|  |  |
| --- | --- |
| Институт (факультет) | Информационных технологий |
| Кафедра | Математическое и программное обеспечение ЭВМ |

КУРСОВАЯ РАБОТА

|  |
| --- |
| по дисциплине Проектирование баз данных |

|  |  |
| --- | --- |
| на тему | Проектирование базы данных кинотеатра |

|  |
| --- |
| Выполнил студент группы 1ПИб-02-2оп-22 |
| *группа* |
| направления подготовки (специальности) |
| 09.03.04, Програмная инженерия |
| *шифр, наименование* |
| Зернов Владислав Александрович |
| *фамилия, имя, отчество* |

|  |
| --- |
| Руководитель |
| Селяничев Олег Леонидович |
| *фамилия, имя, отчество* |
| доцент |
| *должность* |

|  |
| --- |
| Дата представления работы |
| «\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20 \_\_\_ г. |
|  |
| Заключение о допуске к защите |
|  |
|  |
|  |
|  |
| Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| количество баллов |
| Подпись преподавателя\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Череповец, 2025 г.

Оглавление

[Введение 3](#_Toc200317158)

[1. Описание предметной области 4](#_Toc200317159)

[2. Инфологическое проектирование 6](#_Toc200317160)

[3. Даталогическое проектирование 9](#_Toc200317161)

[3.1 Нормализация 10](#_Toc200317162)

[3.2 Логическая модель 37](#_Toc200317163)

[3.3 Физическая модель 40](#_Toc200317164)

[4. Программирование на языке высокого уровня 46](#_Toc200317165)

[Заключение 52](#_Toc200317166)

[Список литературы 53](#_Toc200317167)

[Приложение 1. Техническое задание 55](#_Toc200317168)

[Приложение 2. Руководство пользователя 60](#_Toc200317169)

[Приложение 3. Текст программы 65](#_Toc200317170)

Введение

Сегодня базы данных применяются буквально повсюду – от больших промышленных компаний до городских учреждений. Раньше, до возникновения баз данных, компаниями велся учет данных в журналах, намного увеличивавший шанс сделать ошибку заполняемой информации. При ошибке приходилось ставить соответствующие пометы об обнаружении неправильно внесенной информации, что при многократном корректировании заметно уменьшало пространство в журнале. Стоит отметить не менее важную деталь при заполнении журнала вручную – это медленная скорость занесения сведений о производственной деятельности организации. Скорость зависела от навыка и опыта работы специалиста – если человек неопытен, то журнал заполняется с многократными проверками нужных соответствий, что очень снижало скорость.

Но после появления систем управления баз данных работа внесения, управления и хранения информации стала в разы проще и быстрее. Автоматизация ручного процесса позволяет работать с информацией оперативнее – систематизируя и обрабатывая данные [1]. Скорость работы с данными стала быстрее, по сравнению с человеческим трудом. А наличие ошибок стало не критичным. При допуске ошибок можно удалить, перенести ненужные или устаревшие данные, не беспокоясь о заполняемом месте данных.

Для примера автоматизации баз данных можно взять такую предметную область, как киноиндустрию, а именно кинотеатр. На данный момент, скорость и исправление ошибок актуально для этой области, так как работа кинотеатра осуществляется круглосуточно, а это значит, что поток данных не прекращается, а в соответствии с человеческим фактором, допустить ошибку в таких условиях легко. Для того, чтобы работать с данными, а именно – удалять, добавлять, управлять информацией, понадобится база данных и приложение, которое взаимодействует с ней.

Цель автоматизации – упрощение учета фильмов, ассортимента меню, акций и финансовых операций. Использование базы данных позволит сотрудникам кинотеатра:

* хранить и оперативно обрабатывать информацию о сеансах и посещениях посетителей;
* управлять расписанием сеансов и фильмографией, ассортиментом меню;
* контролировать сроки действия акций;
* автоматизировать финансовую отчетность.

Без автоматизации данные приходилось бы вести в бумажных журналах или таблицах, что увеличивает риск ошибок, усложняет поиск и редактирование информации. Ручной учет также замедляет процессы продажи билетов и формирования отчетов.

Внедрение базы данных минимизирует вероятность дублирования записей, ускорит обработку данных и повысит качество обслуживания посетителей. Это особенно важно в условиях высокой конкуренции на рынке кинотеатров, где удобство и скорость работы напрямую влияют на лояльность посетителей.

Целью данной курсовой работы является проектирование базы данных кинотеатра, её нормализация и создание программы на языке высокого уровня, содержащую в себе пользовательский интерфейс и реализующую запросы к базе данных.

1. Описание предметной области

Кинотеатр — это учреждение, предоставляющее услуги по организации кинопоказов, продаже билетов и сопутствующих товаров.

В учреждении работает персонал, включающий администраторов, кассиров, маркетологовд, а также технический персонал, отвечающий за обслуживание оборудования и контроль показа.

Основная деятельность кинотеатра строится вокруг показа фильмов в специально оборудованных залах. Каждый фильм обладает набором характеристик, включая название, жанровую принадлежность, продолжительность, возрастные ограничения и информацию о создателях. Эти данные постоянно обновляются, но из-за отсутствия единой системы управления часто возникают расхождения между информацией на сайте, в мобильном приложении и на кассе.

Кинозалы как ключевые объекты инфраструктуры имеют свои технические параметры: вместимость, схему расположения мест. Особую сложность представляет актуализация информации при изменении конфигурации залов или проведении технических работ. В настоящее время такие изменения не всегда своевременно доводятся до всех сотрудников, что может приводить к ошибкам при продаже билетов.

Процесс организации кинопоказа включает формирование расписания сеансов, где необходимо учитывать множество факторов: продолжительность фильмов, время на уборку залов, технические требования к оборудованию. В существующей системе часто возникают накладки из-за ручного составления расписания, когда возможны пересечения сеансов или недостаточные временные промежутки между показами.

Продажа билетов представляет собой наиболее динамичный процесс, требующий оперативного обновления данных. Каждая продажа или бронирование должны мгновенно отражаться в системе, чтобы избежать двойного бронирования мест. Однако в текущей практике случаются ситуации, когда информация о занятых местах не синхронизируется между онлайн-продажами и кассой.

Движение информации в кинотеатре представляет собой сложный многоуровневый процесс. От поступления информации о новых фильмах до формирования конечной отчетности данные проходят через множество этапов обработки. В существующей системе эти потоки информации часто дублируются, теряются или искажаются, что снижает общую эффективность управления кинотеатром.

Особую проблему представляет анализ деятельности кинотеатра. Без структурированного хранения данных практически невозможно оперативно получить достоверную информацию о выручке, посещаемости, популярности фильмов или эффективности работы персонала. Это значительно осложняет процесс принятия управленческих решений.

2. Инфологическое проектирование

Моделирование данных – это процесс создания логического представления структуры базы данных [4].

Все этапы проектирования базы данных подразумевают создание моделей данных об интересующей предметной области. В данном случае предметная область – кинотеатр.

Моделирование данных упрощает понимание смысла элементов данных, способствует более плодотворному общению пользователей и разработчиков.

Исходя из важности адекватного отображения предметной области, к моделям данных предъявляют ряд требований, и определенные критерии для оценки их оптимальности [5]:

* структурная достоверность – соответствие способу определения и организации информации в данной предметной области;
* простота – легкость понимания модели разработчиками и пользователями информационной системы;
* выразительность – способность представлять отличия между разными типами данных, связи между данными и ограничения;
* отсутствие избыточности – исключение излишней информации, т.е. любая часть данных должна быть представлена только в одном месте;
* готовность к совместному использованию – отсутствие принадлежности к какому-то особому приложению или использованию технологии;
* расширяемость – способность эволюционировать с целью включения новых требований с минимальным влиянием на существующих пользователей;
* целостность – согласованность по способам использования и управления информацией;
* представление в виде диаграмм – способность представления модели с помощью понятных широкому кругу пользователей обозначений.

Инфологическое (концептуальное) проектирование – процесс создания внешней (инфологической) модели данных о предметной области, не зависящее от любых физических аспектов ее представления [6].

Система управления базами данных (СУБД) – совокупность программ и языковых средств, предназначенных для управления [6].

данными в базе данных, ведения базы данных и обеспечения взаимодействия ее с прикладными программами [4].

На этом этапе используется информация, объединяющая требования пользователей. Инфологическое проектирование базы данных не зависит от таких подробностей ее реализации, как тип выбранной СУБД, набор создаваемых прикладных программ, используемые языки программирования, тип вычислительной системы и т.п. При разработке инфологическая модель постоянно подвергается критической оценке, проверке на соответствие требованиям пользователей, и при необходимости модифицируется. От качества созданной инфологической модели в определяющей степени зависит эффективность конечной базы данных.

Модель данных (МД) – совокупность правил порождения структур данных в базе данных, операций над ними, а также ограничений целостности, определяющих допустимые связи и значения данных, последовательность их изменения [6].

Существует множество моделей данных:

* иерархическая модель данных (ИМД) – это модель, в которой абстрактные понятия находятся в отношении предшествования таким образом, что каждому понятию соответствует только один предшественник (родитель);
* сетевая модель данных (СМД) – это та же ИМД, но у него нет ограничения, что у каждого понятия только один родитель;
* реляционная модель данных (РМД) – это модель, в которой данные можно представить в виде отношений, изменяющихся во времени;
* объектно-ориентированная модель данных (ООМД) представляет структуру, которую можно изобразить графически в виде дерева, узлами которого являются объекты.

В случае кинотеатра лучшим вариантом модели данных является реляционный, так как через него проще всего смоделировать предметную область, так же данная модель гибкая и обеспечивает эффективное управление информацией.

Основными конструктивными элементами инфологических моделей являются сущности, связи между ними и их свойства (атрибуты) [6].

Сущность – любой различимый объект, информацию о котором необходимо хранить в базе данных. Тип сущности – это набор однородных личностей, предметов, событий или идей, выступающих как целое.

Атрибут – поименованная характеристика сущности. Его наименование должно быть уникальным для конкретного типа сущности, но может быть одинаковым для различного типа сущностей. Атрибуты используются для определения того, какая информация должна быть собрана о сущности. Абсолютное различие между типами сущностей и атрибутами отсутствует. Атрибут является таковым только в связи с типом сущности. В другом контексте атрибут может выступать как самостоятельная сущность.

Ключ – минимальный набор атрибутов, по значениям которых можно однозначно найти требуемый экземпляр сущности. Минимальность означает, что исключение из набора любого атрибута не позволяет идентифицировать сущность по оставшимся.

Связь – ассоциирование двух или более сущностей.

Цель инфологического моделирования – обеспечение наиболее естественных для человека способов сбора и представления той информации, которую предполагается хранить в создаваемой базе данных [6]. Поэтому инфологическую модель данных пытаются строить на доступном широкому кругу пользователей и разработчиков языке. Известны следующие средства создания внешних моделей:

* семантические сети;
* язык инфологического моделирования;
* ER-диаграммы.

Семантическая сеть – это граф, дуги которого есть отношения между вершинами (значениями). Семантические сети появились при решении задач разбора и понимания смысла естественного языка [7]. Число отношений, используемых в конкретных семантических сетях, может быть различным.

Концептуальную (инфологическую) модель можно записать в виде формализованного текста [6]. Это менее наглядно, чем ER-диаграмма. Но часто более удобно для “черновых” вариантов модели и для больших предметных областей.

Исходя из теории и предметной области можно прийти к сущности «Кинотеатр», которая будет хранить всю информацию кинотеатра. Атрибуты у сущности будут такие:

* атрибут «Фильм», который будет хранить всю информацию о фильме (название, возрастной рейтинг, стран производства, жанры, длительность);
* атрибут «Зал», который будет хранить информацию о зале (название, вместимость);
* атрибут «Место», который будет хранить информацию о месте (ряд, место);
* атрибут «НачалоСеанса», который хранит дату и время начала сеанса;
* атрибут «КонецСеанса», который хранит дату и время конца сеанса;
* атрибут «Билет», который хранит информацию о билете (стоимость, время покупки);
* атрибут «Меню», который хранит информацию о меню (название, цена);
* атрибут «Покупка», который храннит информацию о покупке (итоговая цена, время покупки);
* атрибут «Скидка», который хранит информацию о скидке (название, начало скидки, конец скидки);
* атрибут «ПримСкидка», который хранит информацию о примененной скидке (процент скидки, сумма скидки у покупки).

3. Даталогическое проектирование

Даталогическое проектирование базы данных – это процесс разработки структуры базы данных на основе логической модели, которая была создана на этапе логического проектирования. Основная цель даталогического проектирования - это создание конкретной реализации базы данных, которая удовлетворяет требованиям предметной области и логической модели.

На этом этапе определяется структура таблиц, определяются отношения между таблицами, атрибуты и ограничения целостности [8]. В процессе даталогического проектирования разрабатываются схемы таблиц, описывающие структуру хранения данных, а также индексы, предназначенные для ускорения запросов к базе данных.

Кроме того, на этом этапе определяются типы данных и форматы хранения данных в базе данных. В процессе даталогического проектирования учитываются требования к производительности базы данных и оптимизации запросов. От успешности даталогического проектирования зависит эффективность работы базы данных и возможность эффективного хранения, поиска и обработки данных.

Конечным результатом даталогического проектирования является описание логической структуры базы данных на языке описания данных. Одним из ключевых аспектов даталогического проектирования является нормализация.

3.1 Нормализация

Реляционная база данных (РБД) - это тип базы данных, основанный на реляционной модели данных, предложенной Эдгаром Коддом в 1970 году [9]. Реляционная база данных представляет собой совокупность таблиц (отношений), связанных между собой отношениями. Каждая таблица содержит набор столбцов, определяющих атрибуты объектов, и набор строк, представляющих конкретные экземпляры этих объектов. Каждая строка в таблице представляет собой кортеж значений, соответствующих каждому из атрибутов.

Реляционная модель данных обладает рядом преимуществ перед другими моделями данных. Она упрощает процесс создания, обновления и управления данными, облегчает выполнение запросов и предоставляет более гибкий и удобный способ доступа к данным. Кроме того, реляционная модель является математически строгой, что позволяет ее применять для формального описания и анализа данных.

Отношения между таблицами могут быть установлены с помощью ключей, которые позволяют связать данные из разных таблиц [9].

Нормализация базы данных - это процесс проектирования структуры базы данных с целью устранения избыточности и повышения ее эффективности. Цель нормализации - это разбиение таблиц на более мелкие, связанные между собой логически и обладающие свойством минимальной избыточности.

Основная идея нормализации заключается в том, что каждый атрибут таблицы должен зависеть только от первичного ключа, а не от других атрибутов. Чем выше степень нормализации, тем меньше повторений данных в таблицах, что в свою очередь улучшает производительность базы данных и снижает вероятность ошибок в данных.

Для реляционных баз данных существует несколько уровней нормализации:

* 1НФ требует, чтобы все значения атрибутов в таблице были атомарными, т.е. не могли быть декомпозированы на более мелкие части;
* 2НФ требует, чтобы каждый не ключевой атрибут зависел только от первичного ключа таблицы;
* 3НФ требует, чтобы каждый не ключевой атрибут зависел только от первичного ключа, а не от других не ключевых атрибутов;
* Форма Бойса-Кодда требует, чтобы все таблицы в базе данных были нормализованы до третьей нормальной формы (3НФ) и затем были рассмотрены для нормализации до 4НФ и 5НФ;
* 4НФ является продолжением третьей нормальной формы (3НФ) и требует дополнительного разделения данных в базе данных на отдельные таблицы. Она предназначена для решения проблем, связанных с многозначными зависимостями, которые могут возникать при использовании композитных ключей в таблицах;
* 5НФ требует, чтобы в базе данных не было многозначных зависимостей между неключевыми атрибутами, и чтобы все неключевые атрибуты зависели только от первичного ключа.

Однако, следует учитывать, что излишняя нормализация может привести к увеличению числа таблиц, что может затруднить выполнение запросов и анализ данных. Поэтому выбор уровня нормализации должен зависеть от конкретных требований предметной области и целей базы данных. В данном случае подойдут первые три нормальные формы.

Процесс нормализации реляционной базы данных начинается с ненормализованной таблицы.

Ненормализованная база данных – это база данных, в которой данные не разбиты на отдельные таблицы в соответствии с определенными правилами нормализации. В ненормализованной базе данных одна и та же информация может дублироваться в нескольких местах, что приводит к проблемам с целостностью данных и затрудняет их обновление и поддержку. В ненормализованной базе данных могут отсутствовать связи между таблицами, что делает невозможным выполнение сложных запросов к данным.

Ненормализованная база данных может быть создана по различным причинам, таким как нехватка времени или знаний у разработчиков, отсутствие опыта в проектировании базы данных, а также из-за исторически сложившейся структуры данных. Однако использование ненормализованной базы данных может привести к трудностям в дальнейшей разработке и обслуживании приложений, использующих эти данные.

Построение ненормализованной таблицы для предметной области «Кинотеатр» представлено в табл.1.

Таблица 1

Кинотеатр

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Фильм** | **Зал** | **Место** | **НачалоСеанса** | **КонецСеанса** | **Билет** | **Меню** | **Покупка** | **Скидка** | **ПримСкидка** |
| Начало 12 США Фантастика 02:30:00 | IMAX 200 | 1 1 | 2025-06-01 19:00 | 2025-06-01 21:30 | 15 2025-05-20 12:00 | Попкорн 5 | 18 2025-05-20 12:05 | Летняя распродажа 2025-05-01 2025-06-01 | 10 2 |
| Титаник 12 США Драма 02:35:00 | Обычный 150 | 2 5 | 2025-06-02 18:00 | 2025-06-02 21:15 | 12 2025-05-21 14:00 | Cola 3 | 15 2025-05-20 12:05 | Летняя распродажа 2025-05-01 2025-06-01 | 5 2 |
| Аватар 12 США Приключения 02:22:00 | 3D зал 180 | 3 10 | 2025-06-03 17:00 | 2025-06-03 20:00 | 14 2025-05-22 15:00 | Начос 4 | 16 2025-05-20 12:05 | Вторник 20 2025-05-24 2025-06-24 | 15 2 |
| Матрица 18 США  Боевик  02:20:00 | Нео 120 | 4 2 | 2025-06-04 20:00 | 2025-06-04 22:30 | 13 2025-05-23 18:00 | Хот-дог 6 | 17,5 2025-05-20 12:05 | Студ среда 2025-05-24 2025-09-24 | 20 2 |
| Интерстеллар 16 США Фантастика 02:40:00 | Космос 210 | 5 8 | 2025-06-05 16:00 | 2025-06-05 19:00 | 16 2025-05-24 10:00 | Комбо 8 | 20 2025-05-20 12:05 | Летняя распродажа 2025-05-01 2025-06-01 | 12 2 |

После того, как создана ненормализованная таблица, стоит провести процесс нормализации к первой нормальной форме.

Первая нормальная форма (1НФ) - это базовое правило реляционной модели данных, которое гарантирует, что каждый атрибут таблицы содержит только простые, атомарные значения, то есть значения, которые невозможно разделить на более мелкие компоненты [10].

Другими словами, в первой нормальной форме каждый столбец таблицы содержит только одно значение, а не несколько значений, разделенных запятыми или другими разделителями. Например, в таблице, содержащей данные о фильмах, каждый столбец должен содержать только один тип данных, такой как название, жанр или длительность, а не комбинацию этих данных.

Соблюдение первой нормальной формы является важным шагом для создания хорошо структурированных баз данных, которые легко поддерживать и использовать.

