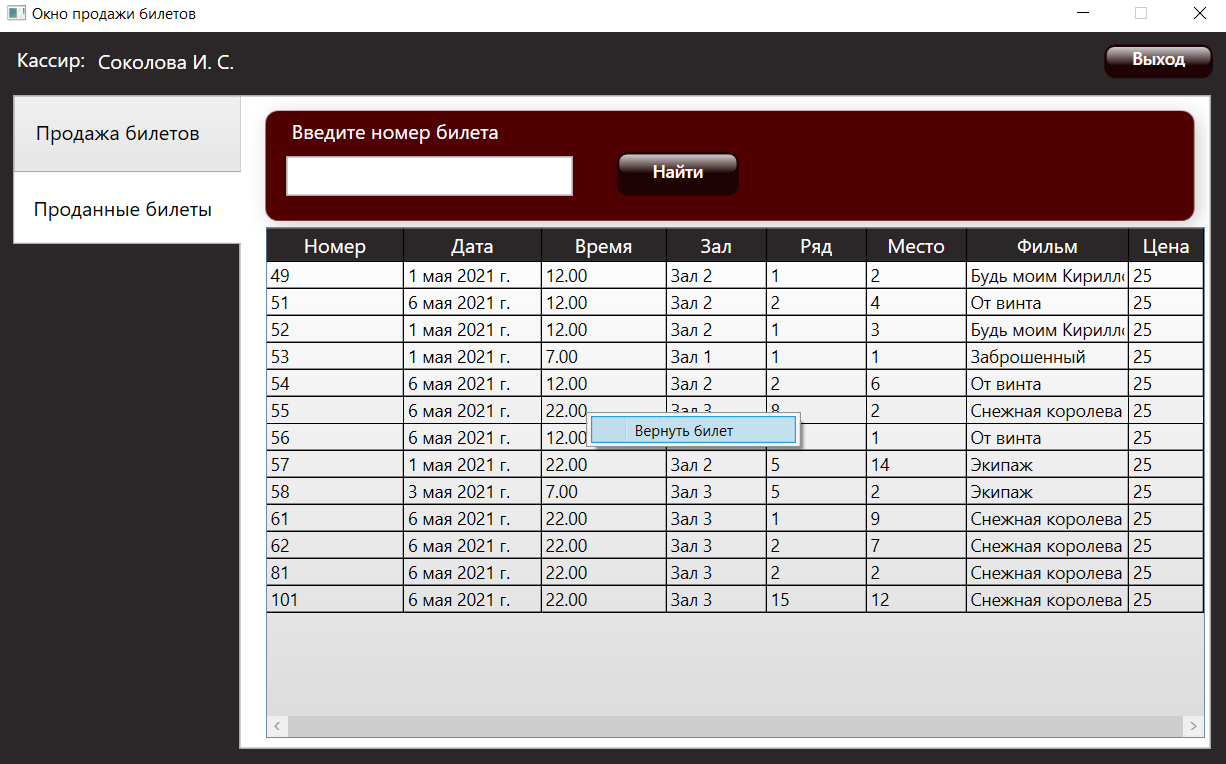


Рисунок 7.21 – Возврат билета

После завершения работы пользователь может покинуть систему нажав на кнопку «Выход» в правом верхнем углу. После этого снова появится окно авторизации (рисунок 7.22).

