

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт (факультет)

Институт информационных технологий

Кафедра

Математического и программного обеспечения ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине

Проектирование баз данных

на тему

Проектирование базы данных «Звёздный экран»

Выполнил студент группы

1ИСб-01-2оп-22

направление подготовки (специальности)

09.03.02, Информационные системы и
технологии

шифр, наименование

Сухарев Евгений Сергеевич

фамилия, имя, отчество

Руководитель

Селяничев Олег Леонидович

фамилия, имя, отчество

Доцент, кандидат технических наук

должность

Дата представления работы

«__» _____ 2025 г.

Заключение о допуске к защите

Оценка _____, _____ количество баллов

Подпись преподавателя _____

Череповец, 2025
год

Оглавление

Введение	3
1. Описание предметной области.....	4
2. Инфологическое проектирование	5
3. Дatalogическое проектирование.....	7
3.1.Нормализация	7
3.1.1. Ненормализованная форма.....	8
3.1.2. Первая нормальная форма	11
3.1.3. Вторая нормальная форма	14
3.1.4. Третья нормальная форма.....	18
3.2. Логическая модель.....	22
3.3. Физическая модель	23
4. Описание программного обеспечения.....	31
Заключение.....	33
Список литературы.....	34
Приложение 1. Техническое задание.....	35
Приложение 2. Текст программы.....	43
Приложение 3. Руководство пользователя.....	57

Современные кинотеатры являются не только местом показа фильмов, но и полноценными центрами досуга, где зрители ожидают высокого уровня сервиса, комфортных условий и удобных способов выбора сеансов. Конкуренция в сфере развлечений постоянно растёт, и для успешной работы кинотеатрам необходимо предлагать не только разнообразный репертуар, но и качественно организованную систему управления всеми процессами - от планирования расписания до продажи билетов и обслуживания посетителей [1].

Тем не менее до сих пор встречаются кинотеатры, где учёт фильмов, бронирование и реализация билетов ведутся с помощью простых таблиц или устаревших программ. Такой подход ограничивает возможности персонала и приводит к ошибкам при формировании расписания и работе с клиентами. Это снижает скорость обслуживания и отрицательно сказывается на впечатлениях зрителей [1].

Отсутствие единой информационной системы создаёт трудности на уровне управления: становится сложно контролировать загрузку залов, своевременно предоставлять актуальную информацию и поддерживать высокий уровень качества сервиса. Подобные недостатки влияют не только на работу персонала, но и на финансовые результаты, а в перспективе могут ослабить позиции кинотеатра на рынке и негативно сказаться на его репутации.

Таким образом, возникает необходимость разработки и внедрения современных инструментов, которые позволят повысить эффективность организации работы кинотеатра, снизить вероятность ошибок и обеспечить высокий уровень обслуживания посетителей.

1. Описание предметной области

Кинотеатр «Звёздный экран» работает круглосуточно и представляет собой организацию, деятельность которой направлена на проведение киносеансов и предоставление сопутствующих услуг. В рамках данной области мы рассматриваем залы различного уровня, билеты, сеансы, зрителей и персонал, обеспечивающий функционирование всех процессов.

Формирование расписания сеансов осуществляется заранее и связано с распределением фильмов по залам и временным интервалам. Каждый сеанс характеризуется названием фильма, временем начала, залом, в котором он демонстрируется, и количеством доступных мест. Залы отличаются уровнем комфорта и вместимостью, что позволяет предлагать различные тарифы и обеспечивать выбор для зрителей с разными предпочтениями.

Информация о расписании доступна посетителям при обращении в кассу или при использовании онлайн-сервисов. На основании этой информации зритель выбирает фильм, время и зал, после чего указывает количество билетов. Сведения о бронировании или покупке фиксируются кассиром и отражаются в системе, что позволяет контролировать наличие свободных мест и упрощает процесс обслуживания.

Администратор играет важную роль в обеспечении согласованной работы всех элементов кинотеатра. Он контролирует корректность информации о сеансах и размещении зрителей, координирует работу кассиров и технического персонала, помогает посетителям с ориентацией в залах и обеспечивает соблюдение правил внутреннего порядка. Технический персонал, в свою очередь, отвечает за исправное функционирование оборудования и поддержание комфортных условий просмотра.

Все перечисленные объекты и процессы формируют структуру работы кинотеатра и обеспечивают возможность организации сеансов и обслуживания зрителей в круглосуточном режиме.

2. Инфологическое проектирование

Инфологическое проектирование представляет собой начальный этап создания базы данных, на котором осуществляется концептуальное моделирование предметной области без привязки к конкретной системе управления базами данных (СУБД). Основная цель данного этапа - создание формального описания структуры данных, отражающего реальные объекты и процессы исследуемой области [2].

Сущность – это существующие в действительности или воображаемые явления или объект, информацию о котором нужно сохранять или выяснять.

Атрибут (свойства) – именованный элемент информации, описывающий сущность. Атрибут может иметь только одно значение в каждый момент [2].

Между сущностями существует некая ассоциация, которая называется связью. Одна сущность может быть связана с другой сущностью или сама с собой. Графически связь изображается линией, соединяющей две сущности. Каждая связь характеризуется именем, типом, классом принадлежности и направлением. Различают четыре типа связи: «один к одному» (1:1); «один ко многим» (1:M); «многие к одному» (M:1) «многие ко многим» (M:M) [2].

В теории баз данных существует несколько основных моделей данных, каждая из которых имеет свои особенности и области применения:

Иерархическая модель данных организует информацию в виде древовидной структуры, где каждый элемент может иметь только одного родителя. Эта модель эффективна для представления строго структурированных данных с четкой иерархией, но ограничена в представлении сложных взаимосвязей между объектами [2].

Сетевая модель данных расширяет возможности иерархической модели, позволяя элементам иметь несколько родителей. Она более гибкая в представлении сложных связей, но требует значительных усилий для навигации и поддержания целостности данных [2].

Реляционная модель данных основана на математической теории отношений и представляет данные в виде таблиц (отношений). Каждая таблица состоит из строк (кортежей) и столбцов (атрибутов). Связи между таблицами устанавливаются через общие атрибуты (ключи) [2].

Объектно-ориентированная модель данных интегрирует концепции объектно-ориентированного программирования с управлением данными, позволяя хранить сложные объекты с их методами и наследованием [2].