Соблюдение первой нормальной формы облегчает выполнение запросов к базе данных, повышает ее производительность и облегчает ее поддержку и модификацию в будущем. Суть нормализации первой начальной формы состоит в следующем:

* поле Фильм разделить на пять полей: НазваниеФильма, ВозрастнойРейтинг, Длительность, Страна, Жанр;
* поле Зал разделить на два поля: НазваниеЗала, Вместимость;
* поле Место разделить на два поля: Ряд, Место;
* поле Билет разделить на два поля: ЦенаБилета, ВремяПокупкиБилета;
* поле Меню разделить на три поля: НазваниеПродукта, ЦенаПродукта;
* поле Скидка разделить на три поля: НазваниеСкидки, НачалоСкидки, КонецСкидки;
* поле Покупка на два поля: ИтоговаяЦена, ВремяПокупки;
* поле ПримСкидка на два полей: ПроцентСкидки, СуммаСкидки.

Результат деления неатомарных полей представлен в табл. 2.

При разработке предыдущей формы можно сделать вывод, что в первой нормальной форме удалось избавиться от избыточности лишь частично.

Таблица 2

Кинотеатр

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Название** | **ВозрастнойРейтинг** | **Длительность** | **Страна** | **Жанр** | **НазваниеЗала** | **Вместимость** | **Ряд** | **Место** | **НачалоСеанса** | **КонецСеанса** | **ЦенаБилета** |  |
| Начало | 12 | 02:20:00 | США | Фантастика | IMAX | 200 | 1 | 1 | 2025-06-01 19:00 | 2025-06-01 21:30 | 15 |  |
| Титаник | 12 | 02:00:00 | США | Драма | Обычный | 150 | 2 | 5 | 2025-06-02 18:00 | 2025-06-02 21:15 | 12 |  |
| Аватар | 12 | 02:30:00 | США | Приключения | 3D зал | 180 | 3 | 10 | 2025-06-03 17:00 | 2025-06-03 20:00 | 14 |  |
| Матрица | 18 | 02:30:00 | США | Боевик | Нео | 120 | 4 | 2 | 2025-06-04 20:00 | 2025-06-04 22:30 | 13 |  |
| Интерстеллар | 16 | 02:10:00 | США | Фантастика | Космос | 210 | 5 | 8 | 2025-06-05 16:00 | 2025-06-05 19:00 | 16 |  |

Продолжение табл.2

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ВремяПокупкиБилета** | **НазваниеПродукта** | **ЦенаПродукта** | **ИтоговаяЦена** | **ВремяПокупки** | **НазваниеСкидки** | **ПроцентСкидки** | **НачалоСкидки** | **КонецСкидки** | **Сумма-Скидки** |
| 2025-05-20 12:00 | Попкорн | 5 | 18 | 2025-05-20 12:05 | Летняя распродажа | 10 | 2025-05-01 | 2025-06-30 | 2 |
| 2025-05-21 14:00 | Cola | 3 | 15 | 2025-05-21 14:05 | Летняя распродажа | 5 | 2025-05-15 | 2025-06-15 | 1,5 |
| 2025-05-22 15:00 | Начос | 4 | 16 | 2025-05-22 15:05 | Вторник 20 | 15 | 2025-05-10 | 2025-06-20 | 2,5 |
| 2025-05-23 18:00 | Хот-дог | 6 | 17,5 | 2025-05-23 18:05 | Студ среда | 20 | 2025-05-05 | 2025-06-05 | 3,5 |
| 2025-05-24 10:00 | Комбо | 8 | 20 | 2025-05-24 10:05 | Летняя распродажа | 12 | 2025-05-01 | 2025-06-10 | 4 |

После проектирования первой нормальной формы создается вторая нормальная форма.

Вторая нормальная форма (2НФ) – это правило реляционной модели данных, которое гарантирует, что каждый атрибут сущности зависит от ее первичного ключа [10]. Применительно к базе данных, чтобы удовлетворять второй нормальной форме, данные должны быть организованы таким образом, чтобы каждый столбец таблицы зависел только от ее первичного ключа, а не от любого другого набора атрибутов.

Таким образом, сущность «Кинотеатр» можно разделить на восемь сущностей, а именно:

* атрибуты, содержащие информацию о фильме, выносятся в сущность «Фильмы»;
* атрибуты, содержащие информацию о сеансе, выносятся в сущность «Сеансы»;
* атрибуты, содержащие информацию о зале, выносятся в сущность «Залы»;
* атрибуты, содержащие информацию о местах, выносятся в сущность «Места»;
* атрибуты, содержащие информацию о билетах, выносятся в сущность «Билеты»;
* атрибуты, содержащие информацию о меню, выносятся в сущность «Меню»;
* атрибуты, содержашие информацию о покупках, выносятся в сущность «Покупки»;
* атрибуты, содержащие информацию о скидках, выносятся в сущность в «Скидки».

Теперь у каждой сущности все ее атрибуты зависят от первичного ключа, а сами сущности связаны между собой, а именно:

* сущность «Места» связана с сущностью «Залы» через атрибут «НомерЗала»;
* сущность «Сеансы» связан с сущностями «Фильмы» и «Залы» через атрибуты «НомерФильма» и «НомерЗала»;
* сущность «Билеты» связана с сущностями «Сеансы» и «Места» через атрибуты «НомерСеанса» и «НомерМеста»;
* сущность «Покупки» связана с сущностями «Билеты», «Меню» и «Скидки» через атрибуты «НомерБилета», «НомерПродукта» и «НомерСкидки».

Результат нормализации в 2НФ представлен в табл. 3-10.

Таблица 3

Фильмы

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **НомерФильма(PK)** | **Название** | **ВозрастнойРейтинг** | **Длительность** | **Страна** | **Жанр** |
| 1 | Начало | 12+ | 00:02:00 | США | Фантастика |
| 2 | Титаник | 12+ | 01:35:00 | США | Драма |
| 3 | Аватар | 12+ | 02:20:00 | США | Приключения |
| 4 | Матрица | 18+ | 02:10:00 | США | Боевик |
| 5 | Интерстеллар | 16+ | 02:30:00 | США | Фантастика |

Таблица 4

Залы

|  |  |  |
| --- | --- | --- |
| **НомерЗала(PK)** | **Название** | **Вместимость** |
| 1 | IMAX | 200 |
| 2 | Обычный | 150 |
| 3 | 3D зал | 180 |
| 4 | Neo | 120 |
| 5 | Космос | 210 |

Таблица 5

Сеансы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **НомерСеанса(PK)** | **НомерФильма** | **НомерЗала** | **НачалоСеанса** | **КонецСеанса** |
| 1 | 1 | 1 | 2025-06-01 19:00 | 2025-06-01 21:00 |
| 2 | 2 | 1 | 2025-06-02 21:00 | 2025-06-02 23:15 |
| 3 | 3 | 1 | 2025-06-03 17:00 | 2025-06-03 20:00 |
| 4 | 4 | 4 | 2025-06-04 20:00 | 2025-06-04 22:00 |
| 5 | 5 | 5 | 2025-06-05 16:00 | 2025-06-05 19:00 |

Таблица 6

Места

|  |  |  |
| --- | --- | --- |
| **НомерМеста(PK)** | **Ряд** | **Место** |
| 1 | 1 | 1 |
| 1 | 1 | 2 |
| 1 | 2 | 1 |
| 2 | 2 | 5 |
| 3 | 3 | 10 |
| 4 | 4 | 2 |
| 5 | 5 | 8 |

Таблица 7

Билеты

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **НомерБилета(PK)** | **НомерСеанаса** | **НомерМеста** | **Цена** | **ВремяПокупки** |
| 1 | 1 | 1 | 15 | 2025-05-20 12:00 |
| 2 | 1 | 2 | 12 | 2025-05-21 14:00 |
| 3 | 2 | 3 | 13 | 2025-05-22 15:00 |
| 4 | 4 | 4 | 14 | 2025-05-23 18:00 |
| 5 | 3 | 5 | 16 | 2025-05-24 10:00 |

Таблица 8

Меню

|  |  |  |  |
| --- | --- | --- | --- |
| **НомерПродукта(PK)** | **Название** | **Цена** | **ФлагДоступности** |
| 1 | Попкорн | 5 | ИСТИНА |
| 2 | Сола | 3 | ИСТИНА |
| 3 | Начос | 4 | ЛОЖЬ |
| 4 | Хот-дог | 6 | ИСТИНА |
| 5 | Комбо | 8 | ИСТИНА |

Таблица 9

Покупки

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Номер-Покупки(PK)** | **Номер-Билета** | **Номер-Продукта** | **Номер-Скидки** | **Итоговая-Цена** | **Время-Покупки** | **Сумма-Скидки** |
| 1 | 1 | 5 | 1 | 20.7 | 2025-05-20 12:05 | 2.3 |
| 2 | 2 | 1 | ПУСТО | 17 | 2025-05-21 14:05 | ПУСТО |
| 3 | 3 | ПУСТО | ПУСТО | 13 | 2025-05-22 15:05 | ПУСТО |
| 4 | ПУСТО | 4 | ПУСТО | 6 | 2025-05-23 18:05 | ПУСТО |
| 5 | 5 | ПУСТО | 3 | 12.8 | 2025-05-24 10:05 | 3.2 |

Таблица 10

Скидки

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Номер-Скидки(PK)** | **Название-Скидки** | **Процент-Скидки** | **Начало-Скидки** | **Конец-Скидки** |
| 1 | Летняя распродажа | 10 | 2025-05-01 | 2025-06-30 |
| 2 | Летняя распродажа | 10 | 2025-05-01 | 2025-06-30 |
| 3 | Вторник 20 | 15 | 2025-05-01 | 2025-06-20 |
| 4 | Супер среда | 20 | 2025-05-01 | 2025-06-15 |
| 5 | Летняя распродажа | 12 | 2025-05-01 | 2025-06-10 |

После создания второй нормальной формы идет последний процесс нормализации для данной предметной области – третья нормальная форма.

Третья нормальная форма (3НФ) – это правило реляционной модели данных, которое гарантирует, что каждый неключевой атрибут сущности зависит только от ее первичного ключа или других неключевых атрибутов, но не от комбинации этих атрибутов [10].

Применительно к базе данных, чтобы удовлетворять третьей нормальной форме, данные должны быть организованы таким образом, чтобы все неключевые поля зависели только от первичного ключа или других неключевых полей в той же таблице.

Соблюдение третьей нормальной формы помогает уменьшить дублирование данных, улучшает структуру базы данных и облегчает ее поддержку и модификацию в будущем.

У сущности «Фильмы» есть несколько неключевых атрибутов, которые независят от первичного ключа, а именно «Жанр», «Страна» и «ВозрастнойРейтинг». Поэтому их можно вынести в отдельные сущности, как «Жанры», «Страны» и «ВозрастныеРейтинги». Связаны с «Фильмы» они будут через внешние ключи в сущности, а именно:

* вместо атрибута «Страна» будет внешний ключ «НомерСтраны»;
* вместо атрибута «ВозрастнойРейтинг» будет внешний ключ «НомерРейтинга»;
* для связи «Фильмы» и «Жанры» будет отдельная сущность «ФильмЖанры», в которой будет два внешних ключа «НомерФильма» и «НомерЖанра»;

Каждая из таблиц 3НФ связана как минимум с одной из таблиц по первичному ключу.

Результат приведения к 3НФ представлен в табл. 11-23.

Таблица 11

Фильмы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Номер-Фильма(PK)** | **Название** | **Номер-Рейтинга(FK)** | **Длительность** | **Номер-Страны(FK)** |
| 1 | Начало | 3 | 00:02:00 | 1 |
| 2 | Титаник | 4 | 01:35:00 | 1 |
| 3 | Аватар | 3 | 02:20:00 | 1 |
| 4 | Матрица | 5 | 02:10:00 | 1 |
| 5 | Интерстеллар | 3 | 02:30:00 | 1 |

Таблица 12

Страны

|  |  |
| --- | --- |
| **НомерСтраны(PK)** | **Название** |
| 1 | США |
| 2 | Россия |
| 3 | Испания |
| 4 | Франция |
| 5 | Южная Корея |

Таблица 13

ВозрастныеРейтинги

|  |  |
| --- | --- |
| **НомерРейтинга(PK)** | **Название** |
| 1 | 0+ |
| 2 | 6+ |
| 3 | 12+ |
| 4 | 16+ |
| 5 | 18+ |

Таблица 14

Жанры

|  |  |
| --- | --- |
| **НомерЖанра(PK)** | **Название** |
| 1 | Боевик |
| 2 | Драма |
| 3 | Научная фантастика |
| 4 | Романтика |
| 5 | Приключения |

Таблица 15

ФильмЖанры

|  |  |
| --- | --- |
| **НомерФильма(FK)** | **НомерЖанра(FK)** |
| 1 | 3 |
| 2 | 2 |
| 3 | 5 |
| 4 | 1 |
| 5 | 3 |

Таблица 16

Залы

|  |  |  |
| --- | --- | --- |
| **НомерЗала(PK)** | **Название** | **Вместимость** |
| 1 | Зал А | 100 |
| 2 | Зал Б | 150 |
| 3 | Зал В | 120 |
| 4 | Зал Г | 130 |
| 5 | Зал Д | 110 |

Таблица 17

Места

|  |  |  |  |
| --- | --- | --- | --- |
| **НомерМеста(PK)** | **НомерЗала(FK)** | **Ряд** | **Место** |
| 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 2 |
| 3 | 2 | 2 | 1 |
| 4 | 2 | 2 | 2 |
| 5 | 3 | 3 | 1 |

Таблица 18

Сеансы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Номер-Сеанса(PK)** | **Номер-Фильма(FK)** | **Номер-Зала(FK)** | **НачалСеанса** | **КонецСеанса** |
| 1 | 1 | 1 | 2025-05-01 00:00:00 | 2025-05-01 02:00:00 |
| 2 | 2 | 2 | 2025-05-02 00:00:00 | 2025-05-02 02:00:00 |
| 3 | 3 | 3 | 2025-05-03 00:00:00 | 2025-05-03 02:00:00 |
| 4 | 4 | 4 | 2025-05-04 00:00:00 | 2025-05-04 02:00:00 |
| 5 | 5 | 5 | 2025-05-05 00:00:00 | 2025-05-05 02:00:00 |

Таблица 19

Билеты

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Номер-Билета(PK)** | **Номер-Сеанса(FK)** | **Номер-Места(FK)** | **Цена** | **ВремяПокупки** |
| 1 | 1 | 1 | 10 | 2025-05-01 10:00:00 |
| 2 | 2 | 2 | 12 | 2025-05-01 11:00:00 |
| 3 | 3 | 3 | 11,5 | 2025-05-01 12:00:00 |
| 4 | 4 | 4 | 9 | 2025-05-01 13:00:00 |
| 5 | 5 | 5 | 10,5 | ПУСТО |

Таблица 20

Меню

|  |  |  |  |
| --- | --- | --- | --- |
| **Номер-Продукта(PK)** | **Название** | **Цена** | **Флаг-Доступности** |
| 1 | Попкорн | 5 | ИСТИНА |
| 2 | Газировка | 3 | ИСТИНА |
| 3 | Начос | 4,5 | ЛОЖЬ |
| 4 | Конфеты | 2 | ИСТИНА |
| 5 | Хот-дог | 6 | ИСТИНА |

Таблица 21

Покупки

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Номер-Покупки(PK)** | **Номер-Билета(FK)** | **Номер-Продукта(FK)** | **Итоговая-Цена** | **ВремяПокупки** |
| 1 | 1 | 5 | 1 | 2025-05-01 10:00:00 |
| 2 | 2 | 1 | ПУСТО | 2025-05-01 11:00:00 |
| 3 | 3 | ПУСТО | ПУСТО | 2025-05-01 12:00:00 |
| 4 | ПУСТО | 4 | ПУСТО | 2025-05-01 13:00:00 |
| 5 | 5 | ПУСТО | 3 | 2025-05-01 14:00:00 |

Таблица 22

Скидки

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Номер-Скидк(PK)** | **Название** | **Процент-Скидки** | **НачалоСкидки** | **КонецСкидки** |
| 1 | Весенняя распродажа | 10 | 2025-04-01 00:00:00 | 2025-04-10 00:00:00 |
| 2 | Специальное предложение выходного дня | 15 | 2025-04-08 00:00:00 | 2025-04-17 00:00:00 |
| 3 | Праздничная скидка | 20 | 2025-04-15 00:00:00 | 2025-04-24 00:00:00 |
| 4 | Вечернее предложение | 5 | 2025-04-22 00:00:00 | 2025-05-01 00:00:00 |
| 5 | Скидка для студентов | 12 | 2025-04-29 00:00:00 | 2025-05-08 00:00:00 |

Таблица 23

ПрименныеСкидки

|  |  |  |  |
| --- | --- | --- | --- |
| **Номер-Применения(PK)** | **Номер-Покупки(FK)** | **Номер-Скидки(FK)** | **Процент-Скидки** |
| 1 | 1 | 1 | 1,5 |
| 2 | 2 | 2 | 2,25 |
| 3 | 3 | 3 | 3,2 |
| 4 | 4 | 4 | 0,55 |
| 5 | 5 | 5 | 1,98 |

3.2 Логическая модель

Логическая модель - модель базы данных, которая описывает структуру данных и их отношения в базе данных на логическом уровне [10]. Она определяет сущности, их атрибуты и связи между ними.

Логическая модель является промежуточным этапом между концептуальной и физической моделями. В концептуальной модели определяются сущности и их связи на высоком уровне абстракции, не привязанные к конкретной реализации базы данных.

В логической модели определяются сущности и их атрибуты, а также связи между сущностями [6]. Она описывает, как данные будут храниться и как они будут использоваться в приложениях. Логическая модель является основой для создания физической модели и базы данных.

Логическая модель может быть представлена в виде схемы базы данных, которая включает в себя таблицы, их столбцы и связи между таблицами. Она может быть представлена в графическом виде с использованием диаграммы ER (Entity-Relationship), которая показывает сущности, их атрибуты и связи между ними.

Логическая модель является важным инструментом в разработке базы данных, поскольку она помогает определить, какие данные нужно хранить, как их нужно хранить и как они будут использоваться в приложениях.

Следующим этапом проектирования базы данных кинотеатра является создание диаграммы, называемой сущность-связь.

ER-диаграммы - это графическое представление логической модели данных, которое используется для описания сущностей и их отношений в базе данных [9]. ER-диаграммы состоят из сущностей, связей и атрибутов. Цель ER-диаграмм заключается в описании логической структуры данных в базе данных и отображении отношений между сущностями, атрибутами и связями. Они помогают разработчикам баз данных создавать модель, которая отражает бизнес-процессы организации, и устанавливать связи между объектами данных. ER-диаграммы помогают разработчикам баз данных лучше понять бизнес-требования, что позволяет им проектировать и создавать более эффективные и удобные базы данных. Они могут быть использованы для установления связей между таблицами в базе данных, чтобы гарантировать целостность данных и предотвратить дублирование информации [9]. В целом, ER-диаграммы представляют собой важный инструмент в разработке баз данных и позволяют разработчикам баз данных лучше понимать, проектировать и создавать базы данных, которые отвечают бизнес-требованиям и обеспечивают эффективную работу приложений.

В ER-диаграммах существует три основных типа связей между сущностями: один-к-одному, один-ко-многим и многие-ко-многим [9].

1. Один-к-одному (One-to-One) - это отношение, когда каждая запись в таблице A имеет связанную запись в таблице B и наоборот.
2. Один-ко-многим (One-to-Many) - это отношение, когда каждая запись в таблице A может иметь несколько связанных записей в таблице B, но каждая запись в таблице B имеет только одну связанную запись в таблице A.
3. Многие-ко-многим (Many-to-Many) - это отношение, когда каждая запись в таблице A может иметь несколько связанных записей в таблице B, и каждая запись в таблице B может иметь несколько связанных записей в таблице A. Для реализации этого отношения требуется дополнительная таблица-связь, которая содержит первичные ключи обеих таблиц.