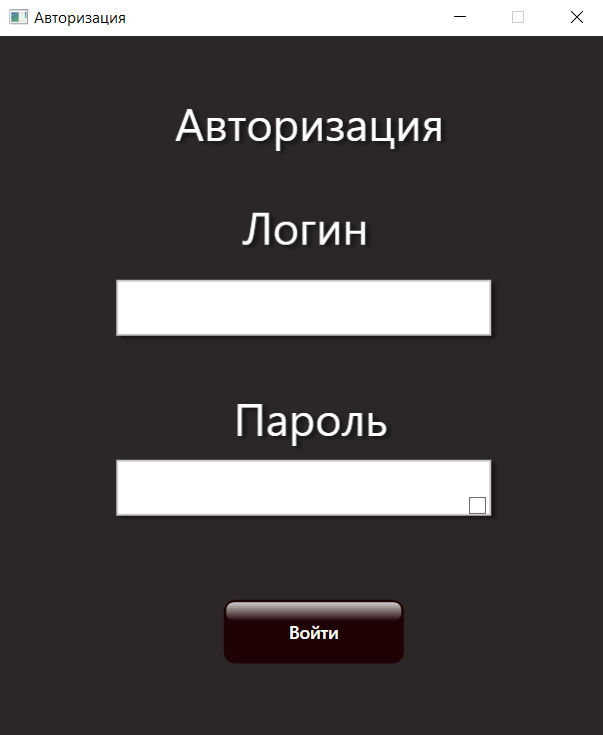


Рисунок 7.22 – Выход из системы

# **Заключение**

В процессе решения поставленной задачи была достигнута поставленная цель по созданию базы данных для автоматизированной системы «Кинотеатр», которая в совокупности с приложением формирует полноценное desktop-приложение для управления рабочим процессом. В данной работе использовалось СУБД Oracle DataBase 12c, а формирование desktop-приложения осуществлялось на языке C# с помощью системы для построения клиентских приложений Windows – WPF. При разработке курсового проекта использовались объекты: таблицы, хранимые процедуры, индексы, последовательности, пакеты.

Основной целью курсового проекта стало проектирование базы данных для дальнейшей интеграции с приложением, которое помогло облегчить взаимодействие с базой данных посредством программного интерфейса. Взаимодействие между приложением и базой данных было реализовано с помощью технологии ADO.NET. При разработке выполнены следующие пункты:

* разработка приложения для взаимодействия с БД;
* регистрация/удаление сотрудника;
* изменение данных сотрудника;
* обновление списка фильмов;
* просмотр информации о каждом фильме;
* обновление списка сеансов;
* сортировка списка сеансов;
* продажа и возврат билетов;
* отправка билета по почте.

Приложение прошло тестирование при использовании в БД большого количество данных. Также были реализованы процедуры для импорта, экспорта данных в формат XML.

Была реализована технология «Резервное копирование и восстановление данных». Шифрование данных было реализовано с помощью пакета DBMS\_CRYPTO.

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

# **Cписок источников**

1. Язык C# и .NET Framework [Электронный ресурс] – http://professorweb.ru/my/csharp/charp\_theory/level1/infonet.php.
2. Руководство по ADO.NET и работе с базами данных [Электронный ресурс] – https://metanit.com/sharp/adonet/.
3. Документация Oracle [Электронный ресурс] / Foundation, Inc. https://docs.oracle.com/cd/B28359\_01/server.111/b31222/toc.htm
4. Официальный сайт Oracle [Электронный ресурс] / Foundation, Inc. https://www.oracle.com/database/database-vault/index.html
5. Продукты Oracle [Электронный ресурс] [Электронный ресурс] / Foundation, Inc. http://www.interface.ru/home.asp?artId=24678
6. Документы Oracle [Электронный ресурс] / Foundation, Inc. http://www.oracle.com/technetwork/database/security/database-vault-ds-12c-1898877.pdf

# **Приложение А**

//Создание таблиц

--table Post

create table Post

(

ID number GENERATED ALWAYS AS IDENTITY (start with 1 increment by 1) PRIMARY KEY,

positionName nvarchar2(50) not null,

accessLevel number not null

);

--table Employee

create table Employee

(

ID number GENERATED ALWAYS AS IDENTITY (start with 1 increment by 1) PRIMARY KEY,

name nvarchar2(50) not null,

secondName nvarchar2(50),

surname nvarchar2(50) not null,

phoneNumber nvarchar2(50),

birthday date not null,

dateOfEnrollment date not null,

salary number(8,2) not null,

IDPost number not null,

foreign key (IDPost)

references Post(ID) on delete cascade

);

--table Users

create table Users

(

ID number GENERATED ALWAYS AS IDENTITY (start with 1 increment by 1) PRIMARY KEY,

login nvarchar2(50),

password nvarchar2(64),

IDEmployee number not null,

foreign key (IDEmployee)

references Employee(ID) on delete cascade

);

--table TypePlace

create table TypePlace

(

ID number GENERATED ALWAYS AS IDENTITY (start with 1 increment by 1) PRIMARY KEY,

name nvarchar2(50) not null,

cost number not null

);

--table HAll

create Table Hall

(

ID number GENERATED ALWAYS AS IDENTITY (start with 1 increment by 1) PRIMARY KEY,

name nvarchar2(50) not null,

capacity number not null,

typeHall nvarchar2(50) not null

);