NoSQL модели (документно-ориентированная, колоночная, графовая) предназначены для работы с большими объемами слабоструктурированных данных и обеспечивают высокую масштабируемость [2].

Для реализации базы данных выбран реляционный подход, так как он обеспечивает удобное представление структурированных данных в виде таблиц с четко определёнными атрибутами и ключами. Сеансы, билеты, залы и персонал могут быть представлены отдельными таблицами, а связи между ними - через первичные и внешние ключи. Такой подход упрощает контроль целостности

данных, позволяет эффективно обрабатывать запросы, обеспечивает возможность масштабирования и дальнейшего расширения функционала системы без нарушения структуры информации [2].

После изучения предметной области, анализа и систематизации информации нужно выделить главную сущность – «Сеанс» – и атрибуты:

- Фильм;
- Зал;
- Билет;
- Зритель;
- Кассир;
- Технический персонал;
- Администратор.

3. Даталогическое проектирование

Даталогическое проектирование – это этап разработки базы данных, на котором создается логическая структура данных, ориентированная на реализацию инфологической модели в конкретной системе управления базами данных. Основная цель этого этапа - определить, как данные будут структурированы и организованы, чтобы эффективно поддерживать процессы обработки и хранения информации в системе [3].

Процесс даталогического проектирования начинается с выявления сущностей, которые будут представлять основные объекты предметной области. На этом этапе сущности получают свои атрибуты - характеристики, которые нужно хранить, и связи между ними. Эти связи могут отражать различные типы взаимоотношений между объектами, такие как связи один ко многим или многие ко многим. Важной задачей является назначение ключей для каждой сущности, что позволит обеспечить уникальность записей и упростит работу с данными [3].

Кроме того, в процессе даталогического проектирования устанавливаются ограничения целостности данных, такие как требования к уникальности значений, корректности ссылок между таблицами и соблюдению других бизнес-правил. Эти ограничения гарантируют, что данные будут храниться и обрабатываться корректно, минимизируя ошибки и аномалии [3].

Даталогическое проектирование направлено на создание логической структуры базы данных, которая отражает бизнес-логику и требования системы, обеспечивая при этом надёжность, гибкость и эффективность работы с данными. Для повышения качества этой структуры и предотвращения избыточности на данном этапе применяется процесс нормализации [4], который помогает оптимизировать организацию данных и повысить их целостность.

3.1. Нормализация

Нормализация – это процесс преобразования структуры данных с целью минимизации избыточности и устранения возможных аномалий при манипуляциях с данными. Основная цель нормализации – это улучшение структуры базы данных таким образом, чтобы данные хранились эффективно и без дублирования, а также чтобы предотвратить логические ошибки, которые могут возникнуть из-за многократного повторения одной и той же информации [4].

Процесс нормализации включает в себя несколько этапов, и каждый из них связан с определенной нормальной формой, в которой должны находиться таблицы базы данных. Эти нормальные формы обеспечивают определенные правила, которые должны соблюдаться для исключения избыточности данных [4].

В ходе нормализации база данных разделяется на несколько таблиц с минимизацией избыточности данных в каждой таблице. Каждая таблица в нормализованной базе данных обычно фокусируется на одной сущности, что улучшает гибкость работы с данными и упрощает их обработку [4].

Каждой нормальной форме соответствует некоторый определенный набор ограничений, и отношение находится в некоторой нормальной форме, если

удовлетворяет свойственному ей набору ограничений [4].

Основные нормальные формы реляционной модели [4]:

- первая нормальная форма (1НФ);
- вторая нормальная форма (2НФ);
- третья нормальная форма (3НФ);
- нормальная форма Бойса-Кодда (НФБК);
- четвертая нормальная форма (4НФ);
- пятая нормальная форма (5НФ).

Для начала нормализации необходимо представить данные в виде ненормализованной таблицы.

3.1.1. Ненормализованная форма

Ненормализованная форма (ННФ) - это исходное представление данных, в котором вся информация по предметной области хранится в единой структуре без применения принципов нормализации [5]. В табл. 1 представлена информация о заказах в ненормализованной форме.

Ненормализованная форма

Таблица 1

1. Сеанс	2. Администратор	3. Технический персонал	4. Кассир
100000, Дюна: Часть вторая, фантастика, Зал Стандарт «Солнечный», Смирнов Сергей Николаевич, Иванов Сергей Петрович, 24.05.2025 13:45, 24.05.2025 15:45	Смирнов Сергей Николаевич, +7-906-678-90-12, smirnov@mail.ru, Ночная смена, Техническая зона	Иванов Сергей Петрович, +7-900-111-22-33, ivanov.tech@cinema.ru, Звукооператор, Дневная смена	Петрова Алена Игоревна, Утренняя смена, 8900111223, petrova@gmail.com
100002, Джокер: Безумие на двоих, Зал Премиум «Лазурный», Федорова Елена Михайловна, Кузнецов Алексей Дмитриевич, 24.05.2025 16:00:00, 24.05.2025 18:00	Федорова Елена Михайловна, +7-907-789-01-23, fedorova@mail.ru, Вечерняя смена, Кассовая зона	Кузнецов Алексей Дмитриевич, +7-901-222-33-44, kuznetsov.tech@cinema.ru, Светооператор, Утренняя смена	Иванов Иван Иванович, +79011234567, ivanov@mail.ru
100003, Холодное сердце 2, «Гранд», Морозов Илья Александрович, Смирнова Анна Викторовна, 25.05.2025 10:30, 25.05.2025 12:30	Морозов Илья Александрович, +7-908-890-12-34, morozov@mail.ru, Дневная смена, Общая координация	Смирнова Анна Викторовна, +7-902-333-44-55, smirnova.tech@cinema.ru, Электрик, Вечерняя смена	Сидоров Алексей Павлович, +7-903-345-67-89, sidorov@mail.ru
100004, Миссия невыполнима 7, Зал 3D «Вдохновение», Ковалева Наталья Сергеевна, Поляков Дмитрий Олегович, 25.05.2025 14:15, 25.05.2025 17:15	Ковалева Наталья Сергеевна, +7-909-901-23-45, kovaleva@mail.ru, Вечерняя смена, Фойе	Поляков Дмитрий Олегович, +7-903-444-55-66, polyakov.tech@cinema.ru, Киномеханик, Дневная смена	Васильев Дмитрий Андреевич, +7-905-567-89-01, vasiliev@mail.ru, Ночная смена
100005, Барби, Зал Стандарт «Элегант», Григорьев Олег Владимирович, Мельникова Елена Сергеевна, 26.05.2025 18:00, 26.05.2025 20:00	Григорьев Олег Владимирович, +7-910-012-34-56, grigoriev@mail.ru, Дневная смена, Залы	Мельникова Елена Сергеевна, +7-904-555-66-77, melnikova.tech@cinema.ru, Специалист по оборудованию зала, Ночная смена	Кузнецова Мария Викторовна, +7-904-456-78-90, kuznetsova@mail.ru