Каждый тип связи в ER-диаграмме имеет свои особенности и ограничения, которые необходимо учитывать при проектировании базы данных. Правильное определение типов связей и их атрибутов помогает обеспечить эффективное и надежное хранение данных в базе данных.

Для проектирования ER-диаграмм в данном проекте выбран инструмент ERwin Data Modeler. ERwin является мощным CASE-средством, которое поддерживает полный цикл проектирования баз данных - от концептуального моделирования до физической реализации.

Ключевые преимущества выбора ERwin включают:

* поддержка всех этапов проектирования - от создания логической модели до генерации физической схемы БД;
* автоматическая проверка соответствия нормальным формам, что особенно важно при выполнении нормализации;
* возможность генерации SQL-скриптов для различных СУБД, включая Oracle, Access и т.д.;
* удобный графический интерфейс для создания и редактирования ER-диаграмм с поддержкой всех типов связей;
* встроенные средства документирования, позволяющие автоматически генерировать отчеты о структуре базы данных.

Использование ERwin значительно ускоряет процесс проектирования базы данных кинотеатра и помогает избежать распространенных ошибок на этапе логического моделирования. Инструмент особенно полезен при работе со сложными структурами данных, позволяя наглядно отображать связи между сущностями и контролировать целостность будущей базы данных.

Логическое представление представлено на рис. 1.

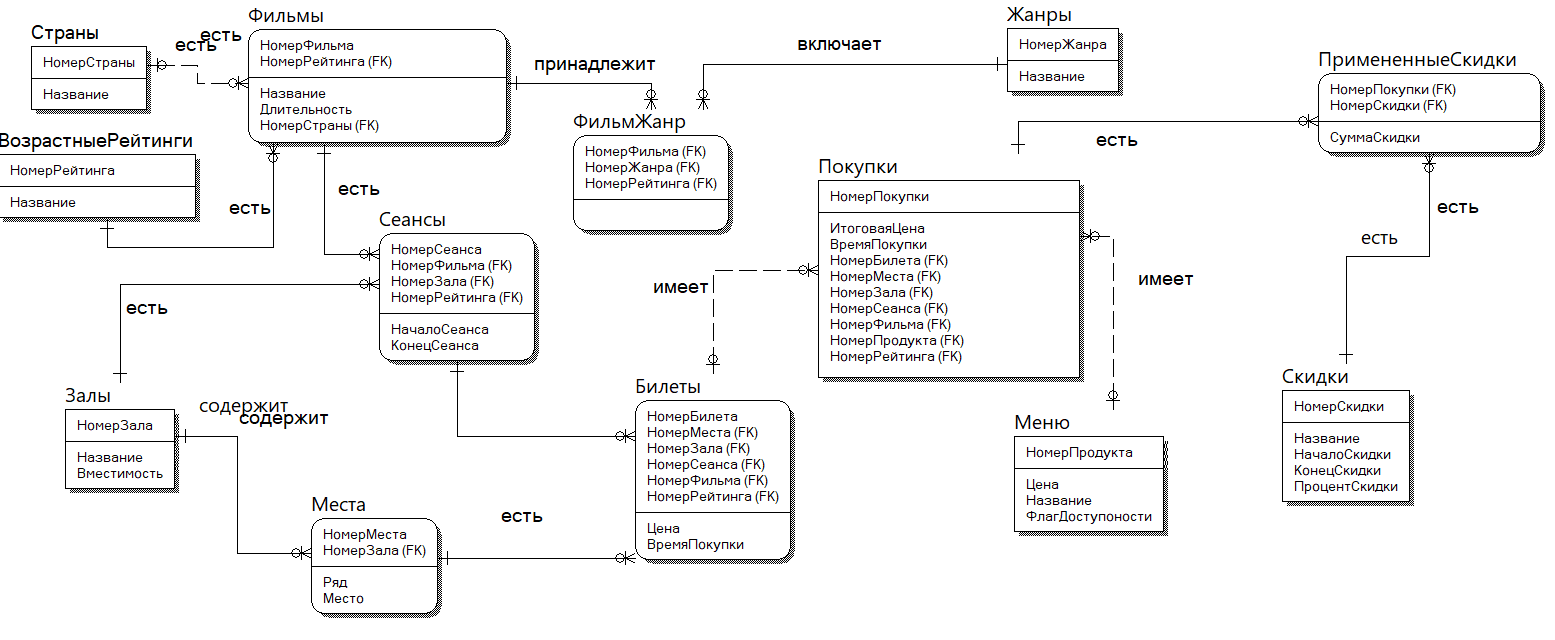


Рис.1. Логическое представление предметной области «Кинотеатр»

3.3 Физическая модель

Физическая модель базы данных представляет собой реализацию логической модели в виде конкретных таблиц и связей между ними [6]. Она описывает, как данные будут храниться на диске и как они будут связаны между собой.

Физическая модель базы данных важна для создания и оптимизации базы данных. Она позволяет определить структуру таблиц и связей между ними, а также настроить индексы и ограничения целостности, чтобы обеспечить быстрый доступ к данным и защитить их от ошибок и нарушений.

Физическая модель базы данных является одним из основных элементов, используемых при проектировании и создании СУБД (системы управления базами данных). СУБД - программное обеспечение, которое обеспечивает управление базами данных, включая создание, хранение, поиск и обработку данных.

Физическая модель базы данных определяет конкретные таблицы, столбцы, связи и другие элементы, которые будут использоваться для хранения данных в СУБД. Она также определяет ограничения целостности, индексы, триггеры и другие элементы, которые обеспечивают правильность и надежность данных.

СУБД используют физическую модель базы данных для создания и управления физическим хранением данных. СУБД используют специальные алгоритмы для организации и хранения данных на диске, чтобы обеспечить быстрый доступ к данным и минимизировать потерю информации при сбоях или сбое оборудования. Кроме того, СУБД позволяют пользователям выполнять запросы к данным и обрабатывать их, используя SQL (Structured Query Language) и другие языки запросов.

Физическая модель базы данных является важным компонентом любой СУБД, поскольку она определяет структуру и организацию данных. Она также обеспечивает эффективность и производительность базы данных, что является важным фактором для многих приложений и систем, которые используют базы данных для хранения и обработки данных.

Для базы данных кинотеатра была выбрана СУБД Microsoft Access в силу следующих преимуществ:

* простота использования - Access предоставляет интуитивно понятный графический интерфейс, что упрощает процесс создания и администрирования базы данных без необходимости глубоких технических знаний;
* интеграция с офисными продуктами - тесная взаимосвязь с другими приложениями пакета Microsoft Office (Excel, Word) облегчает импорт/экспорт данных и создание отчетов.
* достаточная функциональность - для проектов среднего масштаба, таких как база данных кинотеатра, Access предоставляет все необходимые инструменты для хранения и обработки данных без избыточных функций промышленных СУБД;
* минимальные требования к ресурсам - не требует значительных вычислительных мощностей и сложной настройки серверного окружения;
* широкое распространение - доступность и популярность в образовательной среде делают Access удобным выбором для учебных проектов.

Данные характеристики делают Microsoft Access оптимальным решением для реализации физической модели базы данных в рамках данного курсового проекта.

Подробное описание физической модели представлено на табл. 24-36 и на рис. 2.

Таблица 24

«Фильмы»

|  |  |
| --- | --- |
| **Атрибут** | **Тип данных** |
| НомерФильма | Counter |
| Название | Text(50) |
| НомерРейтинга | Long Integer |
| Длительность | Text(8) |
| НомерСтраны | Long Integer |

Таблица 25

Страны

|  |  |
| --- | --- |
| **Атрибут** | **Тип данных** |
| НомерСтраны | Counter |
| Название | Text(40) |

Таблица 26

ВозрастныеРейтинги

|  |  |
| --- | --- |
| **Атрибут** | **Тип данных** |
| НомерРейтинга | Counter |
| Название | Text(3) |

Таблица 27

Жанры

|  |  |
| --- | --- |
| **Атрибут** | **Тип данных** |
| НомерЖанра | Counter |
| Название | Text(25) |

Таблица 28

ФильмЖанры

|  |  |
| --- | --- |
| **Атрибут** | **Тип данных** |
| НомерФильма | Long Integer |
| НомерЖанра | Long Integer |

Таблица 29

Залы

|  |  |
| --- | --- |
| **Атрибут** | **Тип данных** |
| НомерЗала | Counter |
| Название | Text(20) |
| Вместимость | Integer |

Таблица 30

Места

|  |  |
| --- | --- |
| **Атрибут** | **Тип данных** |
| НомерМеста | Counter |
| НомерЗала | Long Integer |
| Ряд | Integer |
| Место | Integer |

Таблица 31

Сеансы

|  |  |
| --- | --- |
| **Атрибут** | **Тип данных** |
| НомерСеанса | Counter |
| НомерФильма | Long Integer |
| НомерЗала | Long Integer |
| НачалаСеанса | Date/Time |
| КонецСеанса | Date/Time |

Таблица 32

Билеты

|  |  |
| --- | --- |
| **Атрибут** | **Тип данных** |
| НомерБилета | Counter |
| НомерСеанса | Long Integer |
| НомерМеста | Long Integer |
| Цена | Long Integer |
| ВремяПокупки | Date/Time |

Таблица 33

Меню

|  |  |
| --- | --- |
| **Атрибут** | **Тип данных** |
| НомерПродукта | Counter |
| Название | Text(30) |
| Цена | Long Integer |
| ФлагДоступности | Bit |

Таблица 34

Покупки

|  |  |
| --- | --- |
| **Атрибут** | **Тип данных** |
| НомерПокупки | Counter |
| НомерБилета | Long Integer |
| НомерПродукта | Long Integer |
| ИтоговаяЦена | Long Integer |
| ВремяПокупки | Date/Time |

Таблица 35

Скидки

|  |  |
| --- | --- |
| **Атрибут** | **Тип данных** |
| НомерСкидки | Counter |
| Название | Text(50) |
| ПроцентСкидки | Small Integer |
| НачалоСкидки | Date/Time |
| КонецСкидки | Date/Time |

Таблица 36

ПримененныеСкидки

|  |  |
| --- | --- |
| **Атрибут** | **Тип данных** |
| НомерПокупки | Long Integer |
| НомерСкидки | Long Integer |
| СуммаСкидки | Long Integer |

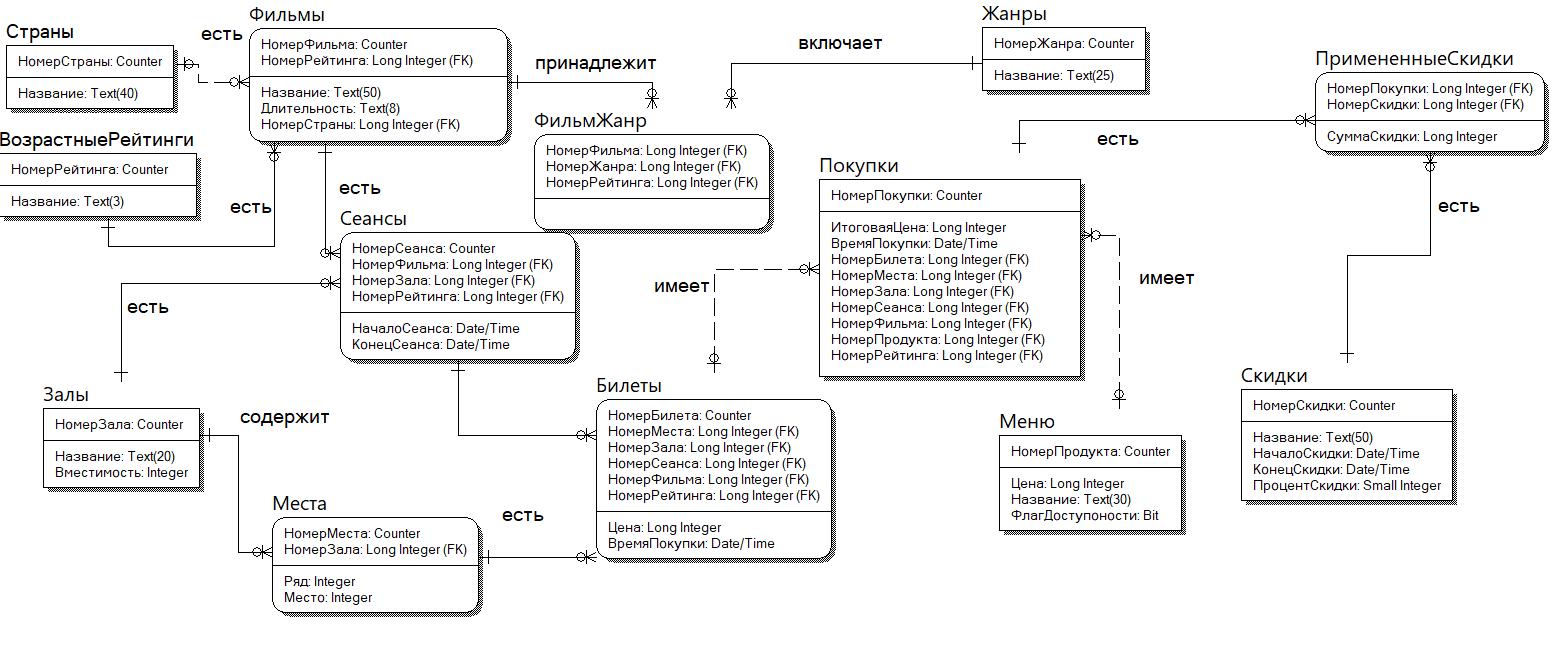


Рис. 2. Физическое представление предметной области «Кинотеатр»

1. Программирование на языке высокого уровня

Программа, выполняющая взаимодействие с базой данных СУБД Access, будет написана на **языке программирования высокого уровня C#**. **C# (си-шарп)** — это объектно-ориентированный язык программирования, разработанный компанией Microsoft как часть платформы .NET. Он предназначен для разработки широкого спектра приложений: от настольных программ и веб-приложений до мобильных и облачных решений. В отличие от C++, язык C# обладает высокой степенью абстракции и безопасностью типов, упрощенной моделью памяти, автоматическим управлением памятью (сборка мусора), а также мощной поддержкой **объектно-ориентированного программирования**: классы, наследование, инкапсуляция, интерфейсы, полиморфизм. C# является одним из основных языков для создания Windows-приложений, включая формы (WinForms), WPF, ASP.NET и др.

Программа будет разрабатываться в **Microsoft Visual Studio 2022**, которая является одной из самых мощных и популярных интегрированных сред разработки (IDE).

Причины выбора Visual Studio:

* **поддержка языка C# и платформы .NET**: Полная интеграция с WinForms, Entity Framework, ADO.NET и другими технологиями .NET;
* **удобство разработки интерфейсов**: Используется визуальный редактор форм, где легко размещать элементы управления (такие как кнопки, таблицы и выпадающие списки);
* **интеграция с базами данных**: Простая работа с Access, SQL Server и другими СУБД;
* **мощные инструменты отладки и профилирования**: Возможность пошагового выполнения, точек останова, отслеживания переменных;
* **широкое сообщество и документация**: Большое количество примеров, расширений, библиотек.

База данных подключается с помощью команды, представленной на рис.3.

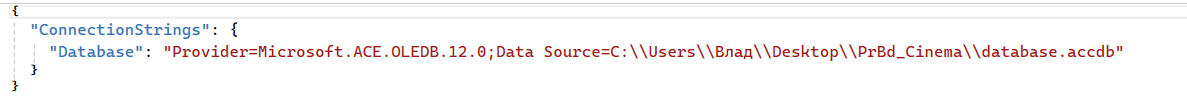


Рис.3. Строка подключения

Программа состоит из **одной главной формы (**MainForm**)**, которая включает:

* **8 вкладок (**ToolStrip**)**;
* таблицы: позволяет просматривать и редактировать данные в таблицах;
* запросы: выполняет предопределённые SQL-запросы по выбору пользователя;
* **взаимодействие с базой данных** реализовано через OleDbConnection, с использованием SQL-запросов и таблиц Access.

Элементы управления, использованные в программе:

|  |  |
| --- | --- |
| Элемент управления | Назначение |
| **Form** (MainForm) | Основное окно приложения |
| ToolStrip | Позволяет переключаться между вкладками “Медиатека”, “Расписание сеансов ”, “Касса”, “Управление залов”, “Меню”, “Акции” |
| **DataGridView** | Табличное отображение данных. Позволяет редактировать, удалять и сохранять записи. Поддерживает автоматическое обновление |
| **ComboBox** (comboTables, comboQueries) | Выпадающие списки для выбора таблицы и запроса |
| **Button** | Кнопки “Создать”, “Изменить”, “Удалить”, и другие для управления отображением и действиями |
| **Label** | Текстовые подсказки и заголовки рядом с элементами |
| **MessageBox** | Всплывающие окна с сообщениями, подсказками и предупреждениями |

Примеры использования перечисленных элементов упрвления в коде и работа с базой данных предсталены на рис. 4-7.

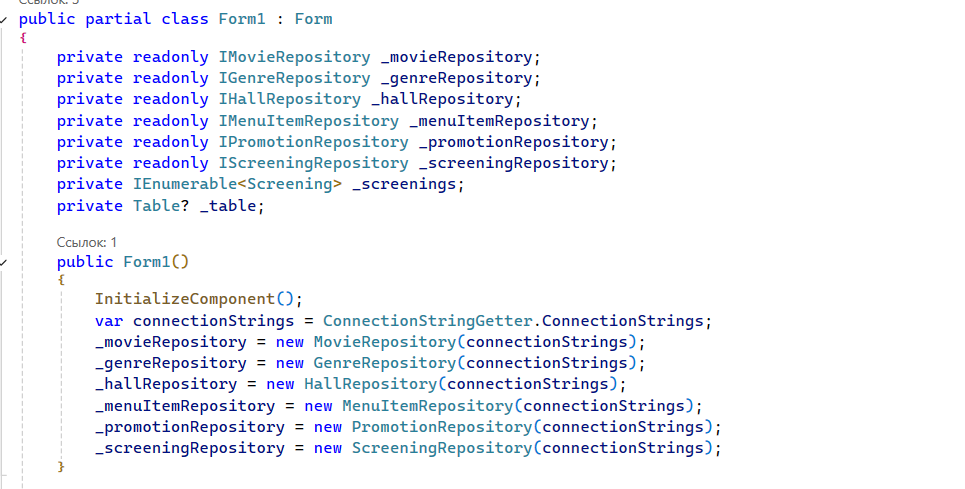


Рис. 4. Код элемента Form



Рис. 5. Пример обработки нажатия на одну из вкладок у ToolStrip и работы с DataGridView

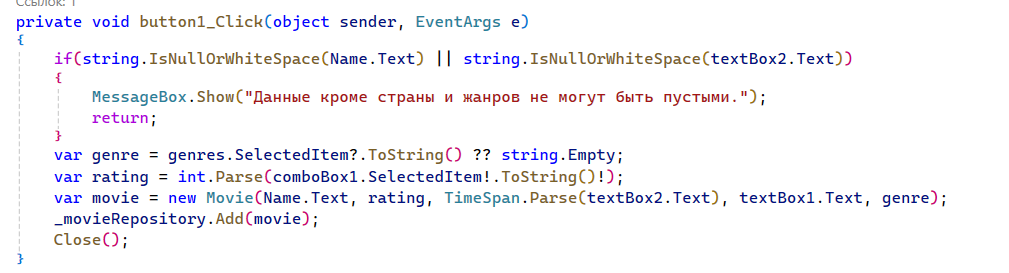


Рис. 6. Пример обработки нажатия кнопки и работы с ComboBox, MessageBox



Рис. 7. Пример работы с базой данных

Программа поддерживает:

* **просмотр и редактирование** всех таблиц в базе данных;
* **удаление записей** с отображением предупреждающих сообщений;
* **добавление новых записей**, за исключением справочных таблиц;
* **автоматическое сохранение изменений** при переходе между таблицами.