--table Place

create table Place

(

ID number GENERATED ALWAYS AS IDENTITY (start with 1 increment by 1) PRIMARY KEY,

rowNumber number not null,

place number not null,

IDHall number not null,

foreign key (IDHall)

references Hall(ID) on delete cascade,

IDTypePlace number not null,

foreign key (IDTypePlace)

references TypePlace(ID) on delete cascade

);

--table Genre

create table Genre

(

ID number GENERATED ALWAYS AS IDENTITY (start with 1 increment by 1) PRIMARY KEY,

name nvarchar2(50)not null

);

--table Film

create table Film

(

ID number GENERATED ALWAYS AS IDENTITY (start with 1 increment by 1) PRIMARY KEY,

name nvarchar2(50) not null,

year nvarchar2(50) not null,

duration nvarchar2(50) not null,

startRelease date not null,

endRelease date not null,

description nvarchar2(500) not null,

ageLimit number not null,

IDGenre number not null,

img blob,

foreign key (IDGenre)

references Genre(ID) on delete cascade

);

--table Seance

create table Seance

(

ID number GENERATED ALWAYS AS IDENTITY (start with 1 increment by 1) PRIMARY KEY,

dateSeance date not null,

timeSeance nvarchar2(50) not null,

IDHall number not null,

foreign key (IDHall)

references Hall(ID) on delete cascade,

IDFilm number not null,

foreign key (IDFilm)

references Film(ID) on delete cascade

);

--table Ticket

create table Ticket

(

ID number GENERATED ALWAYS AS IDENTITY (start with 1 increment by 1) PRIMARY KEY,

cost number,

IDSeance number not null,

foreign key (IDSeance)

references Seance(ID) on delete cascade,

IDPlace number not null,

foreign key (IDPlace)

references Place(ID) on delete cascade,

IDEmployee number not null,

foreign key (IDEmployee)

references Employee(ID) on delete cascade,

ticketNumber int default ticket\_number.nextval not null

);

# **Приложение Б**

--Export/Import XML

CREATE OR REPLACE DIRECTORY UTLDATA AS 'C:/XML';

CREATE OR REPLACE PACKAGE XML\_PACKAGE IS

PROCEDURE EXPORT\_POST\_TO\_XML;

PROCEDURE IMPORT\_POST\_FROM\_XML;

END XML\_PACKAGE;

CREATE OR REPLACE PACKAGE BODY XML\_PACKAGE

IS

PROCEDURE EXPORT\_POST\_TO\_XML

IS

DOC DBMS\_XMLDOM.DOMDocument;

XDATA XMLTYPE;

CURSOR XMLCUR IS

SELECT XMLELEMENT("POST",

MLAttributes('http://www.w3.org/2001/XMLSchema' AS "xmlns:xsi",

'http://www.oracle.com/Post.xsd' AS "xsi:nonamespaceSchemaLocation"),

XMLAGG(XMLELEMENT("POST",

XMLELEMENT("ID",PT.ID),

XMLELEMENT("POSITIONNAME",PT.POSITIONNAME),

XMLELEMENT("ACCESSLEVEL",PT.ACCESSLEVEL)

))

) FROM POST PT;

BEGIN

OPEN XMLCUR;

LOOP

FETCH XMLCUR INTO XDATA;

EXIT WHEN XMLCUR%NOTFOUND;

END LOOP;

CLOSE XMLCUR;

DOC := DBMS\_XMLDOM.NewDOMDocument(XDATA);

DBMS\_XMLDOM.WRITETOFILE(DOC, 'UTLDATA/post.xml');

END EXPORT\_POST\_TO\_XML;

PROCEDURE IMPORT\_POST\_FROM\_XML

IS

L\_CLOB CLOB;

L\_BFILE BFILE := BFILENAME('UTLDATA', 'post.xml');

L\_DEST\_OFFSET INTEGER := 1;

L\_SRC\_OFFSET INTEGER := 1;

L\_BFILE\_CSID NUMBER := 0;

L\_LANG\_CONTEXT INTEGER := 0;

L\_WARNING INTEGER := 0;

P DBMS\_XMLPARSER.PARSER;

V\_DOC DBMS\_XMLDOM.DOMDOCUMENT;

V\_ROOT\_ELEMENT DBMS\_XMLDOM.DOMELEMENT;

V\_CHILD\_NODES DBMS\_XMLDOM.DOMNODELIST;

V\_CURRENT\_NODE DBMS\_XMLDOM.DOMNODE;

PT POST%ROWTYPE;

BEGIN

DBMS\_LOB.CREATETEMPORARY (L\_CLOB, TRUE);