Продолжение табл. 1

5. Фильм	6. Зал	7. Билет	8. Зритель
Дюна: Часть вторая, фантастика, 155 мин, 16+	Стандартный, 120 человек, Зал Стандарт «Солнечный», 2 этаж	2 ряд, 21 место, семейный, 1000 рублей	Иванов Иван Иванович, 20 лет, студент
Джокер: Безумие на двоих, драма, 138 мин, 18+	Премиум, 150 человек, Зал Премиум «Лазурный», 3 этаж	6 ряд, 10 место, льготный, 250 рублей	Сидоров Петр Георгиевич, 30 лет, взрослый
Холодное сердце 2, мультфильм, 103 мин, 6+	IMAX, 100 человек, Зал IMAX «Гранд», 4 этаж	2 ряд, 4 место, комфорт, 600 рублей	Смирнов Никита Александрович, 12 лет, ребенок
Миссия невыполнима 7, боевик, 164 мин, 12+	3D, 50 человек, Зал 3D «Вдохновение», 1 этаж	3 ряд, 5 место, премьерный, 900 рублей	Петрова Аня Артемовна, 35 лет, взрослый
Барби, комедия, 114 мин, 12+	Зал Стандарт «Элегант», 150 человек, стандарт, 1 этаж	1 ряд, 2 место, стандарт, 450 рублей	Кузнецова Евгения Дмитриевна, 65 лет, пенсионер

Избыточность и дублирование данных в ненормализованной форме затрудняют их обновление, поиск и анализ, что снижает эффективность работы с информацией. Именно для устранения этих проблем и упорядочивания хранения информации применяется первая нормальная форма (1НФ).

3.1.2. Первая нормальная форма

Отношение находится в первой нормальной форме (1НФ), если все значения атрибутов являются атомарными (неделимыми), каждый кортеж уникален, а каждый атрибут имеет уникальное имя и описывает только один тип данных [5].

В представленной структуре атрибуты «Сеанс», «Администратор», «Технический персонал», «Кассир», «Фильм», «Зал», «Билет» и «Зритель» не соответствуют требованиям первой нормальной формы, так как содержат составные значения. Например:

- в одном атрибуте «Сеанс» одновременно указаны название фильма, жанр, зал, сотрудники и даты проведения, что делает значение неоднородным;
- в «Администраторе», «Техническом персонале» и «Кассире» объединены ФИО, контактные данные, должность и смена;
- в «Фильме» содержатся название, жанр, продолжительность и возрастное ограничение;
- в «Зале» одновременно указаны вместимость, название и этаж;
- атрибут «Билет» объединяет номер, ряд, место, категорию и цену;
- атрибут «Зритель» содержит одновременно ФИО, возраст и социальный статус.

Для приведения отношения к первой нормальной форме необходимо каждый составной атрибут разложить на отдельные атомарные. Выделим ключи как отдельные:

- сведения о фильмах – Название фильма, Жанр, Длительность, Возрастное ограничение;
- данные о залах – Название зала, Вместимость, Категория зала, Этаж;
- информацию о сотрудниках – ФИО сотрудника, Должность, Почта сотрудника, Номер сотрудника, Смена;
- информацию о билетах – Идентификатор, Ряд, Место, Цена;
- данные зрителя – ФИО зрителя, возраст.

Первая нормальная форма представлена в табл. 2.

Таблица 2

Первая нормальная форма

1. ID сеанса	2. Дата сеанса	3. Время начала	4. Время окончания	5. Название фильма	6. Жанр	7. Длительность	8. Возрастное ограничение	9. Название зала
100001	24.05.2025	13:45	15:45	Дюна: Часть вторая	фантастика	155	16+	Зал Стандарт «Солнечный»
100002	25.05.2025	16:00	18:00	Джокер: Безумие на двоих	драма	138	18+	Зал Премиум «Лазурный»
100003	25.05.2025	10:30	12:30	Холодное сердце 2	мультфильм	103	6+	Зал IMAX «Гранд»
100004	26.05.2025	14:15	17:15	Миссия невыполнима 7	боевик	164	12+	Зал 3D «Вдохновение»
100005	27.05.2025	18:00	22:00	Барби	комедия	114	12+	Зал Стандарт «Элегант»

Продолжение табл. 2

10. Вместимость	11. Категория зала	12. Этаж	13. Ряд	14. Место	15. Тариф	16. Фамилия зрителя	17. Имя зрителя	18. Отчество зрителя
200	Стандартный	2	21	21	Льготный	Иванов	Алексей	Сергеевич
150	Премиум	3	13	7	Семейный	Петрова	Марина	Викторовна
100	IMAX	4	16	18	Комфорт	Сидоров	Дмитрий	Павлович
50	3D	1	19	25	Стандарт	Кузнецов	Ольга	Андреевна
140	VIP	3	22	10	Премиум	Васильев	Сергей	Николаевич

Продолжение табл. 2

19. Фамилия кассира	20. Имя кассира	21. Отчество Кассира	22. Смена кассира	23. Номер кассира	24. Почта кассира	25. Фамилия администратора	26. Имя администратора
Иванов	Иван	Иванович	Дневная	+79011234567	ivanov@mail.ru	Смирнов	Сергей
Петрова	Анна	Сергеевна	Вечерняя	+79022345678	petrova@mail.ru	Федорова	Елена
Сидорова	Алексей	Павлович	Утренняя	+79033456789	sidorov@mail.ru	Морозов	Илья
Кузнецова	Мария	Викторовна	Ночная	+79044567890	kuznetsova@mail.ru	Ковалева	Наталья
Васильев	Дмитрий	Андреевич	Утренняя	+79055678901	vasiliev@mail.ru	Григорьев	Олег