Интерфейс программы представлен на рис.8-11.

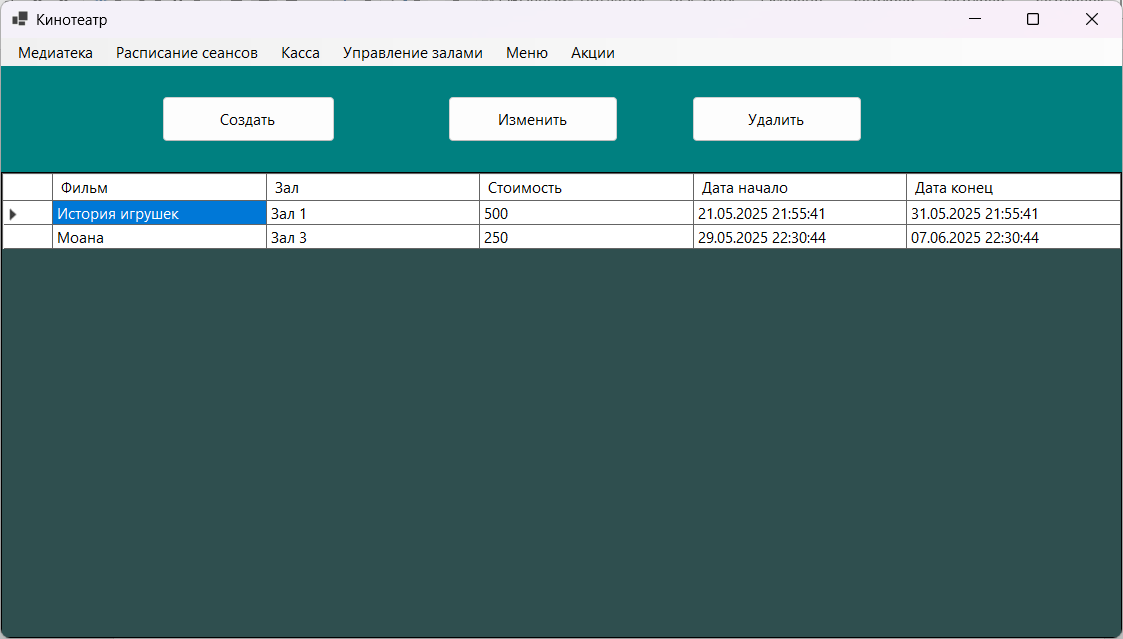


Рис. 8. Вкладка «Расписание сеансов»

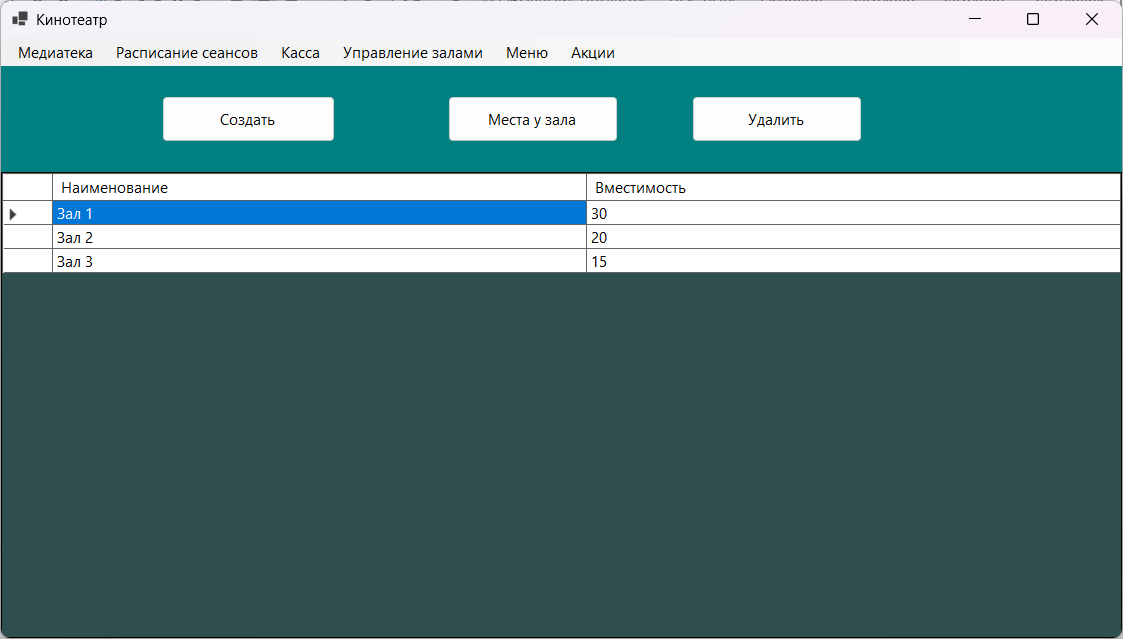


Рис. 9. Вкладка «Управление залами»

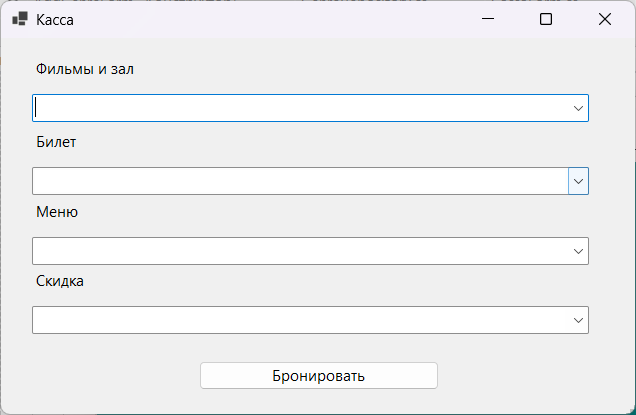


Рис. 10. Окно касса

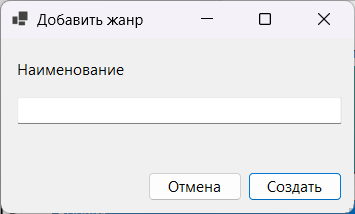


Рис. 11. Модальное окно «Добавить жанр»

Более детальное описание представлено в Руководстве пользователя (прил.2), а полный текст программы представлен в прил. 3.

Заключение

В ходе выполнения курсовой работы была спроектирована база данных «Кинотеатр» и программное обеспечение для взаимодействия с ней. Также были приобретены навыки программирования реляционной базы данных в среде Microsoft Visual Studio 2022 на языке высокого уровня C#.

В ходе работы:

* спроектирована реляционная база данных;
* осуществлена нормализация базы данных;
* написано программное обеспечение на языке C#, которое взаимодействует с базой данных;
* реализован ввод с клавиатуры, вывод на дисплей;
* реализована возможность изменения данных в таблицах;
* реализовано удаление;
* проверена работа программного кода.

Программное обеспечение «Кинотеатр» упрощает работу сотрудника кинотеатра и минимизирует вероятность ошибок в работе и позволит ввести отчетность.

# Список литературы

1. Соколов, Н. В. Применение многофакторного анализа для моделирования системы сокращения ручного труда и автоматизации / Н. В. Соколов, Е. В. Пятаев // Швейная промышленность. – 2007. – № 2. – С. 13-14. – EDN KVVKLB.

2. Semenova, E. I. The importance of high-performance modeling for data-intensive applications / E. I. Semenova, V. V. Khramkov // . – 2022. – No. 21. – P. 300-301. – EDN NMCQDP.

3. Коледин, С. Н. структура базы данных критериев оптимальности каталитических реакций / С. Н. Коледин, С. А. Габитов, К. Ф. Коледина // Математическое моделирование процессов и систем : материалы IX Международной молодежной научно-практической конференции, Стерлитамак, 30 октября – 01 2019 года. – Стерлитамак: Башкирский государственный университет, Стерлитамакский филиал, 2019. – С. 221-223. – EDN VQSKSA.

4. Раковская, Е. Е. Инфологическое проектирование базы данных оборудования АСУ ТП / Е. Е. Раковская // Современный взгляд на будущее науки : Сборник статей Международной научно-практической конференции, Челябинск, 28 октября 2015 года / Ответственный редактор: Сукиасян Асатур Альбертович. Том 2. – Челябинск: Общество с ограниченной ответственностью "ОМЕГА САЙНС", 2015. – С. 79-85. – EDN UMPCZF.

5. Попова, Г. Н. Интеграция баз данных При помощи семантической сети / Г. Н. Попова, Я. И. Торгашов, Н. А. Жабина // The World of Science Without Borders, 11 февраля 2022 года, 2022. – P. 317-320. – EDN WAEQIW.

6. Мукина, И. А. Проектирование инфологической, даталогической и физической моделей базы данных для хранения результатов расчета модифицированного метода анализа иерархий / И. А. Мукина, Д. С. Соловьев, Ю. В. Литовка // Наука сегодня: фундаментальные и прикладные исследования : Материалы международной научно-практической конференции. В 2-х частях, Вологда, 27 сентября 2017 года. Том Часть 1. – Вологда: ООО "Маркер", 2017. – С. 51-53. – EDN ZNGSQL.

7. Мокрозуб, В. Г. Представление продукционных знаний в реляционных базах данных / В. Г. Мокрозуб // Информатика: проблемы, методология, технологии : материалы XV международной научно-методической конференции, Воронеж, 12–13 февраля 2015 года. Vol. Том 3. – Воронеж: Воронежский государственный университет, 2015. – P. 114-117. – EDN WLMQZD.

8. Эрштейн, Л. Б. Обучение студентов нормализации баз данных на основе использования метода минимализации количества объектов / Л. Б. Эрштейн // Открытое и дистанционное образование. – 2019. – № 4(76). – С. 5-12. – DOI 10.17223/16095944/76/1. – EDN ZLVNDA.

9. Саликова, О. В. Выбор методики построения ER – диаграммы на занятиях по дисциплине "Базы данных" / О. В. Саликова // Университетский комплекс как региональный центр образования, науки и культуры : Материалы Всероссийской научно-методической конференции (с международным участием), Оренбург, 30 января – 01 2013 года. – Оренбург: Издательско-полиграфический комплекс "Университет", 2013. – С. 2820-2825. – EDN REGYOJ.

10. Руководство по языку программирования C# <https://metanit.com/sharp/tutorial/> Дата доступа: 04.05.2025.

11. Описание предметной области [https://studbooks.net/2243082/informatika/opisanie\_predmetnoy\_oblasti\_spravochnaya\_sistema\_kinoteatrov\_goroda\_simferopol /](https://studbooks.net/2243082/informatika/opisanie_predmetnoy_oblasti_spravochnaya_sistema_kinoteatrov_goroda_simferopol%20/) Дата доступа: 04.05.2025.

Приложение 1. Техническое задание

МИНОБРАНАУКИ РОССИИ

федеральное государственное бюджетное

образовательное учреждение высшего образования

ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Институт информационных технологий

наименование института (факультета)

Математическое и программное обеспечение ЭВМ

наименование кафедры

Проектирование баз данных

наименование дисциплины в соответствии с учебным планом

УТВЕРЖДАЮ

Зав. кафедрой МПО ЭВМ

д. т.н. \_\_\_\_\_\_\_\_\_ Ершов Е.В.

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_2025 г.

Проектирование базы данных «Кинотеатра»

Техническое задание на курсовую работу

Листов 5

Руководитель: доцент Селяничев О. Л.

Исполнитель: студент гр. 1ПИб-02-2оп-22

Зернов В. А.

2025 г.

Введение

Курсовая работа направлена на проектирование базы данных «Кинотеатра» и приложения для взаимодействия с ней.

1. Основания для разработки

Основанием для разработки является задание на курсовую работу по дисциплине «Проектирование баз данных», выданное на кафедре МПО ЭВМ.

Дата утверждения: 10 февраля 2025 года.

Наименование темы разработки: Проектирование базы данных «Кинотеатра».

2. Назначение разработки

Основной задачей курсовой работы является проектирование базы данных для хранения информации из кинотеатра и взаимодействие посредством программы.

3. Требования к программе

* 1. Требования к функциональным характеристикам

В курсовой работе необходимо реализовать:

* управление фильмами, жанрами, сеансами, залами, меню, акциями;
* бронирование билетов на сеанс с указанием итоговый цены и примененных скидов;
* в соответствии с темой создать базу данных из 10-15 реляционных таблиц;
* написать программу, реализующую 25-30 запросов;
* интерфейс выдержать в виде, привычном для любого пользователя – в духе Microsoft Office;
* структуру таблиц определить вручную – внешними относительно своей программы средствами;
  1. Требования к надежности

В курсовой работе для надежной работы ПО должно быть реализовано:

* корректное отображание вводимых данных пользователем;
* производить изменение и удаление кортежей данных;
* правильно установить типы данных для атрибутов.
  1. Условия эксплуатации

Для корректной работы программы должны соблюдаться следующие условия: установлена ОС Windows, установлена СУБД Microsoft Access, установлена последняя версия .NET.

* 1. Требования к составу и параметрам технических средств

Для того, чтобы использовать ПО для работы необходимо иметь:

1. Процессор: x64; рекомендуется использовать четырехъядерный процессор или лучше;

2. Объем оперативной памяти: минимум 4 Гб и более;

3. Свободное место на диске: минимум 850 Мб;

4. Наличие компьютерной мыши, клавиатуры;

5. Наличие монитора.

* 1. Требования к маркировке и упаковке

Для маркировки и упаковки использовать папку курсовой работы и бумажный конверт с диском внутри.

* 1. Требования к транспортированию и хранению

Транспортировать программу и базу данных разрешается любым лицам.

* 1. Специальные требования

Специальные требования: не раскрывать личные данные создателя базы данных и приложения.

4. Требования к программной документации

* 1. Содержание расчетно-пояснительной записки

Расчетно-появснительная записка должна содержать всю необходимую информацию о работе курсовой работы и теории, а именно:

* 1. Титульный лист.
  2. Оглавление.
  3. Введение (краткая характеристика проблемы).
  4. Основная часть

- дать описание предметной области;

- провести рассуждения на предмет инфологического проектирования, выполнить даталогическое проектирование - обосновать выбор предлагаемой вами модели данных, осуществить процесс нормализации, отразить логическую и физическую модели, построить ER-диаграмму;

- описать, как работает программа (с позиции разработчика!), каковы ее возможности, приемы навигации по ней;

* 1. Заключение
  2. Список литературы.
  3. Приложения (содержит ТЗ, некоторые диаграммы, таблицы, иллюстрирующие материал Основной части, а также распечатки текстов файлов-исходников с комментариями; Руководство пользователя; из основного текста должны быть ссылки на номера разделов Приложения)
  4. Требования к оформлению

В данном пункте представлены требования к оформлению документации в соответствии с ГОСТ (табл. П1.1).

Таблица П1.1

Требования к оформлению

|  |  |
| --- | --- |
| Документ | Печать на отдельных листах формата А4 (210х297 мм); оборотная сторона не заполняется; листы нумеруются. Печать возможна ч/б. Файлы предъявляются на компакт-диске: РПЗ с ТЗ; программный код. Листы и диск в конверте вложены в пластиковую папку скоросшивателя. |
| Страницы | Ориентация – книжная; отдельные страницы, при необходимости, альбомная. Поля: верхнее, нижнее – по 2 см, левое – 3 см , правое – 1 см. |
| Абзацы | Межстрочный интервал – 1, перед и после абзаца – 0. |
| Шрифты | Кегль – 14. В таблицах шрифт 12. Шрифт листинга – 10 (возможно в 2 колонки). |
| Рисунки | Подписывается под ним по центру: Рис.Х. Название. В приложениях: Рис.П1.3. Название |
| Таблицы | Подписывается: над таблицей, выравнивание по правому: «Таблица Х». В следующей строке по центру Название Надписи в «шапке» (имена столбцов, полей) – по центру. В теле таблицы (записи) текстовые значения – выровнены по левому краю, числа, даты – по правому. |

1. Стадии и этапы разработки

В данном пункте описаны стадии и этапы разработки программы (табл. П1.2).

Таблица П1.2

Стадии и этапы разработки

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование этапа разработки | Сроки разработки | Результат выполнения | Отметка о выполнении |
| Определение темы для курсовой работы | 10.02.2024 | Утверждена тема для курсовой работы |  |
| Оформление технического задания | 07.03.2025 | Оформленное техническое задание |  |
| Изучение и описание предметной области | 07.03.2025 | Получены теоретические знания для проектирования базы данных по представленной теме |  |
| Концептуальное проектирование | 20.04.2025 | ER диаграмма |  |
| Инфологическое проектирование | 20.04.2025 | Структурирование информации |  |

Продолжение табл П1. 2

|  |  |  |  |
| --- | --- | --- | --- |
| Даталогическое проектирование | 20.04.2025 | Схема базы данных на основе конкретной модели |  |
| Нормализация | 25.04.2025 | БД в виде 3НФ |  |
| Программирование | 25.04.2025 | Готовый вариант программы |  |
| Оформление сопроводительной документации | 30.04.2025 | Оформленная сопроводительная документация |  |

6. Порядок контроля и приемки

В данном пункте описан порядок контроля и приемки курсовой работы (табл. П1.3).

Таблица П1.3

Порядок контроля и приемки

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование контрольного этапа выполнения курсовой работы | Сроки контроля | Результат выполнения | Отметка о приемке результата контрольного этапа |
| Сдача технического задания | 07.10.2024 | Согласованное техническое задания |  |
| Сдача расчетно-пояснительной записки | 22.05.2025 | Согласованная расчетно-пояснительная записка |  |
| Сдача курсовой работы | 2.05.2025 | Получение оценки за выполненную работу |  |

# Приложение 2. Руководство пользователя

1. Общие сведения о программе

Данная программа представляет собой приложение для работы с данными Кинотеатра. Это приложение работает с реляционной нормализованной базой данных.

Программное обеспечение выполняет следующие функции:

* вывод данных на экран монитора;
* реализация запросов;
* добавление новых записей;
* удаление записей.

2. Описание установки

Вставить диск (или флеш-накопитель) в оптический привод компьютера (или в USB разъем). Открыть папку с содержимым диска. В папке диска найти файлы Cinema.exe и database.mdb. И перенести их на свой ПК.

3. Описание запуска

Программа «Кинотеатр» запускается с помощью файла Cinema.exe, при запуске появится интерфейс главного окна программы рис. П2.1.

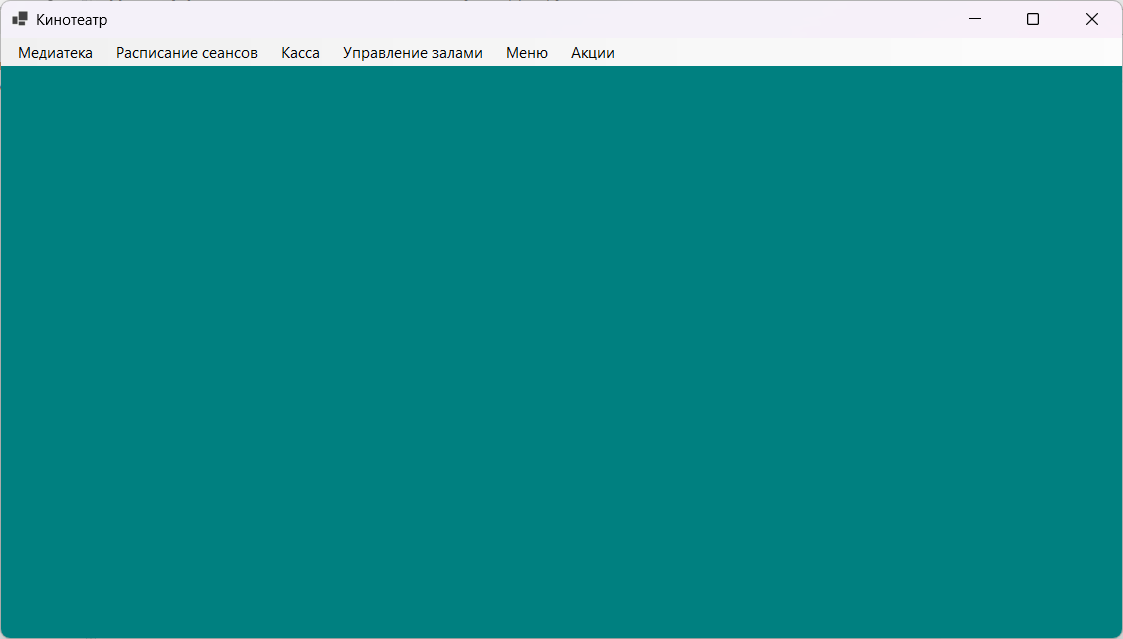


Рис. П2.1. Интерфейс главного окна

4. Инструкции по работе

Программа позволяет работать с записями базы данных. На главном меню есть вкладки с именами, которые присутствует в базе данных рис. П2.3. При выборе одной из них, в главном окне появляются данные, соответствующие этой таблице рис. П2.2 – 2.3.