DBMS\_LOB.FILEOPEN(L\_BFILE, DBMS\_LOB.FILE\_READONLY);

DBMS\_LOB.LOADCLOBFROMFILE (DEST\_LOB => L\_CLOB, SRC\_BFILE => L\_BFILE, AMOUNT => DBMS\_LOB.LOBMAXSIZE,

DEST\_OFFSET => L\_DEST\_OFFSET, SRC\_OFFSET => L\_SRC\_OFFSET, BFILE\_CSID => L\_BFILE\_CSID,

LANG\_CONTEXT => L\_LANG\_CONTEXT, WARNING => L\_WARNING);

DBMS\_LOB.FILECLOSE(L\_BFILE);

COMMIT;

P := DBMS\_XMLPARSER.NEWPARSER;

DBMS\_XMLPARSER.PARSECLOB(P,L\_CLOB);

V\_DOC := DBMS\_XMLPARSER.GETDOCUMENT(P);

V\_ROOT\_ELEMENT := DBMS\_XMLDOM.Getdocumentelement(v\_Doc);

V\_CHILD\_NODES:= DBMS\_XMLDOM.GETCHILDRENBYTAGNAME(V\_ROOT\_ELEMENT,'\*');

FOR i IN 0 .. DBMS\_XMLDOM.GETLENGTH(V\_CHILD\_NODES) - 1

LOOP

V\_CURRENT\_NODE := DBMS\_XMLDOM.ITEM(V\_CHILD\_NODES,i);

DBMS\_XSLPROCESSOR.VALUEOF(V\_CURRENT\_NODE,

'ID/text()',PT.ID);

DBMS\_XSLPROCESSOR.VALUEOF(V\_CURRENT\_NODE,

'POSITIONNAME/text()',PT.POSITIONNAME);

DBMS\_XSLPROCESSOR.VALUEOF(V\_CURRENT\_NODE,

'ACCESSLEVEL/text()',PT.ACCESSLEVEL);

BEGIN

DBMS\_OUTPUT.PUT\_LINE('ID: '||PT.ID);

DBMS\_OUTPUT.PUT\_LINE('POSITIONNAME: '||PT.POSITIONNAME);

DBMS\_OUTPUT.PUT\_LINE('ACCESSLEVEL: '||PT.ACCESSLEVEL);

DBMS\_OUTPUT.PUT\_LINE('');

END;

END LOOP;

DBMS\_LOB.FREETEMPORARY(L\_CLOB);

DBMS\_XMLPARSER.FREEPARSER(P);

DBMS\_XMLDOM.FREEDOCUMENT(V\_DOC);

COMMIT;

EXCEPTION

WHEN OTHERS THEN

DBMS\_LOB.FREETEMPORARY(L\_CLOB);

DBMS\_XMLPARSER.FREEPARSER(P);

DBMS\_XMLDOM.FREEDOCUMENT(V\_DOC);

RAISE\_APPLICATION\_ERROR(-20101, 'IMPORT XML ERROR'|| SQLERRM);

END IMPORT\_POST\_FROM\_XML;

END XML\_PACKAGE;

begin

XML\_PACKAGE.EXPORT\_POST\_TO\_XML();

XML\_PACKAGE.IMPORT\_POST\_FROM\_XML();

end;

# **Приложение В**

--Scheduler

create or replace procedure deleteFilmSheduler

is

begin

delete from Seance s where IDFilm in (select ID from Film where EndRelease < trunc(sysdate));

commit;

delete from Film where ID in (select ID from Film where EndRelease < trunc(sysdate));

commit;

end deleteFilmSheduler;

begin

dbms\_scheduler.create\_schedule(

schedule\_name =>'Delete\_film',

start\_date =>CURRENT\_TIMESTAMP,

repeat\_interval => 'FREQ=MINUTELY;INTERVAL=1'

);

end;

begin

dbms\_scheduler.create\_program(

program\_name => 'Delete\_film\_progarm',

program\_type => 'STORED\_PROCEDURE',

program\_action => 'deleteFilmSheduler',

number\_of\_arguments => 0,

enabled => true

);

end;

begin

dbms\_scheduler.create\_job(

job\_name => 'Delete\_film\_job',

program\_name => 'Delete\_film\_progarm',

schedule\_name =>'Delete\_film',

enabled => true

);

end;

exec dbms\_Scheduler.run\_job('Delete\_film\_job');