Продолжение табл. 2

27. Отчество администратора	28. Смена администратора	29. Зона ответственности	30. Номер администратора	31. Почта администратора	32. Фамилия сотрудника техперсонала	33. Имя сотрудника техперсонала	34. Отчество сотрудника техперсонала
Николаевич	Дневная	Фойе	+79532745314	smirnov@mail.ru	Иванов	Сергей	Петрович
Михайловна	Вечерняя	Залы	+79532745314	fedorova@mail.ru	Кузнецов	Алексей	Дмитриевич
Алексеевна	Утренняя	Кассовая зона	+79543222845	morozov@mail.ru	Смирнова	Анна	Викторовна
Ковалева	Ночная	Техническая зона	+79482759402	kovaleva@mail.ru	Поляков	Дмитрий	Олегович
Григорьев	Утренняя	Общая координация	+79482359431	grigoriev@mail.ru	Мельникова	Елена	Сергеевна

Продолжение табл. 2

35. Номер сотрудника техперсонала	36. Смена сотрудника техперсонала	37. Почта сотрудника техперсонала	38. Специализация сотрудника
+79812745276	Дневная	ivanov.tech@cinema.ru	Звукооператор
+79992745123	Вечерняя	kuznetsov.tech@cinema.ru	Светооператор
+79114328459	Утренняя	smirnova.tech@cinema.ru	Киномеханик
+79213456789	Ночная	polyakov.tech@cinema.ru	Электрик
+79812345677	Утренняя	melnikova.tech@cinema.ru	Специалист по оборудованию зала

Приведение к 1НФ делает структуру отношения более строгой и однородной, но усложняет её за счёт увеличения числа атрибутов. При этом дублирование данных всё ещё возможно - 1НФ не устраняет избыточность. Именно для сокращения избыточности и устранения частичных зависимостей применяется вторая нормальная форма (2НФ).

3.1.3. Вторая нормальная форма

Отношение находится во второй нормальной форме (2НФ), если оно удовлетворяет требованиям первой нормальной формы и каждый неключевой атрибут зависит от всего первичного ключа, а не от его части [5].

В представленной структуре базы данных в 1НФ все сведения о фильмах, залах, билетах, зрителях и сотрудниках хранятся в одной большой таблице. Это приводит к избыточности и дублированию информации: например, данные о фильме повторяются для каждого сеанса с этим фильмом, а характеристики зала дублируются для каждой записи, связанной с этим залом. При этом наблюдаются частичные функциональные зависимости: атрибуты, описывающие фильм (название, жанр, длительность, возрастное ограничение), зависят только от конкретного фильма, а не от всей записи о сеансе; характеристики зала (название, вместимость, категория, этаж) зависят только от зала, а не от сеанса в целом; данные о зрителе или сотруднике повторяются независимо от контекста их использования. Такое построение нарушает требования второй нормальной формы и усложняет обновление данных.

Таблица «Билет» связана с таблицами «Сеанс», «Зритель» и «Кассир» через соответствующие идентификаторы (ID сеанса, ID зрителя, ID кассира), что позволяет однозначно определить сеанс показа, покупателя и сотрудника, оформившего продажу.

Атрибуты из исходной таблицы перераспределены и размещены в следующих таблицах (табл. 3-9):

- Сеанс (ID сеанса, ID фильма, ID зала, ID сотрудника, ID администратора, Дата сеанса, Время начала, Время окончания), связана с таблицей Фильм атрибутом ID фильма, Зал – ID зала, Техперсонал – ID сотрудника, Администратор – ID администратора;
- Фильм (ID фильма, Название, Жанр, Длительность, Возрастное ограничение), связана с таблицей Сеанс атрибутом ID фильма;
- Зал (ID зала, Название зала, Вместимость, Категория, Этаж), связана с таблицей Сеанс атрибутом ID зала;
- Билет (ID билета, Ряд, Место, Тариф, Цена, ID сеанса, ID зрителя, ID кассира), связана с таблицей Сеанс атрибутом ID сеанса, Зритель – ID зрителя, Кассир – ID кассира;
- Зритель (ID зрителя, Фамилия, Имя, Отчество, Возраст), связана с таблицей Билет атрибутом ID билета;
- Кассир (ID кассира, Фамилия, Имя, Отчество, Смена, Номер, Почта), связана с таблицей Билет атрибутом ID кассира;
- Техперсонал (ID сотрудника, Фамилия, Имя, Отчество, Смена, Номер,

Почта, Специализация), связана с таблицей Сеанс атрибутом ID сотрудника;

- Администратор (ID администратора, Фамилия, Имя, Отчество, Смена, Номер, Почта, Зона ответственности), связана с таблицей Сеанс атрибутом ID администратора.

Таблица 3

Сеанс

ID сеанса	ID фильма	ID зала	ID сотрудника	ID администратора	Дата сеанса	Время начала	Время окончания
1500000	1500000	500001	1400002	200001	24.05.2025	10:00	13:00
1500001	1500003	500004	1400001	200004	12.06.2025	11:00	12:00
1500002	1500004	500000	1400000	200004	13.05.2025	14:00	16:00
1500003	1500003	500002	1400003	200002	14.05.2025	22:00	00:00
1500004	1500001	500003	1400004	200000	14.05.2025	01:00	03:00

Таблица 4

Фильм

ID Фильма	Название фильма	Жанр	Длительность	Возрастное ограничение
1500000	Дюна: Часть вторая	Фантастика	155	16+
1500001	Джокер: Безумие на двоих	Драма	138	18+
1500002	Холодное сердце 2	Мультфильм	103	6+
1500003	Миссия невыполнима 7	Боевик	164	12+
1500004	Барби	Комедия	114	12+

Таблица 5

Зал

ID зала	Категория зала	Вместимость	Названия зала	Этаж
500000	Стандартный	200	Зал Стандарт «Солнечный»	2
500001	Премиум	150	Зал Премиум «Лазурный»	3
500002	IMAX	100	Зал IMAX «Гранд»	4
500003	VIP	50	Зал 3D «Вдохновение»	1
500004	3D	140	Зал Стандарт «Элегант»	3

Таблица 6

Билет

ID билета	Ряд	Место	Цена	Тариф	ID сеанса	ID зрителя	ID кассира
300000	2	21	1000	Стандарт	1000009	700002	800000
300001	7	45	4500	Льготный	1000019	700003	800001
300002	13	2	2300	Комфорт	1000023	700005	800000
300003	11	2	3210	Семейный	1000342	700004	800003
300004	10	1	3200	Премиум	1000123	700001	800001