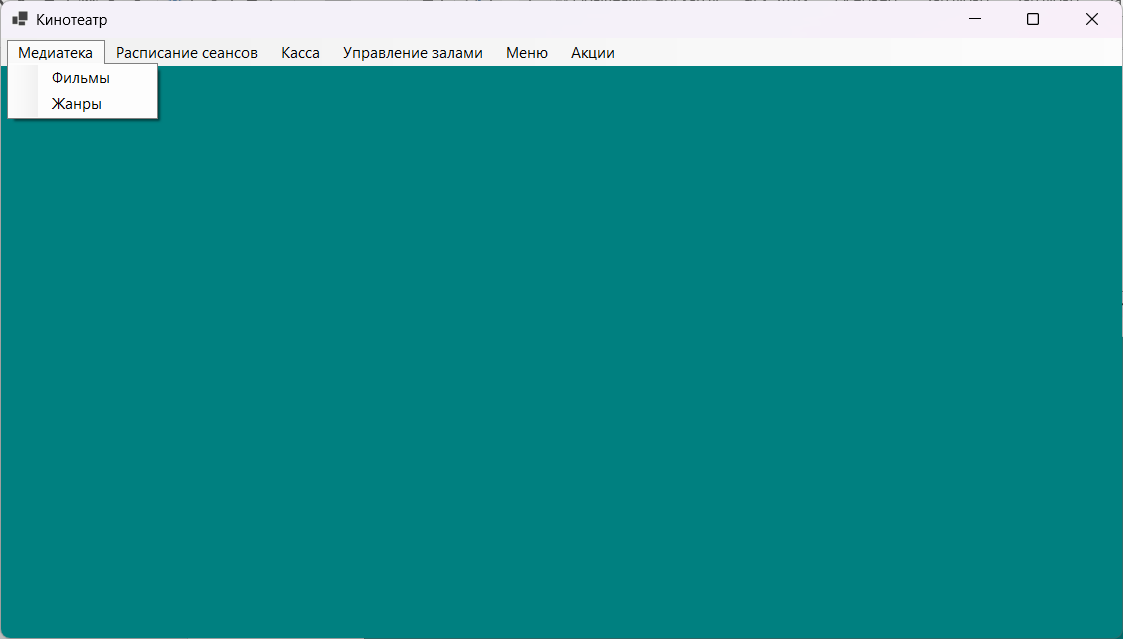


Рис. П2.2. Просмотр таблиц

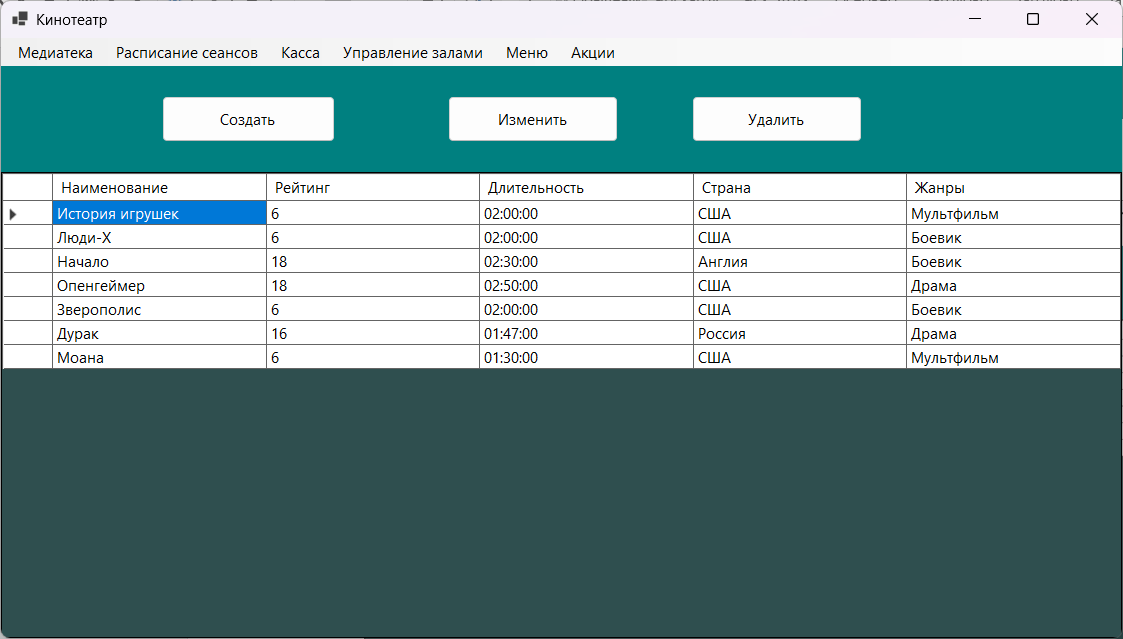


Рис. П2.3. Отображение таблицы «Фильмы»

Редактирование данных происходит через отдельное окно при нажатии кнопки «Изменить». Пользователь должен выбрать строку таблицы, которые хочет изменить и нажать на кнопку «Изменить», затем откроется окно для ввода данных и там ввести данные, которые хочет изменить и нажать кнопку «Обновить». Данный функционал представлен на рис П2.4-П2.5. Удаление записи происходит с помощью выделения записи щелчком мыши по самому первому столбцу, после чего необходимо нажать кнопку «Удалить», после нажатия на которую происходит удаление выбранных записей. Данные меняются автоматически с открытием той или иной вкладки.

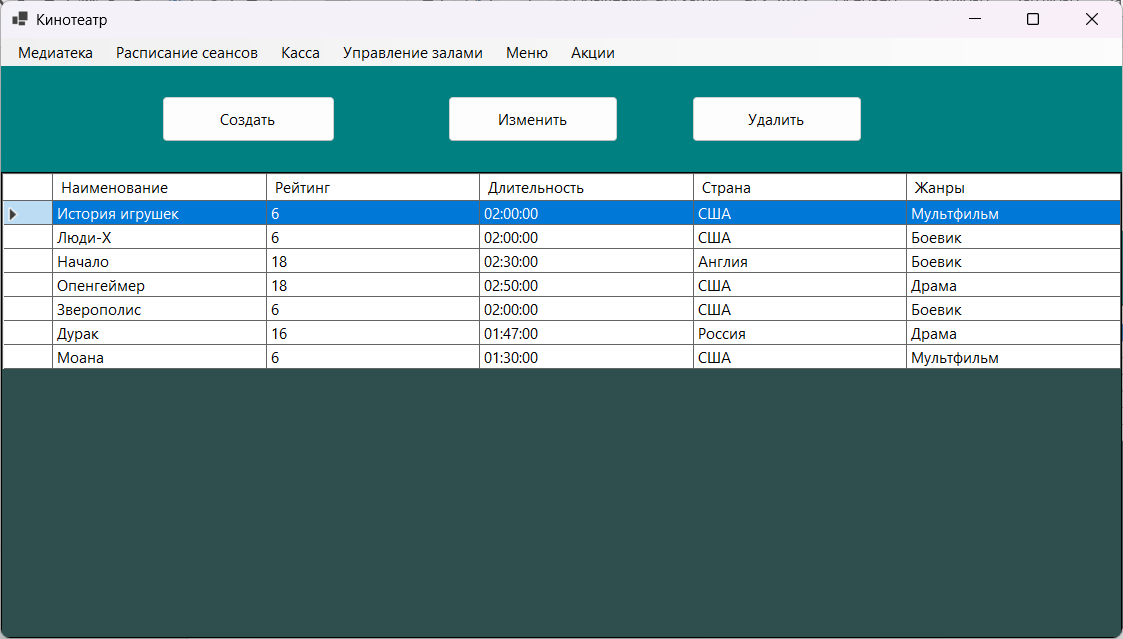


Рис. П2.4. Выбрана строка для изменения

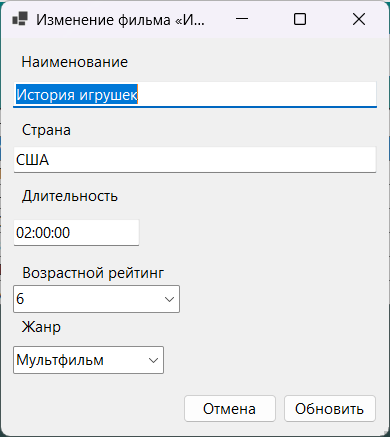


Рис П2.5. Окно для изменения записи

Дополнительно были созданы всплывающие окна, оповещающие об ошибках проведенной операции. Представлено рис. П2.6-10

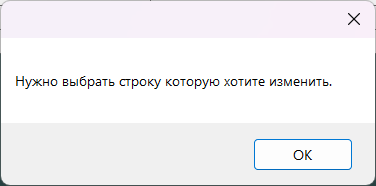


Рис. П2.8. Нажатие кнопки «Обновить» без выбранной строки

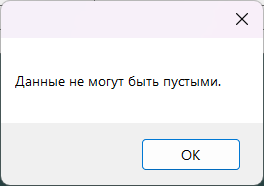


Рис. П2.9. Нажатие кнопки «Создать» или «Обновить» при незаполненных данных

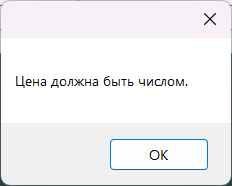


Рис. П2.10. Нажатие кнопки «Создать» или «Обновить» с полем, которое числовое и ввести не число

Выход из программы представлен на рис. П2.11.

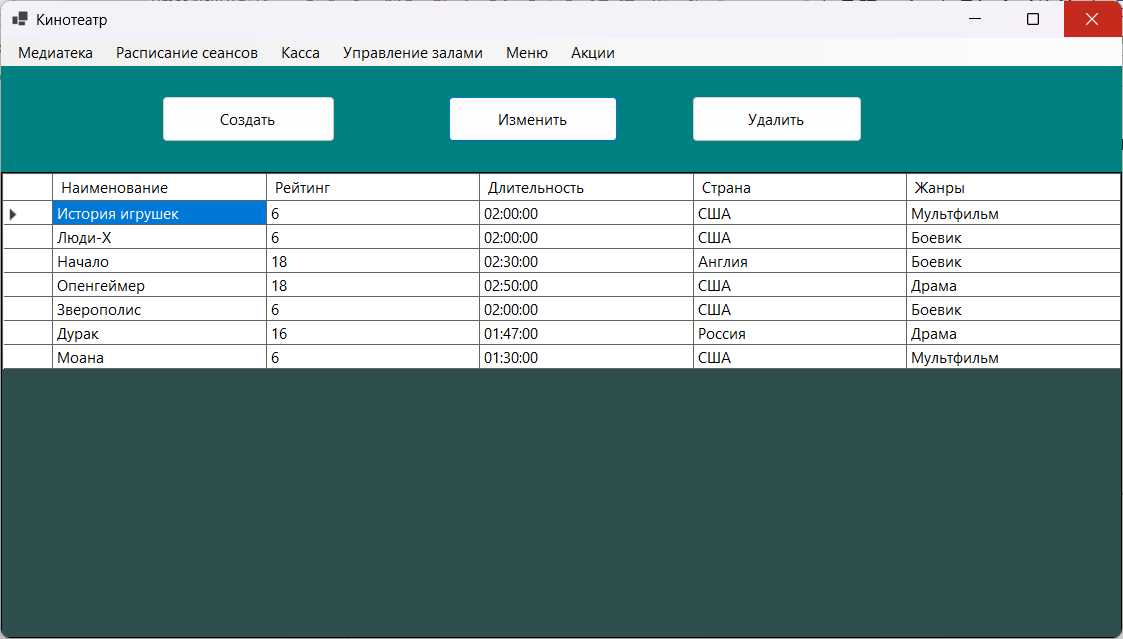


Рис. П2.11. Выход из программы

Приложение 3. Текст программы

Файл "Result.cs":

namespace Core;

public class Result

{

private Result(bool isSuccess)

{

IsSuccess = isSuccess;

}

public bool IsSuccess

{

get;

init;

}

public static Result Success() => new Result(true);

public static Result Failure() => new Result(false);

}

Файл "Genre.cs":

﻿namespace Core.Genres;

public record Genre(string DisplayName);

Файл "IGenreRepository.cs":

﻿namespace Core.Genres.Repositories;

public interface IGenreRepository

{

Result Add(Genre genre);

IEnumerable<Genre> GetGenres();

Result Update(string oldName, Genre genre);

Result Delete(string genreName);

}

Файл "Hall.cs":

﻿namespace Core.Halls;

public record Hall(string DisplayName, int Capacity);

Файл "Seat.cs":

﻿namespace Core.Halls;

public record Seat(int Row, int SeatNumber);

Файл "IHallRepository.cs":

﻿namespace Core.Halls.Repositories;

public interface IHallRepository

{

Result Add(Hall hall);

IEnumerable<Hall> GetHalls();

Result Delete(string hallName);

IEnumerable<Seat> GetAllSeatsByHallName(string hallName);

}

Файл "MenuItem.cs":

﻿namespace Core.MenuItems;

public record MenuItem(string DisplayName, int Price, bool IsAvailable = true);

Файл "IMenuItemRepository.cs":

﻿namespace Core.MenuItems.Repositories;

public interface IMenuItemRepository

{

Result Add(MenuItem menuItem);

IEnumerable<MenuItem> GetMenuItems();

Result Update(string oldName, MenuItem menuItem);

Result Delete(string name);

}

Файл "Movie.cs":

﻿namespace Core.Movies;

public class Movie

{

public Movie(string displayName, int rating, TimeSpan duration, string? country, string genres)

{

DisplayName = displayName;

Rating = rating;

Duration = duration;

Country = country;

Genres = genres;

}

public Movie()

{

}

public string DisplayName

{

get;

init;

}

public int Rating

{

get;

init;

}

public TimeSpan Duration

{

get;

init;

}

public string? Country

{

get;

init;

}

public string Genres

{

get;

init;

}

}

Файл "IMovieRepository.cs":

﻿namespace Core.Movies.Repositories;

public interface IMovieRepository

{

Result Add(Movie movie);

Result Update(Movie oldData, Movie movie);

IEnumerable<Movie> GetMovies();

Result Delete(Movie movie);

}

Файл "Promotion.cs":

﻿namespace Core.Promotions;

public record Promotion(string DisplayName, int DiscountPercent, DateTime StartDateTime, DateTime EndDateTime);

Файл "IPromotionRepository.cs":

﻿namespace Core.Promotions.Repositories;

public interface IPromotionRepository

{

Result Add(Promotion promotion);

IEnumerable<Promotion> GetPromotions();

Result Delete(string name);

}

Файл "Purchase.cs":

﻿namespace Core.Purchases;

public record Purchase(int TicketId, string? MenuItems, string? PromotionName);

Файл "IPurchaseRepository.cs":

﻿namespace Core.Purchases.Repositories;

public interface IPurchaseRepository

{

int? Add(Purchase purchase);

Result Delete();

}

Файл "Screening.cs":

﻿namespace Core.Screenings;

public record Screening(int Id, string MovieName, string HallName, int Price, DateTime StartDateTime, DateTime EndDateTime);

Файл "IScreeningRepository.cs":

﻿namespace Core.Screenings.Repositories;

public interface IScreeningRepository

{

Result Add(Screening screening);

IEnumerable<Screening> GetScreenings();

Result Update(Screening screening);

Result Delete(Screening screening);

IEnumerable<(int Row, int SeatNumber, int TicketId)> GetSeatTickets(int scrId);

}

Файл "GenreRepository.cs":

﻿using System.Data.OleDb;

using Core;

using Core.Genres;

using Core.Genres.Repositories;

using Core.Movies;

using Dapper;

using Infrastructure.Options;

namespace Infrastructure.Genres;

public class GenreRepository : IGenreRepository

{

private readonly string \_connectionString;

public GenreRepository(ConnectionStrings connectionStrings)

{

ArgumentNullException.ThrowIfNull(connectionStrings);

\_connectionString = connectionStrings.Database;

}

public Result Update(string oldName, Genre genre)

{

const string queryGenre = @"SELECT Id FROM Genres WHERE Genres.DisplayName = @DisplayName";

using var connection = new OleDbConnection(\_connectionString);

connection.Open();

using var transaction = connection.BeginTransaction();

try

{

var existingId = connection.QueryFirstOrDefault<int?>(queryGenre,

new

{

DisplayName = oldName

},

transaction: transaction);

if (!existingId.HasValue)

{

transaction.Commit();

return Result.Failure();

}

var check = connection.QueryFirstOrDefault<int?>(queryGenre, new

{

DisplayName = genre.DisplayName,

});

if(check.HasValue)

{

transaction.Commit();

return Result.Failure();

}

var updateQuery = $@"

UPDATE Genres

SET Genres.DisplayName = '{genre.DisplayName}'

WHERE Genres.Id = {existingId}";

connection.Execute(updateQuery,

transaction: transaction);

transaction.Commit();

return Result.Success();

}

catch

{

transaction.Rollback();

throw;

}

finally

{

connection.Close();

}

}

public Result Delete(string genreName)

{

const string queryGenre = @"SELECT Id FROM Genres WHERE Genres.DisplayName = @DisplayName";

const string countMovieGenre = @"SELECT COUNT(\*) FROM MovieGenres WHERE GenreId = @id;";

using var connection = new OleDbConnection(\_connectionString);

connection.Open();

using var transaction = connection.BeginTransaction();

try

{

var existingId = connection.QueryFirstOrDefault<int?>(queryGenre,

new

{

DisplayName = genreName

},

transaction: transaction);

if (!existingId.HasValue)

{

transaction.Commit();

return Result.Failure();

}

var countMovieWithGenre = connection.QueryFirstOrDefault<int?>(countMovieGenre,

new

{

id = existingId.Value

},

transaction: transaction);

if (countMovieWithGenre.HasValue && countMovieWithGenre.Value > 0)

{

transaction.Commit();

return Result.Failure();

}

var deleteQuery = $@"

delete from Genres where Id={existingId}";

connection.Execute(deleteQuery,

transaction: transaction);

transaction.Commit();

return Result.Success();

}

catch

{

transaction.Rollback();

throw;

}

finally

{

connection.Close();

}

}

public Result Add(Genre genre)

{

const string insertQuery = $"insert into Genres({nameof(Genre.DisplayName)}) values(@DisplayName);";

const string checkQuery = @"

SELECT Id FROM Genres

WHERE DisplayName = @DisplayName;";

using var connection = new OleDbConnection(\_connectionString);

connection.Open();

using var transaction = connection.BeginTransaction();

try

{

var existingId = connection.QueryFirstOrDefault<int?>(checkQuery,new

{

genre.DisplayName

},

transaction: transaction);

if (existingId.HasValue)

{

transaction.Commit();

return Result.Failure();

}

connection.Execute(insertQuery,

transaction: transaction,

param: new

{

genre.DisplayName

});

transaction.Commit();

return Result.Success();

}

catch

{

transaction.Rollback();

return Result.Failure();

}

finally

{

connection.Close();

}

}

public IEnumerable<Genre> GetGenres()

{

using var connection = new OleDbConnection(\_connectionString);

connection.Open();

var result = connection.Query<Genre>($"SELECT DisplayName as {nameof(Genre.DisplayName)} FROM Genres");

connection.Close();

return result;

}

}

Файл "HallRepository.cs":

﻿using System.Data.OleDb;

using Core;

using Core.Halls;

using Core.Halls.Repositories;

using Dapper;

using Infrastructure.Options;

namespace Infrastructure.Halls;

public class HallRepository : IHallRepository

{

private readonly string \_connectionString;

public HallRepository(ConnectionStrings connectionStrings)

{

ArgumentNullException.ThrowIfNull(connectionStrings);

\_connectionString = connectionStrings.Database;

}

public Result Add(Hall hall)

{

const string checkQuery = @"

SELECT Id FROM Halls

WHERE DisplayName = @DisplayName";

const string insertQuery = @"

INSERT INTO Halls (DisplayName, Capacity)

VALUES (@DisplayName, @Capacity)";

const string insertSeatsByHall = @"insert into Seats(HallId, Row, SeatNumber) values(@HallId, @Row, @SeatNumber)";

const string getIdQuery = "SELECT @@IDENTITY";

using var connection = new OleDbConnection(\_connectionString);

connection.Open();

using var transaction = connection.BeginTransaction();

try

{

var existingId = connection.QueryFirstOrDefault<int?>(checkQuery,

new

{

hall.DisplayName,

},

transaction: transaction);

if (existingId.HasValue)

{

transaction.Commit();

return Result.Failure();

}

connection.Execute(insertQuery,

new

{

hall.DisplayName,

hall.Capacity

},

transaction: transaction);

var newId = connection.QueryFirst<int>(getIdQuery, transaction: transaction);

var fullRows = hall.Capacity / 10;

var remainingSeats = hall.Capacity % 10;

var totalRows = fullRows + (remainingSeats > 0 ? 1 : 0);

for (var row = 1; row <= fullRows; row++)

{

for (var seatNumber = 1; seatNumber <= 10; seatNumber++)

{

connection.Execute(insertSeatsByHall,

new

{

HallId = newId,

Row = row,

SeatNumber = seatNumber

},

transaction: transaction);

}

}

if (remainingSeats > 0)

{

for (var i = 1; i <= remainingSeats; i++)

{

connection.Execute(insertSeatsByHall,

new

{

HallId = newId,

Row = totalRows,

SeatNumber = i

},

transaction: transaction);

}

}

transaction.Commit();

return Result.Success();

}

catch

{

transaction.Rollback();

throw;

}

finally

{

connection.Close();

}

}

public IEnumerable<Hall> GetHalls()

{

using var connection = new OleDbConnection(\_connectionString);

connection.Open();

var result = connection.Query<Hall>($"SELECT Halls.DisplayName, Halls.Capacity FROM Halls");

connection.Close();

return result;

}

public Result Delete(string hallName)

{

const string findIdQuery = @"

SELECT Id FROM Halls

WHERE DisplayName = @DisplayName;";

const string deleteSeats = @"DELETE FROM Seats WHERE Seats.HallId = @HallId";

using var connection = new OleDbConnection(\_connectionString);

connection.Open();

using var transaction = connection.BeginTransaction();

try

{

var existingId = connection.QueryFirstOrDefault<int?>(findIdQuery,

new

{

DisplayName = hallName

},

transaction: transaction);

if (!existingId.HasValue)

{

transaction.Commit();

return Result.Failure();

}

connection.Execute(deleteSeats,

new

{

HallId = existingId.Value

},

transaction: transaction);

var deleteQuery = $@"

delete from Halls where Halls.Id = {existingId};";

connection.Execute(deleteQuery, transaction: transaction);

transaction.Commit();

return Result.Success();

}

catch

{

transaction.Rollback();

throw;

}

finally

{

connection.Close();

}

}

public IEnumerable<Seat> GetAllSeatsByHallName(string hallName)

{

const string findIdQuery = @"

SELECT Id FROM Halls

WHERE DisplayName = @DisplayName;";

using var connection = new OleDbConnection(\_connectionString);

connection.Open();

using var transaction = connection.BeginTransaction();

try

{

var existingId = connection.QueryFirstOrDefault<int?>(findIdQuery,

new

{

DisplayName = hallName

},

transaction: transaction);

if (!existingId.HasValue)

{

transaction.Commit();

return [];

}

var result = connection.Query<Seat>($"SELECT Seats.Row, Seats.SeatNumber FROM Seats WHERE Seats.HallId = {existingId}", transaction: transaction);

transaction.Commit();

return result;

}

catch

{

transaction.Rollback();

throw;

}

finally

{

connection.Close();

}

}

}

Файл "MenuItemRepository.cs":

﻿using System.Data.OleDb;

using Core;

using Core.MenuItems;

using Core.MenuItems.Repositories;

using Dapper;

using Infrastructure.Options;

namespace Infrastructure.MenuItems;

public class MenuItemRepository : IMenuItemRepository

{

private readonly string \_connectionString;

public MenuItemRepository(ConnectionStrings connectionStrings)

{

ArgumentNullException.ThrowIfNull(connectionStrings);

\_connectionString = connectionStrings.Database;

}

public Result Add(MenuItem menuItem)

{

const string insertQuery = $"insert into MenuItems(DisplayName, Price, IsAvailable) values(@DisplayName, @Price, True);";

const string checkQuery = @"

SELECT Id FROM MenuItems

WHERE DisplayName = @DisplayName;";

using var connection = new OleDbConnection(\_connectionString);

connection.Open();

using var transaction = connection.BeginTransaction();

try

{

var existingId = connection.QueryFirstOrDefault<int?>(checkQuery,new

{

menuItem.DisplayName

},

transaction: transaction);

if (existingId.HasValue)

{

transaction.Commit();

return Result.Failure();

}

connection.Execute(insertQuery,

transaction: transaction,

param: new

{

DisplayName = menuItem.DisplayName,

Price = menuItem.Price

});

transaction.Commit();

return Result.Success();

}

catch

{

transaction.Rollback();

return Result.Failure();

}

finally

{

connection.Close();

}

}

public IEnumerable<MenuItem> GetMenuItems()

{

using var connection = new OleDbConnection(\_connectionString);

connection.Open();

var result = connection.Query<MenuItem>($"SELECT MenuItems.DisplayName, MenuItems.Price, MenuItems.IsAvailable FROM MenuItems");

connection.Close();

return result;

}

public Result Update(string oldName, MenuItem menuItem)

{

const string queryGenre = @"

SELECT Id FROM MenuItems

WHERE DisplayName = @DisplayName;";

using var connection = new OleDbConnection(\_connectionString);

connection.Open();

using var transaction = connection.BeginTransaction();

try

{

var existingId = connection.QueryFirstOrDefault<int?>(queryGenre,

new

{

DisplayName = oldName

},

transaction: transaction);

if (!existingId.HasValue)

{

transaction.Commit();

return Result.Failure();

}

var check = connection.QueryFirstOrDefault<int?>(queryGenre, new

{

DisplayName = menuItem.DisplayName,

}, transaction: transaction);

if(check.HasValue && check.Value != existingId)

{

transaction.Commit();

return Result.Failure();

}

var isAvailable = menuItem.IsAvailable ? "True" : "False";

var updateQuery = $@"

UPDATE MenuItems

SET MenuItems.DisplayName = '{menuItem.DisplayName}',

MenuItems.Price = {menuItem.Price},

MenuItems.IsAvailable = {isAvailable}

WHERE MenuItems.Id = {existingId}";

connection.Execute(updateQuery,

transaction: transaction);

transaction.Commit();

return Result.Success();

}

catch

{

transaction.Rollback();

throw;

}

finally

{

connection.Close();

}

}

public Result Delete(string name)

{

const string findIdQuery = @"

SELECT Id FROM MenuItems

WHERE DisplayName = @DisplayName;";

using var connection = new OleDbConnection(\_connectionString);

connection.Open();

using var transaction = connection.BeginTransaction();

try

{

var existingId = connection.QueryFirstOrDefault<int?>(findIdQuery,

new

{

DisplayName = name

},

transaction: transaction);

if (!existingId.HasValue)

{

transaction.Commit();

return Result.Failure();

}

var deleteQuery = $@"

delete from MenuItems where MenuItems.Id = {existingId};";

connection.Execute(deleteQuery, transaction: transaction);

transaction.Commit();

return Result.Success();

}

catch

{

transaction.Rollback();

throw;

}

finally

{

connection.Close();

}

}

}

Файл "MovieRepository.cs":

﻿using System.Data.OleDb;

using Core;

using Core.Genres;

using Core.Movies;

using Core.Movies.Repositories;

using Dapper;

using Infrastructure.Options;

namespace Infrastructure.Movies;

public class MovieRepository : IMovieRepository

{

private readonly string \_connectionString;

public MovieRepository(ConnectionStrings connectionStrings)

{

ArgumentNullException.ThrowIfNull(connectionStrings);

\_connectionString = connectionStrings.Database;

}

public IEnumerable<Movie> GetMovies()

{

using var connection = new OleDbConnection(\_connectionString);

connection.Open();

const string getMovieQuery = $@"

SELECT Movies.Id as {nameof(MovieDto.Id)},

Movies.DisplayName as {nameof(MovieDto.DisplayName)},

Movies.Rating as {nameof(MovieDto.Rating)},

Movies.Duration as {nameof(MovieDto.Duration)},

Movies.Country as {nameof(MovieDto.Country)}

FROM Movies;

";

const string getMovieGenresQuery = @$"

SELECT

MovieGenres.MovieId AS {nameof(MovieGenreDto.MovieId)},

g.Id AS {nameof(MovieGenreDto.GenreId)},

g.DisplayName as {nameof(MovieGenreDto.DisplayName)}

FROM MovieGenres

INNER JOIN Genres g ON MovieGenres.GenreId = g.Id;

";

var movieDtos = connection.Query<MovieDto>(getMovieQuery);

var movieGenres = connection.Query<MovieGenreDto>(getMovieGenresQuery);

connection.Close();

var genresByMovie = movieGenres.GroupBy(g => g.MovieId)

.ToDictionary(g => g.Key, g => string.Join(", ", g.Select(x => x.DisplayName)));

return movieDtos.Select(m => new Movie(m.DisplayName,

m.Rating,

TimeSpan.Parse(m.Duration),

m.Country,

genresByMovie.TryGetValue(m.Id, out var genres) ? genres : string.Empty));

}

public Result Delete(Movie movie)

{

const string findIdQuery = @"

SELECT Id FROM Movies

WHERE DisplayName = @DisplayName

AND Duration = @Duration

AND Country = @Country

AND Rating = @Rating";

const string deleteMovieGenresQuery = @"DELETE FROM MovieGenres WHERE MovieGenres.MovieId = @MovieId";

using var connection = new OleDbConnection(\_connectionString);

connection.Open();

using var transaction = connection.BeginTransaction();

try

{

var existingId = connection.QueryFirstOrDefault<int?>(findIdQuery,

new

{

movie.DisplayName,

Duration = movie.Duration.ToString(),

movie.Country,

movie.Rating

},

transaction: transaction);

if (!existingId.HasValue)

{

transaction.Commit();

return Result.Failure();

}

if (movie.Genres.Length > 1)

{

connection.Execute(deleteMovieGenresQuery,

new { MovieId = existingId.Value },

transaction: transaction);

}

var deleteQuery = $@"

delete from Movies where Movies.Id = {existingId};";

connection.Execute(deleteQuery,

transaction: transaction);

transaction.Commit();

return Result.Success();

}

catch

{

transaction.Rollback();

throw;

}

finally

{

connection.Close();

}

}

public Result Update(Movie oldData, Movie movie)

{

const string findIdQuery = @"

SELECT Id FROM Movies

WHERE DisplayName = @DisplayName

AND Duration = @Duration

AND Country = @Country

AND Rating = @Rating";

const string deleteMovieGenresQuery = @"DELETE FROM MovieGenres WHERE MovieGenres.MovieId = @MovieId";

const string insertMovieGenresQuery = @"INSERT INTO MovieGenres(MovieGenres.MovieId, MovieGenres.GenreId) VALUES(@MovieId, @GenreId)";

const string queryGenre = @"SELECT Id FROM Genres WHERE Genres.DisplayName = @DisplayName";

using var connection = new OleDbConnection(\_connectionString);

connection.Open();

using var transaction = connection.BeginTransaction();

try

{

var existingId = connection.QueryFirstOrDefault<int?>(findIdQuery,

new

{

oldData.DisplayName,

Duration = oldData.Duration.ToString(),

oldData.Country,

oldData.Rating

},

transaction: transaction);

if (!existingId.HasValue)

{

transaction.Commit();

return Result.Failure();

}

var check = connection.QueryFirstOrDefault<int?>(findIdQuery,

new

{

movie.DisplayName,

Duration = movie.Duration.ToString(),

movie.Country,

movie.Rating

},

transaction: transaction);

if(check.HasValue && check.Value != existingId.Value)

{

transaction.Commit();

return Result.Failure();

}

var updateQuery = $@"

UPDATE Movies

SET Movies.DisplayName = '{movie.DisplayName}',

Movies.Duration = '{movie.Duration}',

Movies.Country = '{movie.Country}',

Movies.Rating = {movie.Rating}

WHERE Movies.Id = {existingId}";

connection.Execute(updateQuery,

transaction: transaction);

if(oldData.Genres.Length > 1)

{

connection.Execute(deleteMovieGenresQuery,

new { MovieId = existingId.Value },

transaction: transaction);

}

if (movie.Genres.Length > 0)

{

var genreId = connection.QueryFirst<int>(queryGenre,

new { DisplayName = movie.Genres },

transaction: transaction);

connection.Execute(insertMovieGenresQuery,

new

{

MovieId = existingId.Value,

GenreId = genreId

},

transaction: transaction);

}

transaction.Commit();

return Result.Success();

}

catch

{

transaction.Rollback();

throw;

}

finally

{

connection.Close();

}

}

public Result Add(Movie movie)

{

const string checkQuery = @"

SELECT Id FROM Movies

WHERE DisplayName = @DisplayName

AND Duration = @Duration

AND Country = @Country

AND Rating = @Rating";

const string insertQuery = @"

INSERT INTO Movies (DisplayName, Duration, Country, Rating)

VALUES (@DisplayName, @Duration, @Country, @Rating)";

const string insertMovieGenresQuery = @"insert into MovieGenres(MovieId, GenreId) values(@MovieId, @GenreId)";

const string queryGenre = @"SELECT Genres.Id FROM Genres WHERE Genres.DisplayName = @DisplayName";

const string getIdQuery = "SELECT @@IDENTITY";

using var connection = new OleDbConnection(\_connectionString);

connection.Open();

using var transaction = connection.BeginTransaction();

try

{

var existingId = connection.QueryFirstOrDefault<int?>(checkQuery,

new

{

movie.DisplayName,

Duration = movie.Duration.ToString(),

movie.Country,

movie.Rating

},

transaction: transaction);

if (existingId.HasValue)

{

transaction.Commit();

return Result.Failure();

}

connection.Execute(insertQuery,

new

{

movie.DisplayName,

Duration = movie.Duration.ToString(),

movie.Country,

movie.Rating

},

transaction: transaction);

var newId = connection.QueryFirst<int>(getIdQuery, transaction: transaction);

if(movie.Genres.Length > 0)

{

var genreId = connection.QueryFirst<int>(queryGenre,

transaction: transaction,

param: new

{

DisplayName = movie.Genres

});

connection.Execute(insertMovieGenresQuery,

new

{

MovieId = newId,

GenreId = genreId

},

transaction);

}

transaction.Commit();

return Result.Success();

}

catch

{

transaction.Rollback();

throw;

}

finally

{

connection.Close();

}

}

private class MovieDto

{

public int Id

{

get;

set;

}

public string DisplayName

{

get;

set;

}

public int Rating

{

get;

set;

}

public string Duration

{

get;

set;

}

public string? Country

{

get;

set;

}

}

private class MovieGenreDto

{

public int MovieId

{

get;

set;

}

public int GenreId

{

get;

set;

}

public string DisplayName

{

get;

set;

}

}

}

Файл "ConnectionStrings.cs":

﻿namespace Infrastructure.Options;

public record ConnectionStrings(string Database);

Файл "PromotionRepository.cs":

﻿using System.Data.OleDb;

using Core;

using Core.Promotions;

using Core.Promotions.Repositories;

using Dapper;

using Infrastructure.Options;

namespace Infrastructure.Promotions;

public class PromotionRepository : IPromotionRepository

{

private readonly string \_connectionString;

public PromotionRepository(ConnectionStrings connectionStrings)

{

ArgumentNullException.ThrowIfNull(connectionStrings);

\_connectionString = connectionStrings.Database;

}

public Result Add(Promotion promotion)

{

const string checkQuery = @"

SELECT Id FROM Promotions

WHERE DisplayName = @DisplayName;";

using var connection = new OleDbConnection(\_connectionString);

connection.Open();

using var transaction = connection.BeginTransaction();

try

{

var existingId = connection.QueryFirstOrDefault<int?>(checkQuery,

new

{

promotion.DisplayName

},

transaction: transaction);

if (existingId.HasValue)

{

transaction.Commit();

return Result.Failure();

}

var insertQuery =

$"insert into Promotions(DisplayName, DiscountPercent, StartDateTime, EndDateTime) values('{promotion.DisplayName}', {promotion.DiscountPercent}, '{promotion.StartDateTime.ToShortDateString()}', '{promotion.EndDateTime.Date.ToShortDateString()}');";

connection.Execute(insertQuery,

transaction: transaction);

transaction.Commit();

return Result.Success();

}

catch

{

transaction.Rollback();

return Result.Failure();

}

finally

{

connection.Close();

}

}

public IEnumerable<Promotion> GetPromotions()

{

using var connection = new OleDbConnection(\_connectionString);

connection.Open();

var result = connection.Query<Promotion>($@"

SELECT

DisplayName as {nameof(Promotion.DisplayName)},

DiscountPercent as {nameof(Promotion.DiscountPercent)},

StartDateTime as {nameof(Promotion.StartDateTime)},

EndDateTime as {nameof(Promotion.EndDateTime)}

FROM Promotions");

connection.Close();

return result;

}

public Result Delete(string name)

{

const string checkQuery = @"

SELECT Id FROM Promotions

WHERE DisplayName = @DisplayName;";

using var connection = new OleDbConnection(\_connectionString);

connection.Open();

using var transaction = connection.BeginTransaction();

try

{

var existingId = connection.QueryFirstOrDefault<int?>(checkQuery,

new

{

DisplayName = name

},

transaction: transaction);

if (!existingId.HasValue)

{

transaction.Commit();

return Result.Failure();

}

var query = $"DELETE FROM Promotions WHERE Promotions.Id = {existingId}";

connection.Execute(query,

transaction: transaction);

transaction.Commit();

return Result.Success();

}

catch

{

transaction.Rollback();

return Result.Failure();

}

finally

{

connection.Close();

}

}

}

Файл "PurchaseRepository.cs":

﻿using System.Data.OleDb;

using Core;

using Core.Purchases;

using Core.Purchases.Repositories;

using Dapper;

using Infrastructure.Options;

namespace Infrastructure.Purchases;

public class PurchaseRepository : IPurchaseRepository

{

private readonly string \_connectionString;

public PurchaseRepository(ConnectionStrings connectionStrings)

{

ArgumentNullException.ThrowIfNull(connectionStrings);

\_connectionString = connectionStrings.Database;

}

public int? Add(Purchase purchase)