Таблица 7

Зритель

ID зрителя	Фамилия зрителя	Имя зрителя	Отчество зрителя	Возраст
700000	Иванов	Алексей	Сергеевич	28
700001	Петров	Марина	Викторовна	34
700002	Сидорова	Дмитрий	Павлович	41
700003	Кузнецова	Ольга	Андреевна	25
700004	Васильев	Сергей	Николаевич	12

Таблица 8

Кассир

ID кассира	Фамилия кассира	Имя кассира	Отчество кассира	Номер	Почта	Смена
800000	Иванов	Иван	Иванович	+79011234567	ivanov@mail.ru	Утренняя
800001	Петрова	Анна	Сергеевна	+79022345678	petrova@mail.ru	Дневная
800002	Сидоров	Алексей	Павлович	+79033456789	sidorov@mail.ru	Вечерняя
800003	Кузнецова	Мария	Викторовна	+79044567890	kuznetsova@mail.ru	Ночная
800004	Васильев	Дмитрий	Андреевич	+79055678901	vasiliev@mail.ru	Вечерняя

Таблица 9

Администратор

ID администратора	Фамилия администратора	Имя администратора	Отчество администратора	Телефон	Почта	Зона ответственности	Смена
200000	Смирнов	Даниил	Иванович	+79066789012	smirnov@mail.ru	Фойе	Утренняя
200001	Федорова	Елена	Владимирович	+79077890123	fedorova@mail.ru	Залы	Дневная
200002	Морозов	Никита	Анатолевич	+79088901234	morozov@mail.ru	Кассовая зона	Вечерняя
200003	Ковалева	Наталья	Сергеевна	+79099012345	kovaleva@mail.ru	Техническая зона	Ночная
200004	Григорьев	Олег	Владимирович	+79100123456	grigoriev@mail.ru	Общая координация	Вечерняя

Таблица 10

Техперсонал

ID сотрудника	Фамилия сотрудника	Имя сотрудника	Отчество сотрудника	Телефон	Почта	Специализация	Смена
1400000	Иванов	Сергей	Петрович	+79001112233	ivanov.tech@cinema.ru	Звукооператор	Утренняя
1400001	Смирнов	Алексей	Дмитриевич	+79012223344	kuznetsov.tech@cinema.ru	Светооператор	Дневная
1400002	Кузнецова	Анна	Викторовна	+79023334455	smirnova.tech@cinema.ru	Киномеханик	Вечерняя
1400003	Поляков	Дмитрий	Олегович	+79034445566	polyakov.tech@cinema.ru	Электрик	Ночная
1400004	Мельникова	Елена	Сергеевна	+79045556677	melnikova.tech@cinema.ru	Специалист по оборудованию зала	Вечерняя

Во второй нормальной форме остаются транзитивные функциональные зависимости, которые вызывают аномалии вставки, удаления и обновления данных. Для их устранения необходимо привести структуру к третьей нормальной форме.

3.1.4. Третья нормальная форма

Отношение находится в третьей нормальной форме (3НФ), если оно удовлетворяет требованиям второй нормальной формы и каждый неключевой атрибут нетранзитивно зависит от первичного ключа, то есть отсутствуют зависимости неключевых атрибутов от других неключевых атрибутов [5].

Исходная структура базы данных содержала транзитивные зависимости, что нарушало требования третьей нормальной формы. В сущности «Зал» категория зала хранилась непосредственно, что приводило к избыточности и сложностям при обновлении данных. Похожая проблема наблюдалась в сущности «Билет», где данные о тарифах зависели от других атрибутов этой же сущности, что усложняло обновление и увеличивало вероятность ошибок при изменении данных.

Для устранения транзитивных зависимостей была проведена нормализация структуры базы данных с выделением отдельных сущностей. В частности, была создана сущность «Жанр фильма», связанная с сущностью «Фильм». Также была добавлена сущность «Возрастные ограничения», что позволило точно и удобно хранить информацию об ограничениях без повторений и ошибок. В сущности «Фильм» теперь хранятся ссылки на соответствующие жанры и возрастные ограничения через внешние ключи, что устранило избыточность данных.

Сущность «Зал» была декомпозирована для выделения информации о категориях залов. В новой сущности «Категория зала» хранятся идентификаторы и наименования категорий, а в сущности «Зал» теперь содержится только ссылка на нужную категорию через внешний ключ.

Сущность «Билет» была разделена: теперь она хранит данные о самом билете, ряде, месте и цене, а информация о тарифах вынесена в отдельную сущность «Тариф», которая содержит идентификаторы и типы тарифов. Это решение устранило транзитивные зависимости и повысило гибкость базы данных.

Сущности, связанные с персоналом, были структурированы следующим образом. Сущность «Кассиры» теперь содержит атрибут с идентификатором сущности «Смены», где хранится информация о типах смен. Сущность «Администратор» содержит атрибут с идентификатором сущности «Зоны ответственности», что позволило чётко определить области работы каждого администратора. Сущность «Технический персонал» была дополнена атрибутом с идентификатором сущности «Специализация», в которой содержатся типы специализаций технических сотрудников.

Сущность «Сеанс» была обновлена и теперь содержит ссылки на ключевые сущности, такие как «Фильм» и «Зал». Это обеспечило целостность данных и упростило их обработку при формировании расписания показов.

Таблицы базы данных в третьей нормальной форме находятся в табл. 10-24.

Таблица 10

Сеанс

ID сеанса	ID фильма	ID зала	ID сотрудника	Дата сеанса	Время начала	Время окончания
1500000	1500000	500001	1400002	24.05.2025	10:00	13:00
1500001	1500003	500004	1400001	12.06.2025	11:00	12:00
1500002	1500004	500000	1400000	13.05.2025	14:00	16:00
1500003	1500003	500002	1400003	14.05.2025	22:00	00:00
1500004	1500001	500003	1400004	14.05.2025	01:00	03:00

Таблица 11

Жанр

ID жанра	Жанр
160000	Комедия
160001	Драма
160002	Боевик
160003	Фантастика
160004	Ужасы

Таблица 12

Возрастное ограничение

ID ограничения	Возрастное ограничение
400000	0+
400001	6+
400002	12+
400003	16+
400004	18+

Таблица 13

Фильм

ID фильма	ID жанра	ID ограничения	Название	Длительность
1500000	160000	400000	Дюна: Часть вторая	150
1500001	160001	400001	Джокер: Безумие на двоих	189
1500002	160002	400002	Холодное сердце 2	103
1500003	160003	400003	Миссия невыполнима 7	152
1500004	160001	400004	Барби	135

Таблица 14

Категория зала

ID категории зала	Категория зала
1	Стандарт
2	Премиум
3	IMAX
4	VIP
5	3D

Таблица 15

Зал

ID зала	ID категории зала	Вместимость	Название зала	Этаж
500000	900000	200	Зал Стандарт «Солнечный»	1
500001	900001	150	Зал Премиум «Лазурный»	2
500002	900004	100	Зал IMAX «Гранд»	4
500003	900003	50	Зал 3D «Вдохновение»	3
500004	900002	140	Зал Стандарт «Элегант»	1

Таблица 16

Тариф

ID тарифа	Тариф
1300000	Стандарт
1300001	Льготный
1300002	Комфорт
1300003	Семейный
1300004	Премиум

Таблица 17

Билет

ID билета	Ряд	Место	Цена	ID зала	ID тарифа	ID зрителя	ID кассира	ID сеанса
300000	2	21	1000	500002	1300002	700002	800001	100001
300001	7	24	4500	500002	1300003	700003	800002	100000
300002	13	37	2300	500004	1300000	700004	800001	100002
300003	11	11	3210	500003	1300003	700001	800004	100003
300004	10	41	3200	500003	1300000	700000	800003	100004

Таблица 18

Зритель

ID зрителя	Фамилия зрителя	Имя зрителя	Отчество зрителя	Возраст
700000	Иванов	Алексей	Сергеевич	28
700001	Петров	Марина	Викторовна	34
700002	Сидорова	Дмитрий	Павлович	41
700003	Кузнецова	Ольга	Андреевна	25
700004	Васильев	Сергей	Николаевич	12

Таблица 19

Смена

ID смены	Смена
1100000	Утренняя
1100001	Дневная
1100002	Вечерняя
1100003	Ночная

Таблица 20

Кассир

ID кассира	Фамилия кассира	Имя кассира	Отчество кассира	Номер	Почта	ID Смены
800000	Иванов	Иван	Иванович	+79011234567	ivanov@mail.ru	1100000
800001	Петрова	Анна	Сергеевна	+79022345678	petrova@mail.ru	1100001
800002	Сидоров	Алексей	Павлович	+79033456789	sidorov@mail.ru	1100002
800003	Кузнецова	Мария	Викторовна	+79044567890	kuznetsova@mail.ru	1100003
800004	Васильев	Дмитрий	Андреевич	+79055678901	vasiliev@mail.ru	1100000

Таблица 21

Зона ответственности

ID зоны ответственности	Зона ответственности
600000	Фойе
600001	Залы
600002	Техническая зона
600003	Общая координация

Таблица 22

Администратор

ID администратора	Фамилия	Имя	Отчество	Номер	Почта	ID зоны ответственности	ID смены
200000	Смирнов	Даниил	Иванович	+79066789012	smirnov@mail.ru	600000	1100000
200001	Федорова	Елена	Петрович	+79077890123	fedorova@mail.ru	600001	1100001
200002	Морозов	Виктор	Анатольевич	+79088901234	morozov@mail.ru	600002	1100002
200003	Ковалева	Анна	Сергеевна	+79099012345	kovaleva@mail.ru	600003	1100003
200004	Григорьев	Олег	Петрович	+79100123456	grigoriev@mail.ru	600001	1100000

Специализация

ID специализации	Специализация
1200000	Киномеханик
1200001	Звукооператор
1200002	Светооператор
1200003	Электрик

Таблица 24

Техперсонал

ID сот-рудника	Фамилия	Имя	Отчество	Номер	Почта	ID специ-ализации	ID смены
1400000	Иванов	Даниил	Иванович	+794327 56316	ivanov.tech@c inema.ru	1200002	1100000
1400001	Петров	Валерий	Петрович	+794837 15476	petrov.tech@ci nema.ru	1200001	1100001
1400002	Кузьмин	Никита	Анатольевич	+792817 46575	kuzmin.tech@ cinema.ru	1200003	1100002
1400003	Поляков	Дмитрий	Олегович	+790344 45566	polyakov.tech @cinema.ru	1200002	1100003
1400004	Мулина	Елена	Сергеевна	+790455 56677	melnikova.tech @cinema.ru	1200004	1100000

Таблица приведена к третьей нормальной форме, поскольку достижение 3НФ в большинстве практических случаев обеспечивает оптимальный баланс между устранением избыточности данных и сохранением удобства работы с базой. Третья нормальная форма позволяет избавиться от большинства аномалий обновления и избыточных зависимостей, что значительно повышает целостность и эффективность хранения информации. При этом дальнейшее усложнение структуры за счёт перехода к более высоким нормальным формам часто не приносит существенных преимуществ и может привести к излишнему усложнению схемы данных и снижению производительности. Поэтому этап даталогического проектирования обычно завершается после достижения третьей нормальной формы, после чего начинается следующий этап - логическое проектирование, где создаётся детальная структура базы данных с учётом выбранной модели и требований системы.

3.2. Логическая модель

После приведения структуры базы данных к третьей нормальной форме формируется логическая модель, которая отображает основные элементы предметной области и связи между ними. В логической модели выделяются сущности - объекты, представляющие важные элементы системы, их атрибуты - характеристики этих объектов, а также связи между сущностями. Важной частью модели являются ключи: первичные - для уникальной идентификации записей, и

внешние - для установления связей между таблицами [6].

Для создания и визуализации логической модели применяется программа ERwin Data Modeler. Это CASE-средство позволяет проектировать, документировать и сопровождать структуры баз данных, обеспечивая наглядное представление всех компонентов системы и их взаимодействий [7].

На рис. 1 представлена ER-диаграмма логической модели кинотеатра, построенная с использованием ERwin Data Modeler.