{

using var connection = new OleDbConnection(\_connectionString);

connection.Open();

using var transaction = connection.BeginTransaction();

try

{

var now = DateTime.Now;

var ticket = connection.QueryFirstOrDefault<(int Id, int Price)?>(

$"SELECT Id, Price FROM Tickets WHERE Id = {purchase.TicketId};",

transaction: transaction);

if (!ticket.HasValue)

{

transaction.Commit();

return null;

}

var totalPrice = ticket.Value.Price;

int? menuItemId = null;

if (!string.IsNullOrEmpty(purchase.MenuItems))

{

var menuItem = connection.QueryFirstOrDefault<(int Id, int Price)?>(

$"SELECT Id, Price FROM MenuItems WHERE DisplayName = '{purchase.MenuItems}' AND IsAvailable = True;",

transaction: transaction);

if (menuItem.HasValue)

{

menuItemId = menuItem.Value.Id;

totalPrice += menuItem.Value.Price;

}

}

int? promotionId = null;

int discountAmount = 0;

if (!string.IsNullOrEmpty(purchase.PromotionName))

{

var promotion = connection.QueryFirstOrDefault<(int Id, int DiscountPercent)?>(

$"SELECT Id, DiscountPercent FROM Promotions WHERE DisplayName = '{purchase.PromotionName}';",

transaction: transaction);

if (promotion.HasValue)

{

promotionId = promotion.Value.Id;

discountAmount = (totalPrice \* promotion.Value.DiscountPercent) / 100;

totalPrice -= discountAmount;

}

}

connection.Execute(

$"INSERT INTO Purchases(TicketId, MenuItemId, TotalPrice, PurchaseDateTime) VALUES({ticket.Value.Id}, {(menuItemId.HasValue ? menuItemId.Value.ToString() : "NULL")}, {totalPrice}, '{now.ToShortDateString()}');",

transaction: transaction);

var purchaseId = connection.QueryFirst<int>("SELECT @@IDENTITY;", transaction: transaction);

if (promotionId.HasValue)

{

connection.Execute(

$"INSERT INTO AppliedDiscounts(PurchaseId, PromotionId, DiscountAmount) VALUES({purchaseId}, {promotionId.Value}, {discountAmount});",

transaction: transaction);

}

connection.Execute(

$"UPDATE Tickets SET PurchaseDateTime = '{now}' WHERE Id = {ticket.Value.Id};",

transaction: transaction);

transaction.Commit();

return totalPrice;

}

catch

{

transaction.Rollback();

return null;

}

finally

{

connection.Close();

}

}

public Result Delete()

{

using var connection = new OleDbConnection(\_connectionString);

connection.Open();

using var transaction = connection.BeginTransaction();

try

{

var purchaseId = connection.QueryFirstOrDefault<int?>(

"SELECT TOP 1 Id FROM Purchases ORDER BY PurchaseDateTime DESC;",

transaction: transaction);

if (!purchaseId.HasValue)

{

transaction.Commit();

return Result.Failure();

}

connection.Execute(

$"DELETE FROM AppliedDiscounts WHERE PurchaseId = {purchaseId.Value};",

transaction: transaction);

connection.Execute(

$"UPDATE Tickets SET PurchaseDateTime = NULL WHERE Id IN (SELECT Ticked FROM Purchases WHERE Id = {purchaseId.Value});",

transaction: transaction);

connection.Execute(

$"DELETE FROM Purchases WHERE Id = {purchaseId.Value};",

transaction: transaction);

transaction.Commit();

return Result.Success();

}

catch

{

transaction.Rollback();

return Result.Failure();

}

finally

{

connection.Close();

}

}

}

Файл "ScreeningRepository.cs":

﻿using System.Data.OleDb;

using Core;

using Core.Screenings;

using Core.Screenings.Repositories;

using Dapper;

using Infrastructure.Options;

namespace Infrastructure.Screenings;

public class ScreeningRepository : IScreeningRepository

{

private readonly string \_connectionString;

public ScreeningRepository(ConnectionStrings connectionStrings)

{

ArgumentNullException.ThrowIfNull(connectionStrings);

\_connectionString = connectionStrings.Database;

}

public Result Add(Screening screening)

{

const string checkMovieQuery = "SELECT Id FROM Movies WHERE DisplayName = @MovieName;";

const string getHallQuery = "SELECT Id, Capacity FROM Halls WHERE DisplayName = @HallName;";

const string getSeatsQuery = "SELECT Id FROM Seats WHERE HallId = @HallId ORDER BY Row, SeatNumber;";

const string checkScreeningQuery = @"SELECT Id FROM Screenings

WHERE MovieId = @MovieId AND HallId = @HallId;";

const string getLastScreeningIdQuery = "SELECT @@IDENTITY;";

const string insertTicketQuery = @"INSERT INTO Tickets(ScreeningId, SeatId, Price)

VALUES(@ScreeningId, @SeatId, @Price);";

using var connection = new OleDbConnection(\_connectionString);

connection.Open();

using var transaction = connection.BeginTransaction();

try

{

var movieId = connection.QueryFirstOrDefault<int?>(checkMovieQuery,

new

{

MovieName = screening.MovieName

},

transaction: transaction);

if (!movieId.HasValue)

{

transaction.Commit();

return Result.Failure();

}

var hall = connection.QueryFirstOrDefault<(int Id, int Capacity)?>(getHallQuery,

new

{

HallName = screening.HallName

},

transaction: transaction);

if (!hall.HasValue)

{

transaction.Commit();

return Result.Failure();

}

var existingId = connection.QueryFirstOrDefault<int?>(checkScreeningQuery,

new

{

MovieId = movieId,

HallId = hall.Value.Id

},

transaction: transaction);

if (existingId.HasValue)

{

transaction.Commit();

return Result.Failure();

}

var insertScreeningQuery = $@"INSERT INTO Screenings(MovieId, HallId, StartDateTime, EndDateTime)

VALUES({movieId}, {hall.Value.Id}, '{screening.StartDateTime}', '{screening.EndDateTime}');";

connection.Execute(insertScreeningQuery,

transaction: transaction);

var screeningId = connection.QueryFirst<int>(getLastScreeningIdQuery, transaction: transaction);

var seats = connection.Query<int>(getSeatsQuery,

new

{

HallId = hall.Value.Id

},

transaction: transaction);

foreach (var seatId in seats)

{

connection.Execute(insertTicketQuery,

new

{

ScreeningId = screeningId,

SeatId = seatId,

Price = screening.Price

},

transaction: transaction);

}

transaction.Commit();

return Result.Success();

}

catch

{

transaction.Rollback();

return Result.Failure();

}

finally

{

connection.Close();

}

}

public IEnumerable<Screening> GetScreenings()

{

const string query = @"SELECT DISTINCT

s.Id,

m.DisplayName AS MovieName,

h.DisplayName AS HallName,

t.Price,

s.StartDateTime,

s.EndDateTime

FROM ((Screenings s

INNER JOIN Movies m ON s.MovieId = m.Id)

INNER JOIN Halls h ON s.HallId = h.Id)

INNER JOIN Tickets t ON t.ScreeningId = s.Id";

using var connection = new OleDbConnection(\_connectionString);

connection.Open();

var result = connection.Query<Screening>(query);

connection.Close();

return result;

}

public Result Update(Screening screening)

{

const string updateTicketsQuery = @"UPDATE Tickets SET

Price = @Price

WHERE ScreeningId = @ScreeningId AND PurchaseDateTime IS NULL;";

using var connection = new OleDbConnection(\_connectionString);

connection.Open();

using var transaction = connection.BeginTransaction();

try

{

var updateScreeningQuery = $@"UPDATE Screenings SET

StartDateTime = '{screening.StartDateTime}',

EndDateTime = '{screening.EndDateTime}'

WHERE Id = {screening.Id};";

connection.Execute(updateScreeningQuery,

transaction: transaction);

connection.Execute(updateTicketsQuery,

new

{

Price = screening.Price,

ScreeningId = screening.Id

},

transaction: transaction);

transaction.Commit();

return Result.Success();

}

catch

{

transaction.Rollback();

return Result.Failure();

}

finally

{

connection.Close();

}

}

public Result Delete(Screening screening)

{

const string deleteTicketsQuery = "DELETE FROM Tickets WHERE ScreeningId = @ScreeningId;";

const string deleteScreeningQuery = "DELETE FROM Screenings WHERE Id = @Id;";

using var connection = new OleDbConnection(\_connectionString);

connection.Open();

using var transaction = connection.BeginTransaction();

try

{

connection.Execute(deleteTicketsQuery,

new

{

ScreeningId = screening.Id

},

transaction: transaction);

connection.Execute(deleteScreeningQuery,

new

{

screening.Id

},

transaction: transaction);

transaction.Commit();

return Result.Success();

}

catch

{

transaction.Rollback();

return Result.Failure();

}

finally

{

connection.Close();

}

}

public IEnumerable<(int Row, int SeatNumber, int TicketId)> GetSeatTickets(int scrId)

{

using var connection = new OleDbConnection(\_connectionString);

connection.Open();

var result = connection.Query<(int, int, int)>(

$"SELECT s.Row, s.SeatNumber, t.Id FROM (Tickets t INNER JOIN Seats s ON t.SeatId = s.Id) WHERE t.ScreeningId = {scrId} AND t.PurchaseDateTime IS NULL");

connection.Close();

return result;

}

}

Файл "CassaForm.cs":

﻿using Core.Genres.Repositories;

using Core.Halls.Repositories;

using Core.MenuItems.Repositories;

using Core.Movies.Repositories;

using Core.Promotions.Repositories;

using Core.Screenings.Repositories;

using Core.Screenings;

using Infrastructure.Genres;

using Infrastructure.Halls;

using Infrastructure.MenuItems;

using Infrastructure.Movies;

using Infrastructure.Promotions;

using Infrastructure.Screenings;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using UI.Helpers;

using Core.Purchases.Repositories;

using Infrastructure.Purchases;

using Core.Purchases;

namespace UI

{

public partial class CassaForm : Form

{

private readonly IMovieRepository \_movieRepository;

private readonly IHallRepository \_hallRepository;

private readonly IMenuItemRepository \_menuItemRepository;

private readonly IPromotionRepository \_promotionRepository;

private readonly IScreeningRepository \_screeningRepository;

private readonly IPurchaseRepository \_purchaseRepository;

private IEnumerable<Screening> \_screenings;

private IEnumerable<(int Row, int SeatNumber, int TicketId)> values;

public CassaForm()

{

InitializeComponent();

var connectionStrings = ConnectionStringGetter.ConnectionStrings;

\_movieRepository = new MovieRepository(connectionStrings);

\_hallRepository = new HallRepository(connectionStrings);

\_menuItemRepository = new MenuItemRepository(connectionStrings);

\_promotionRepository = new PromotionRepository(connectionStrings);

\_screeningRepository = new ScreeningRepository(connectionStrings);

\_purchaseRepository = new PurchaseRepository(connectionStrings);

\_screenings = \_screeningRepository.GetScreenings();

comboBox1.Items.AddRange(\_screenings.Select(s => $"{s.MovieName} {s.HallName}").ToArray());

comboBox3.Items.AddRange(\_menuItemRepository.GetMenuItems().Select(m => m.DisplayName).ToArray());

comboBox4.Items.AddRange(\_promotionRepository.GetPromotions().Select(p => p.DisplayName).ToArray());

}

private void button1\_Click(object sender, EventArgs e)

{

var menuName = comboBox3.SelectedItem as string;

var promotion = comboBox4.SelectedItem as string;

var ticketId = values.First(v => $"{v.Row} Ряд {v.SeatNumber} место" == comboBox2.SelectedItem!.ToString()).TicketId;

var totalPrice = \_purchaseRepository.Add(new Purchase(ticketId, menuName, promotion));

MessageBox.Show($"Итоговая цена: {totalPrice}");

Close();

}

private void change\_movie(object sender, EventArgs e)

{

// Get tickets

comboBox2.Items.Clear();

var id = \_screenings.First(s => $"{s.MovieName} {s.HallName}" == comboBox1.SelectedItem!.ToString()).Id;

values = \_screeningRepository.GetSeatTickets(id);

comboBox2.Items.AddRange(values.Select(s => $"{s.Row} Ряд {s.SeatNumber} место").ToArray());

}

}

}

Файл "Form1.cs":

using Core.Genres.Repositories;

using Core.Halls.Repositories;

using Core.MenuItems;

using Core.MenuItems.Repositories;

using Core.Movies;

using Core.Movies.Repositories;

using Core.Promotions.Repositories;

using Core.Screenings;

using Core.Screenings.Repositories;

using Infrastructure.Genres;

using Infrastructure.Halls;

using Infrastructure.MenuItems;

using Infrastructure.Movies;

using Infrastructure.Promotions;

using Infrastructure.Screenings;

using System.Data;

using UI.AddData;

using UI.ChangeData;

using UI.Helpers;

namespace UI;

public partial class Form1 : Form

{

private readonly IMovieRepository \_movieRepository;

private readonly IGenreRepository \_genreRepository;

private readonly IHallRepository \_hallRepository;

private readonly IMenuItemRepository \_menuItemRepository;

private readonly IPromotionRepository \_promotionRepository;

private readonly IScreeningRepository \_screeningRepository;

private IEnumerable<Screening> \_screenings;

private Table? \_table;

public Form1()

{

InitializeComponent();

var connectionStrings = ConnectionStringGetter.ConnectionStrings;

\_movieRepository = new MovieRepository(connectionStrings);

\_genreRepository = new GenreRepository(connectionStrings);

\_hallRepository = new HallRepository(connectionStrings);

\_menuItemRepository = new MenuItemRepository(connectionStrings);

\_promotionRepository = new PromotionRepository(connectionStrings);

\_screeningRepository = new ScreeningRepository(connectionStrings);

}

private void moviesToolStripMenuItem1\_Click(object sender, EventArgs e)

{

mainTable.Visible = true;

createButton.Visible = true;

updateAndShowSeatsButton.Visible = true;

updateAndShowSeatsButton.Text = "Èçìåíèòü";

deleteButton.Visible = true;

\_table = Table.Movie;

UpatedMovieTable();

}

private void UpatedMovieTable()

{

var dataTable = InitialTable("Íàèìåíîâàíèå", "Ðåéòèíã", "Äëèòåëüíîñòü", "Ñòðàíà", "Æàíðû");

var movies = \_movieRepository.GetMovies();

foreach (var movie in movies)

{

dataTable.Rows.Add(movie.DisplayName, movie.Rating, movie.Duration, movie.Country, movie.Genres);

}

mainTable.DataSource = dataTable;

}

private void UpatedGenreTable()

{

var dataTable = InitialTable("Íàèìåíîâàíèå");

var genres = \_genreRepository.GetGenres();

foreach (var genre in genres)

{

dataTable.Rows.Add(genre.DisplayName);

}

mainTable.DataSource = dataTable;

}

private void UpatedHallsTable()

{

var dataTable = InitialTable("Íàèìåíîâàíèå", "Âìåñòèìîñòü");

var halls = \_hallRepository.GetHalls();

foreach (var hall in halls)

{

dataTable.Rows.Add(hall.DisplayName, hall.Capacity);

}

mainTable.DataSource = dataTable;

}

private DataTable InitialTable(params string[] columnsName)

{

var dataTable = new DataTable();

foreach (string columnName in columnsName)

{

dataTable.Columns.Add(columnName);

}

return dataTable;

}

private void genresToolStripMenuItem1\_Click(object sender, EventArgs e)

{

mainTable.Visible = true;

createButton.Visible = true;

updateAndShowSeatsButton.Visible = true;

updateAndShowSeatsButton.Text = "Èçìåíèòü";

deleteButton.Visible = true;

\_table = Table.Genres;

UpatedGenreTable();

}

private void createButton\_Click(object sender, EventArgs e)

{

switch (\_table)

{

case Table.Movie:

var addForm = new AddMovieForm();

addForm.ShowDialog();

UpatedMovieTable();

break;

case Table.Genres:

var addGenreForm = new AddGenreForm();

addGenreForm.ShowDialog();

UpatedGenreTable();

break;

case Table.HallManager:

var addHallForm = new AddHallForm();

addHallForm.ShowDialog();

UpatedHallsTable();

break;

case Table.Menu:

var addMenuForm = new AddMenuItemForm();

addMenuForm.ShowDialog();

UpatedMenuTable();

break;

case Table.Promotions:

var addPromForm = new AddPromotionForm();

addPromForm.ShowDialog();

UpatedPromotionTable();

break;

case Table.Screening:

var addScr = new AddScreeningForm();

addScr.ShowDialog();

UpatedScreeningTable();

break;

default:

MessageBox.Show("Êàê âû íàæàëè íà êíîïêó?");

break;

}

}

private void updateButton\_Click(object sender, EventArgs e)

{

if (mainTable.SelectedRows.Count == 0 && \_table == Table.HallManager)

{

MessageBox.Show("Íóæíî âûáðàòü ñòðîêó, ÷òîáû ïîñìîòðåòü ìåñòà ó çàëà.");

return;

}

if (mainTable.SelectedRows.Count == 0)

{

MessageBox.Show("Íóæíî âûáðàòü ñòðîêó êîòîðóþ õîòèòå èçìåíèòü.");

return;

}

var selectedRow = mainTable.SelectedRows[0].Cells;

switch (\_table)

{

case Table.Movie:

var movie = new Movie(selectedRow[0].Value.ToString()!, int.Parse(selectedRow[1].Value.ToString()!), TimeSpan.Parse(selectedRow[2].Value.ToString()!), selectedRow[3].Value.ToString(), selectedRow[4].Value.ToString());

var addForm = new ChangeMovieForm(movie);

addForm.ShowDialog();

UpatedMovieTable();

break;

case Table.Genres:

var addGenreForm = new ChangeGenreForm(selectedRow[0].Value.ToString()!);

addGenreForm.ShowDialog();

UpatedGenreTable();

break;

case Table.HallManager:

new SeatsForm(selectedRow[0].Value.ToString()!).ShowDialog();

break;

case Table.Menu:

var menu = new MenuItem(selectedRow[0].Value.ToString()!, int.Parse(selectedRow[1].Value.ToString()!), bool.Parse(selectedRow[2].Value.ToString()!));

var addMenuForm = new ChangeMenuItemForm(menu);

addMenuForm.ShowDialog();

UpatedMenuTable();

break;

case Table.Screening:

var scr = \_screenings.First(s => s.MovieName == selectedRow[0].Value.ToString()! && s.HallName == selectedRow[1].Value.ToString()!);

var form = new ChangeScreeningForm(scr);

form.ShowDialog();

UpatedScreeningTable();

break;

default:

MessageBox.Show("Êàê âû íàæàëè íà êíîïêó?");

break;

}

}

private void deleteButton\_Click(object sender, EventArgs e)

{

if (mainTable.SelectedRows.Count == 0)

{

MessageBox.Show("Íóæíî âûáðàòü ñòðîêó êîòîðóþ õîòèòå óäàëèòü");

return;

}

var selectedRow = mainTable.SelectedRows[0].Cells;

switch (\_table)

{

case Table.Movie:

var movie = new Movie(selectedRow[0].Value.ToString()!, int.Parse(selectedRow[1].Value.ToString()!), TimeSpan.Parse(selectedRow[2].Value.ToString()!), selectedRow[3].Value.ToString(), selectedRow[4].Value.ToString());

\_movieRepository.Delete(movie);

UpatedMovieTable();

break;

case Table.Genres:

var result = \_genreRepository.Delete(selectedRow[0].Value.ToString()!);

if (!result.IsSuccess)

{

MessageBox.Show("Íåâîçìîæíî óäàëèòü æàíðû, êîòîðûå ïðèâÿçàíû ê ôèëüìàì.");

return;

}

UpatedGenreTable();

break;

case Table.HallManager:

\_hallRepository.Delete(selectedRow[0].Value.ToString()!);

UpatedHallsTable();

break;

case Table.Menu:

\_menuItemRepository.Delete(selectedRow[0].Value.ToString()!);

UpatedMenuTable();

break;

case Table.Promotions:

\_promotionRepository.Delete(selectedRow[0].Value.ToString()!);

UpatedPromotionTable();

break;

case Table.Screening:

\_screeningRepository.Delete(\_screenings.First(s => s.MovieName == selectedRow[0].Value.ToString()! && s.HallName == selectedRow[1].Value.ToString()!));

UpatedScreeningTable();

break;

default:

MessageBox.Show("Êàê âû íàæàëè íà êíîïêó?");

break;

}

}

private void hallManagerToolStripMenuItem\_Click(object sender, EventArgs e)

{

mainTable.Visible = true;

createButton.Visible = true;

updateAndShowSeatsButton.Visible = true;

updateAndShowSeatsButton.Text = "Ìåñòà ó çàëà";

deleteButton.Visible = true;

\_table = Table.HallManager;

UpatedHallsTable();

}

private void menuToolStripMenuItem\_Click(object sender, EventArgs e)

{

mainTable.Visible = true;

createButton.Visible = true;

updateAndShowSeatsButton.Visible = true;

updateAndShowSeatsButton.Text = "Èçìåíèòü";

deleteButton.Visible = true;

\_table = Table.Menu;

UpatedMenuTable();

}

private void UpatedMenuTable()

{

var dataTable = InitialTable("Íàèìåíîâàíèå", "Öåíà", "Íàëè÷èå");

var menuItems = \_menuItemRepository.GetMenuItems();

foreach (var menuItem in menuItems)

{

dataTable.Rows.Add(menuItem.DisplayName, menuItem.Price, menuItem.IsAvailable);

}

mainTable.DataSource = dataTable;

}

private void UpatedPromotionTable()

{

var dataTable = InitialTable("Íàèìåíîâàíèå", "Ïðîöåíò ñêèäêè", "Íà÷àëî ñêèäêè", "Êîíåö ñêèäêè");

var promotions = \_promotionRepository.GetPromotions();

foreach (var promotion in promotions)

{

dataTable.Rows.Add(promotion.DisplayName, promotion.DiscountPercent, promotion.StartDateTime, promotion.EndDateTime);

}

mainTable.DataSource = dataTable;

}

private void promotionToolStripMenuItem\_Click(object sender, EventArgs e)

{

mainTable.Visible = true;

createButton.Visible = true;

updateAndShowSeatsButton.Visible = false;

deleteButton.Visible = true;

\_table = Table.Promotions;

UpatedPromotionTable();

}

private void screeningToolStripMenuItem\_Click(object sender, EventArgs e)

{

mainTable.Visible = true;

createButton.Visible = true;

updateAndShowSeatsButton.Visible = true;

updateAndShowSeatsButton.Text = "Èçìåíèòü";

deleteButton.Visible = true;

\_table = Table.Screening;

UpatedScreeningTable();

}

private void UpatedScreeningTable()

{

var dataTable = InitialTable("Ôèëüì", "Çàë", "Ñòîèìîñòü", "Äàòà íà÷àëî", "Äàòà êîíåö");

\_screenings = \_screeningRepository.GetScreenings();

foreach (var screening in \_screenings)

{

dataTable.Rows.Add(screening.MovieName, screening.HallName, screening.Price, screening.StartDateTime, screening.EndDateTime);

}

mainTable.DataSource = dataTable;

}

private void cassaToolStripMenuItem\_Click(object sender, EventArgs e)

{

var form = new CassaForm();

form.ShowDialog();

}

private enum Table

{

Movie,

Genres,

HallManager,

Menu,

Promotions,

Screening

}

}

Файл "Program.cs":

namespace UI;

internal static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main()

{

// To customize application configuration such as set high DPI settings or default font,

// see https://aka.ms/applicationconfiguration.

ApplicationConfiguration.Initialize();

Application.Run(new Form1());

}

}

Файл "SeatsForm.cs":

﻿using Core.Halls.Repositories;

using Infrastructure.Halls;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using UI.Helpers;

namespace UI

{

public partial class SeatsForm : Form

{

private readonly IHallRepository \_hallRepository;

public SeatsForm(string hallName)

{

InitializeComponent();

var connectionsStrings = ConnectionStringGetter.ConnectionStrings;

\_hallRepository = new HallRepository(connectionsStrings);

var dataTable = InitialTable("Ряд", "Номер места");

var seatsByHall = \_hallRepository.GetAllSeatsByHallName(hallName);

foreach (var seat in seatsByHall)

{

dataTable.Rows.Add(seat.Row, seat.SeatNumber);

}

seats.DataSource = dataTable;

}

private DataTable InitialTable(params string[] columnsName)

{

var dataTable = new DataTable();

foreach (string columnName in columnsName)

{

dataTable.Columns.Add(columnName);

}

return dataTable;

}

}

}

Файл "AddGenreForm.cs":

﻿using Core.Genres;

using Core.Genres.Repositories;

using Infrastructure.Genres;

using Infrastructure.Movies;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using UI.Helpers;

namespace UI.AddData

{

public partial class AddGenreForm : Form

{

private readonly IGenreRepository \_genreRepository;

public AddGenreForm()

{

var connectionsStrings = ConnectionStringGetter.ConnectionStrings;

\_genreRepository = new GenreRepository(connectionsStrings);

InitializeComponent();

}

private void button2\_Click(object sender, EventArgs e)

{

Close();

}

private void button1\_Click(object sender, EventArgs e)

{

if (string.IsNullOrWhiteSpace(Name.Text))

{

MessageBox.Show("Наименование не может быть пустым или из пробелов.");

return;

}

var result = \_genreRepository.Add(new Genre(Name.Text));

if(result.IsSuccess)

{

Close();

return;

}

MessageBox.Show("Либо что-то пошло не так, либо вы пытаетесь создать жанр с наименованием, которое уже существует.");

}

}

}

Файл "AddHallForm.cs":

﻿using Core.Genres;

using Core.Halls;

using Core.Halls.Repositories;

using Core.Movies;

using Infrastructure.Halls;

using Infrastructure.Movies;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using UI.Helpers;

using static System.Windows.Forms.VisualStyles.VisualStyleElement;

namespace UI.AddData

{

public partial class AddHallForm : Form

{

private readonly IHallRepository \_hallRepository;

public AddHallForm()

{

InitializeComponent();

var connectionsStrings = ConnectionStringGetter.ConnectionStrings;

\_hallRepository = new HallRepository(connectionsStrings);

}

private void button1\_Click(object sender, EventArgs e)

{

if (string.IsNullOrWhiteSpace(Name.Text) || string.IsNullOrWhiteSpace(capacity.Text))

{

MessageBox.Show("Данные не могут быть пустыми.");

return;

}

if (!int.TryParse(capacity.Text, out var result))

{

MessageBox.Show("Вместимость должно быть числом.");

return;

}

var hall = new Hall(Name.Text, result);

var res = \_hallRepository.Add(hall);

if(!res.IsSuccess)

{

MessageBox.Show("Либо что-то произошло, либо вы пытаетесь создать зал, который уже существует.");

return;

}

Close();

}

private void button2\_Click(object sender, EventArgs e)

{

Close();

}

}

}

Файл "AddMenuItemForm.cs":

﻿using Core.Halls.Repositories;

using Core.Halls;

using Core.MenuItems.Repositories;

using Infrastructure.Halls;

using Infrastructure.MenuItems;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using UI.Helpers;

using Core.MenuItems;

namespace UI.AddData

{

public partial class AddMenuItemForm : Form

{

private IMenuItemRepository \_menuItemRepository;

public AddMenuItemForm()

{

InitializeComponent();

var connectionsStrings = ConnectionStringGetter.ConnectionStrings;

\_menuItemRepository = new MenuItemRepository(connectionsStrings);

}

private void button1\_Click(object sender, EventArgs e)

{

if (string.IsNullOrWhiteSpace(Name.Text) || string.IsNullOrWhiteSpace(price.Text))

{

MessageBox.Show("Данные не могут быть пустыми.");

return;

}

if (!int.TryParse(price.Text, out var result))

{

MessageBox.Show("Цена должна быть числом.");

return;

}

var menuItem = new MenuItem(Name.Text, result);

var res = \_menuItemRepository.Add(menuItem);

if (!res.IsSuccess)

{

MessageBox.Show("Либо что-то произошло, либо вы пытаетесь создать продукт, который уже существует.");

return;

}

Close();

}

private void button2\_Click(object sender, EventArgs e)

{

Close();

}

}

}

Файл "AddMovieForm.cs":

﻿using Core.Genres.Repositories;

using Core.Movies;

using Core.Movies.Repositories;

using Infrastructure.Genres;

using Infrastructure.Movies;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using UI.Helpers;

namespace UI.AddData

{

public partial class AddMovieForm : Form

{

private readonly IMovieRepository \_movieRepository;

private readonly IGenreRepository \_genreRepository;

public AddMovieForm()

{

InitializeComponent();

var connectionsStrings = ConnectionStringGetter.ConnectionStrings;

\_movieRepository = new MovieRepository(connectionsStrings);

\_genreRepository = new GenreRepository(connectionsStrings);

genres.Items.Clear();

genres.Items.AddRange(\_genreRepository.GetGenres().Select(g => g.DisplayName).ToArray());

}

private void button2\_Click(object sender, EventArgs e)

{

Close();

}

private void button1\_Click(object sender, EventArgs e)

{

if(string.IsNullOrWhiteSpace(Name.Text) || string.IsNullOrWhiteSpace(textBox2.Text))

{

MessageBox.Show("Данные кроме страны и жанров не могут быть пустыми.");

return;

}

var genre = genres.SelectedItem?.ToString() ?? string.Empty;

var rating = int.Parse(comboBox1.SelectedItem!.ToString()!);

var movie = new Movie(Name.Text, rating, TimeSpan.Parse(textBox2.Text), textBox1.Text, genre);

\_movieRepository.Add(movie);

Close();

}

}

}

Файл "AddPromotionForm.cs":

﻿using Core.MenuItems;

using Core.Promotions;

using Core.Promotions.Repositories;

using Infrastructure.Promotions;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Diagnostics;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using UI.Helpers;

namespace UI.AddData

{

public partial class AddPromotionForm : Form

{

private readonly IPromotionRepository \_promotionRepository;

public AddPromotionForm()

{

InitializeComponent();

var connectionsStrings = ConnectionStringGetter.ConnectionStrings;

\_promotionRepository = new PromotionRepository(connectionsStrings);

}

private void button2\_Click(object sender, EventArgs e)

{

Close();

}

private void button1\_Click(object sender, EventArgs e)

{

if (string.IsNullOrWhiteSpace(Name.Text) || string.IsNullOrWhiteSpace(textBox1.Text))

{

MessageBox.Show("Данные не могут быть пустыми.");

return;

}

if (!int.TryParse(textBox1.Text, out var result) && (result > 100 || result < 0))

{

MessageBox.Show("Цена должна быть числом и меньше 100 и больше 0.");

return;

}

var promotion = new Promotion(Name.Text, result, dateTimePicker1.Value, dateTimePicker2.Value);

var res = \_promotionRepository.Add(promotion);

if (!res.IsSuccess)

{

MessageBox.Show("Либо что-то произошло, либо вы пытаетесь создать скидку, которая уже существует.");

return;

}

Close();

}

}

}

Файл "AddScreeningForm.cs":

﻿using Core.Screenings;

using Core.Screenings.Repositories;

using Infrastructure.Halls;

using Infrastructure.MenuItems;

using Infrastructure.Movies;

using Infrastructure.Options;

using Infrastructure.Promotions;

using Infrastructure.Screenings;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Diagnostics;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using UI.Helpers;

using static System.Windows.Forms.VisualStyles.VisualStyleElement;

namespace UI.AddData

{

public partial class AddScreeningForm : Form

{

private readonly IScreeningRepository \_screeningRepository;

public AddScreeningForm()

{

InitializeComponent();

var connectionStrings = ConnectionStringGetter.ConnectionStrings;

var \_hallRepository = new HallRepository(connectionStrings);

var \_movieRepository = new MovieRepository(connectionStrings);

\_screeningRepository = new ScreeningRepository(connectionStrings);

comboBox1.Items.AddRange(\_movieRepository.GetMovies().Select(m => m.DisplayName).ToArray());

comboBox2.Items.AddRange(\_hallRepository.GetHalls().Select(h => h.DisplayName).ToArray());

}

private void button1\_Click(object sender, EventArgs e)

{

if (string.IsNullOrWhiteSpace(textBox1.Text))

{

MessageBox.Show("Цена не должна быть пустой.");

return;

}

if (!int.TryParse(textBox1.Text, out var result))

{

MessageBox.Show("Цена должна быть числом.");

return;

}

var screening = new Screening(0, comboBox1.SelectedItem!.ToString()!, comboBox2.SelectedItem!.ToString()!, result, dateTimePicker1.Value, dateTimePicker2.Value);

\_screeningRepository.Add(screening);

Close();

}

private void button2\_Click(object sender, EventArgs e)

{

Close();

}

}

}

Файл "ChangeGenreForm.cs":

﻿using Core.Genres;

using Core.Genres.Repositories;

using Infrastructure.Genres;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using UI.Helpers;

namespace UI.ChangeData

{

public partial class ChangeGenreForm : Form

{

private readonly string \_genreOldName;

private readonly IGenreRepository \_genreRepository;

public ChangeGenreForm(string oldName)

{

InitializeComponent();

Name.Text = oldName;

\_genreOldName = oldName;

var connectionsStrings = ConnectionStringGetter.ConnectionStrings;

\_genreRepository = new GenreRepository(connectionsStrings);

}

private void button2\_Click(object sender, EventArgs e)

{

Close();

}

private void button1\_Click(object sender, EventArgs e)

{

if (string.IsNullOrWhiteSpace(Name.Text))

{

MessageBox.Show("Наименование не может быть пустым или из пробелов.");

return;

}

var result = \_genreRepository.Update(\_genreOldName, new Genre(Name.Text));

if (result.IsSuccess)

{

Close();

return;

}

MessageBox.Show("Либо что-то пошло не так, либо вы пытаетесь создать жанр с наименованием, которое уже существует.");

}

}

}

Файл "ChangeMenuItemForm.cs":

﻿using Core.MenuItems;

using Core.MenuItems.Repositories;

using Infrastructure.MenuItems;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Diagnostics;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using UI.Helpers;

namespace UI.ChangeData

{

public partial class ChangeMenuItemForm : Form

{

private MenuItem \_oldData;

private IMenuItemRepository \_menuItemRepository;

public ChangeMenuItemForm(MenuItem menuItem)

{

InitializeComponent();

\_oldData = menuItem;

Name.Text = menuItem.DisplayName;

Price.Text = menuItem.Price.ToString();

checkBox1.Checked = menuItem.IsAvailable;

var connectionsStrings = ConnectionStringGetter.ConnectionStrings;

\_menuItemRepository = new MenuItemRepository(connectionsStrings);

}

private void button2\_Click(object sender, EventArgs e)

{

Close();

}

private void button1\_Click(object sender, EventArgs e)

{

if (string.IsNullOrWhiteSpace(Name.Text) || string.IsNullOrWhiteSpace(Price.Text))

{

MessageBox.Show("Данные не могут быть пустыми.");

return;

}

if (!int.TryParse(Price.Text, out var result))

{

MessageBox.Show("Цена должна быть числом.");

return;

}

var menuItem = new MenuItem(Name.Text, result, checkBox1.Checked);

var res = \_menuItemRepository.Update(\_oldData.DisplayName, menuItem);

if (!res.IsSuccess)

{

MessageBox.Show("Либо что-то произошло, либо вы пытаетесь изменить продукт на тот, который уже существует.");

return;

}

Close();

}

}

}

Файл "ChangeMovieForm.cs":

﻿using Core.Genres.Repositories;

using Core.Movies;

using Core.Movies.Repositories;

using Infrastructure.Genres;

using Infrastructure.Movies;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using UI.Helpers;

namespace UI.ChangeData

{

public partial class ChangeMovieForm : Form

{

private readonly Movie \_oldData;

private readonly IMovieRepository \_movieRepository;

private readonly IGenreRepository \_genreRepository;

public ChangeMovieForm(Movie movie)

{

InitializeComponent();

Name.Text = movie.DisplayName;

textBox1.Text = movie.Country;

textBox2.Text = movie.Duration.ToString();

comboBox1.SelectedItem = movie.Rating.ToString();

var connectionsStrings = ConnectionStringGetter.ConnectionStrings;

\_movieRepository = new MovieRepository(connectionsStrings);

\_genreRepository = new GenreRepository(connectionsStrings);

genres.Items.Clear();

genres.Items.AddRange(\_genreRepository.GetGenres().Select(g => g.DisplayName).ToArray());

genres.SelectedItem = movie.Genres;

\_oldData = new Movie(movie.DisplayName, movie.Rating, movie.Duration, movie.Country, movie.Genres);

}

private void button2\_Click(object sender, EventArgs e)

{

Close();

}

private void button1\_Click(object sender, EventArgs e)

{

if (string.IsNullOrWhiteSpace(Name.Text) || string.IsNullOrWhiteSpace(textBox2.Text))

{

MessageBox.Show("Данные кроме страны и жанров не могут быть пустыми.");

return;

}

var genre = genres.SelectedItem?.ToString() ?? string.Empty;

var rating = int.Parse(comboBox1.SelectedItem!.ToString()!);

var movie = new Movie(Name.Text, rating, TimeSpan.Parse(textBox2.Text), textBox1.Text, genre);

var result = \_movieRepository.Update(\_oldData, movie);

if(result.IsSuccess)

{

Close();

return;

}

MessageBox.Show("Чтобы изменить фильм нужно, чтобы хотя один параметр был уникален.");

}

}

}

Файл "ChangeScreeningForm.cs":

﻿using Core.Screenings;

using Core.Screenings.Repositories;

using Infrastructure.Options;

using Infrastructure.Screenings;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using UI.Helpers;

namespace UI.ChangeData

{

public partial class ChangeScreeningForm : Form

{

private readonly IScreeningRepository \_screeningRepository;

private Screening \_oldData;

public ChangeScreeningForm(Screening screening)

{

InitializeComponent();

\_oldData = screening;

textBox1.Text = screening.Price.ToString();

dateTimePicker1.Value = screening.StartDateTime;

dateTimePicker2.Value = screening.EndDateTime;

var connectionStrings = ConnectionStringGetter.ConnectionStrings;

\_screeningRepository = new ScreeningRepository(connectionStrings);

}

private void button1\_Click(object sender, EventArgs e)

{

if (string.IsNullOrWhiteSpace(textBox1.Text))

{

MessageBox.Show("Цена не должна быть пустой.");

return;

}

if (!int.TryParse(textBox1.Text, out var result))

{

MessageBox.Show("Цена должна быть числом.");

return;

}

var screening = new Screening(\_oldData.Id, \_oldData.MovieName, \_oldData.HallName, result, dateTimePicker1.Value, dateTimePicker2.Value);

\_screeningRepository.Update(screening);

Close();

}

private void button2\_Click(object sender, EventArgs e)

{

Close();

}

}

}

Файл "ConnectionStringGetter.cs":

﻿using System.IO;

using System.Text.Json;

using Infrastructure.Options;

namespace UI.Helpers;

public static class ConnectionStringGetter

{

public static readonly ConnectionStrings ConnectionStrings;

static ConnectionStringGetter()

{

var settings = JsonDocument.Parse(File.ReadAllText("appsettings.json"));

ConnectionStrings = settings.RootElement.GetProperty("ConnectionStrings").Deserialize<ConnectionStrings>()

?? throw new InvalidOperationException("Необходимо вставить строку подключения для работы с приложением.");

}

}