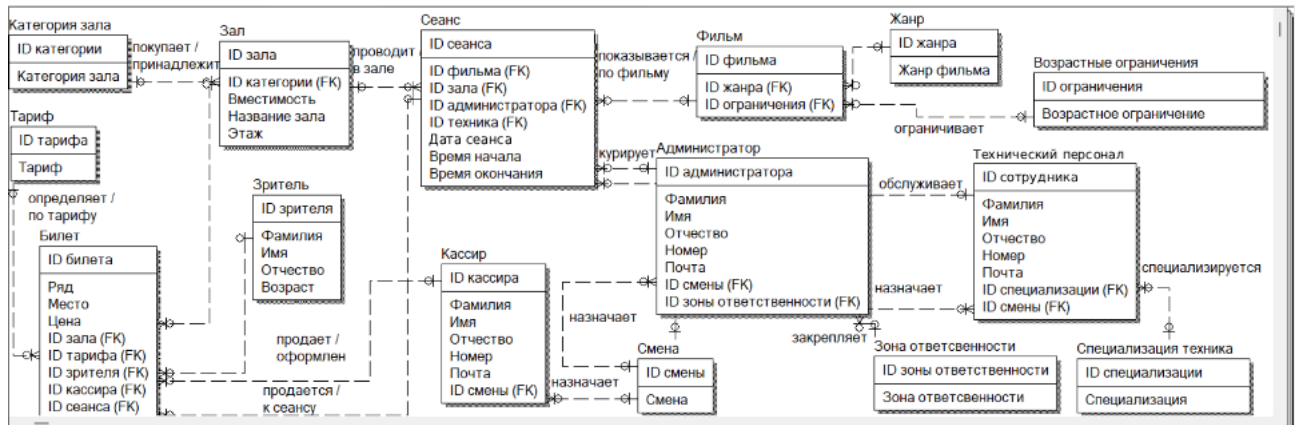


Рис. 1. Логическая модель

3.3. Физическая модель

Физическое проектирование базы данных - процесс подготовки описания реализации базы данных на вторичных запоминающих устройствах; на этом этапе рассматриваются основные отношения, организация файлов и индексов, предназначенных для обеспечения эффективного доступа к данным, а также все связанные с этим ограничения целостности и средства защиты.

Как правило, основной целью физического проектирования базы данных является описание способа физической реализации логического проекта базы данных [8].

Для выполнения физического проектирования базы данных нужно выбрать конкретную СУБД, так как этот этап проектирования связан с ней.

Перечень реляционных СУБД, которые могут использоваться в разрабатываемой программе:

- MySQL — это СУБД, распространяемая как свободное программное обеспечение и используемая для упрощения работы с базами данных. Она включает в себя библиотеку внутреннего сервера, с помощью которой можно использовать MySQL в отдельных программах [8]
- MariaDB - это свободная реляционная система управления базами данных, созданная на основе MySQL. MariaDB была разработана сообществом под руководством создателей MySQL после того, как последняя была приобретена компанией Oracle. Система полностью совместима с MySQL и предлагает дополнительные возможности: улучшенную производительность, расширенный набор движков хранения данных (включая Aria, ColumnStore для аналитики), встроенную поддержку JSON, улучшенную репликацию и отказоустойчивость.

MariaDB активно развивается, обеспечивает высокую скорость обработки запросов и подходит как для небольших приложений, так и для крупных корпоративных систем. Распространяется под лицензией GPL и поддерживает кроссплатформенность [9];

- PostgreSQL — это объектно-реляционная система управления базами данных. Она написана на языке C и распространяется свободно. СУБД PostgreSQL работает со стандартным языком запросов SQL, а также с его дополненной версией - PL/pgSQL. Особенности PostgreSQL: масштабируемость, кроссплатформенность, безопасность, поддержка различных типов данных, расширяемость, открытый код, активное сообщество, поддержка NoSQL, требовательность к ресурсам, сложная настройка [10];

- Microsoft Access представляет собой удобную и функциональную систему управления базами данных, предлагающую широкий набор инструментов для создания, определения и обработки данных. Благодаря интуитивно понятному интерфейсу и встроенным возможностям, Access обеспечивает простоту разработки и управления базами данных, что способствует эффективной автоматизации бизнес-процессов и улучшению работы бизнес-приложений. [11].

В рамках курсовой работы выбрана СУБД Microsoft Access как наиболее подходящая по критериям скорости разработки, минимальных затрат и соответствия объему задачи.

Физическая структура базы представлена схемами таблиц (табл. 25–39) и визуализирована на (рис. 2).

Таблица 25

Схема данных таблицы «Фильм»

Атрибут	Тип данных
ID фильма	Счетчик, Первичный ключ
ID жанра	Текстовый (20)
ID ограничения	Числовой (Длинное целое), Внешний ключ
Название	Числовой (Длинное целое), Внешний ключ
Длительность	Числовой

Таблица 26

Схема данных таблицы «Жанр»

Атрибут	Тип данных
ID жанра	Счетчик, Первичный ключ
Жанр	Текстовый (50)

Таблица 27

Схема данных таблицы «Возрастные ограничения»

Атрибут	Тип данных
ID ограничения	Счетчик, Первичный ключ
Ограничение	Текстовый (50)

Таблица 28

Схема данных таблицы «Тариф»

Атрибут	Тип данных
ID тарифа	Счетчик, Первичный ключ
Тариф	Текстовый (50)

Таблица 29

Схема данных таблицы «Билет»

Атрибут	Тип данных
ID билета	Счетчик, Первичный ключ
Ряд	Числовой
Место	Числовой
Цена	Денежный
ID зала	Числовой (Длинное целое), Внешний ключ
ID тарифа	Числовой (Длинное целое), Внешний ключ
ID зрителя	Числовой (Длинное целое), Внешний ключ
ID кассира	Числовой (Длинное целое), Внешний ключ
ID сенса	Числовой (Длинное целое), Внешний ключ

Таблица 30

Схема данных таблицы «Сеанс»

Атрибут	Тип данных
ID с еанса	Счетчик, Первичный ключ
ID фильма	Числовой (Длинное целое), Внешний ключ
ID зала	Числовой (Длинное целое), Внешний ключ
ID администратора	Числовой (Длинное целое), Внешний ключ
ID сотрулника	Числовой (Длинное целое), Внешний ключ
Время начала	Дата/время
Время окончания	Дата/время

Таблица 31

Схема данных таблицы «Кассир»

Атрибут	Тип данных
ID к ассира	Счетчик, Первичный ключ
Фамилия	Текстовый (50)
Имя	Текстовый (50)
Отчество	Текстовый (50)
Номер	Текстовый (11)
Почта	Текстовый (50)
ID смены	Числовой (Длинное целое)

Таблица 32

Схема данных таблицы «Зона ответственности»

Атрибут	Тип данных
ID зоны ответственности	Счетчик, Первичный ключ
Зона ответственности	Текстовый (50)

Таблица 33

Схема данных таблицы «Администратор»

Атрибут	Тип данных
ID администратора	Счетчик, Первичный ключ
Фамилия	Текстовый (50)
Имя	Текстовый (50)
Отчество	Текстовый (50)
Номер	Текстовый (11)
Почта	Текстовый (50)
ID смены	Числовой (Длинное целое), Внешний ключ
ID зоны ответственности	Числовой (Длинное целое), Внешний ключ

Таблица 34

Схема данных таблицы «Специализация»

Атрибут	Тип данных
ID специализации	Счетчик, Первичный ключ
Специализация	Числовой (Длинное целое), Внешний ключ

Таблица 35

Схема данных таблицы «Технический персонал»

Атрибут	Тип данных
ID сотрудника	Счетчик, Первичный ключ
Фамилия	Текстовый (50)
Имя	Текстовый (50)
Отчество	Текстовый (50)
Номер	Текстовый (50)
Почта	Текстовый (50)
ID специализация	Числовой (Длинное целое), Внешний ключ
ID смены	Числовой (Длинное целое), Внешний ключ

Таблица 36

Схема данных таблицы «Категория зала»

Атрибут	Тип данных
ID категории	Счетчик, Первичный ключ
Категория зала	Текстовый (50)

Таблица 37

Схема данных таблицы «Зал»

Атрибут	Тип данных
ID зала	Счетчик, Первичный ключ
ID категории	Числовой (Длинное целое), Внешний ключ
Вместимость	Числовой (Длинное целое), Внешний ключ
Название зала	Текстовый (50)
Этаж	Числовой

Схема данных таблицы «Зритель»

Атрибут	Тип данных
ID зрителя	Счетчик, Первичный ключ
Фамилия	Текстовый (50)
Имя	Текстовый (50)
Отчество	Текстовый (50)
Возраст	Числовой

Таблица 39

Схема данных таблицы «Смена»

Атрибут	Тип данных
ID смены	Счетчик, Первичный ключ
Смена	Текстовый (50)

На рис. 2 представлена физическая модель данных.

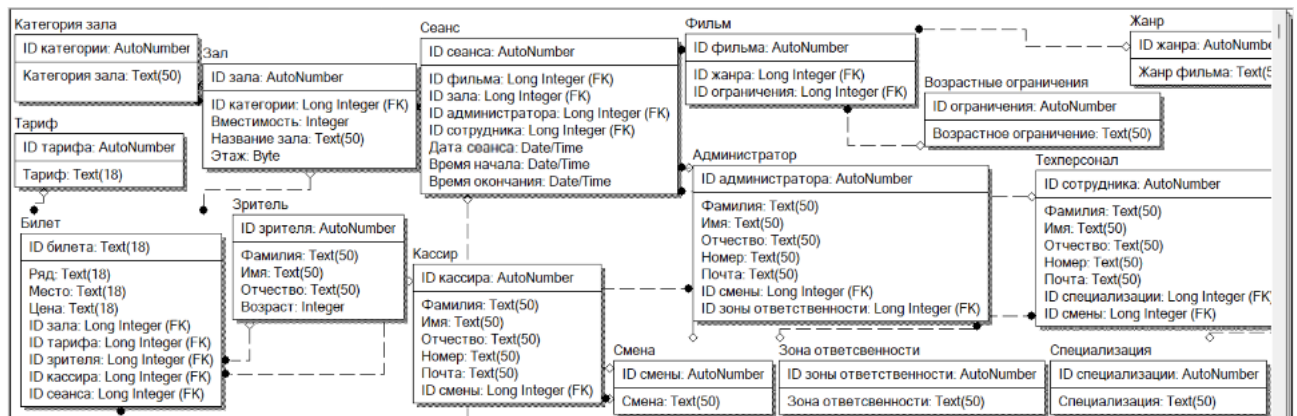


Рис. 2. Физическая модель

После создания физического представления логической модели в программе ERwin необходимо сгенерировать базу данных СУБД Microsoft Access. Для этого нужно в режиме представления физической области нажать на вкладку «Database», в меню инструментов нажать на «Choose database...» (рис. 3), выбрать СУБД Access. Далее модель подключается к файлу базы данных через выбор опции «Database connection...» во вкладке «Database», после этого нужно указать расположение файла и нажать кнопку «Connect» (рис. 4).

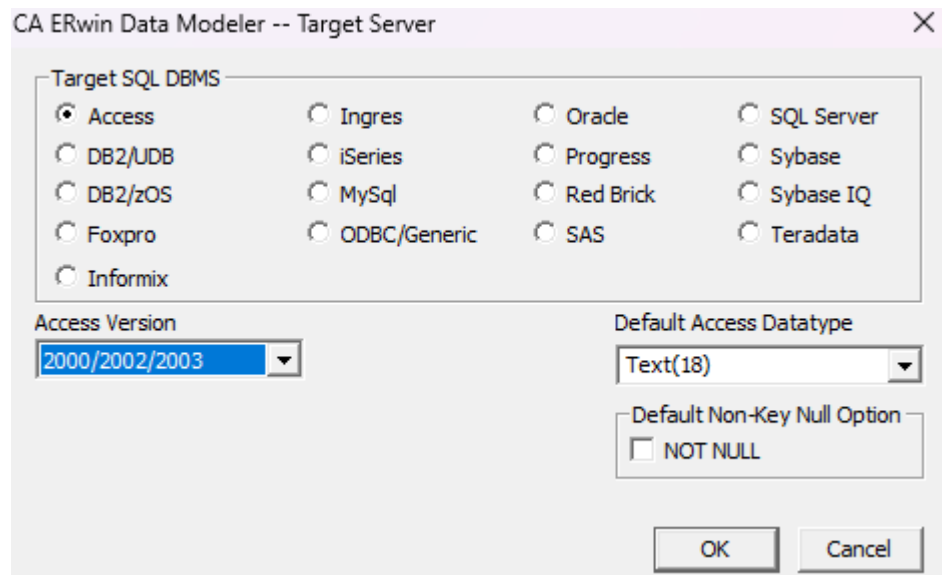


Рис. 3. Выбор СУБД

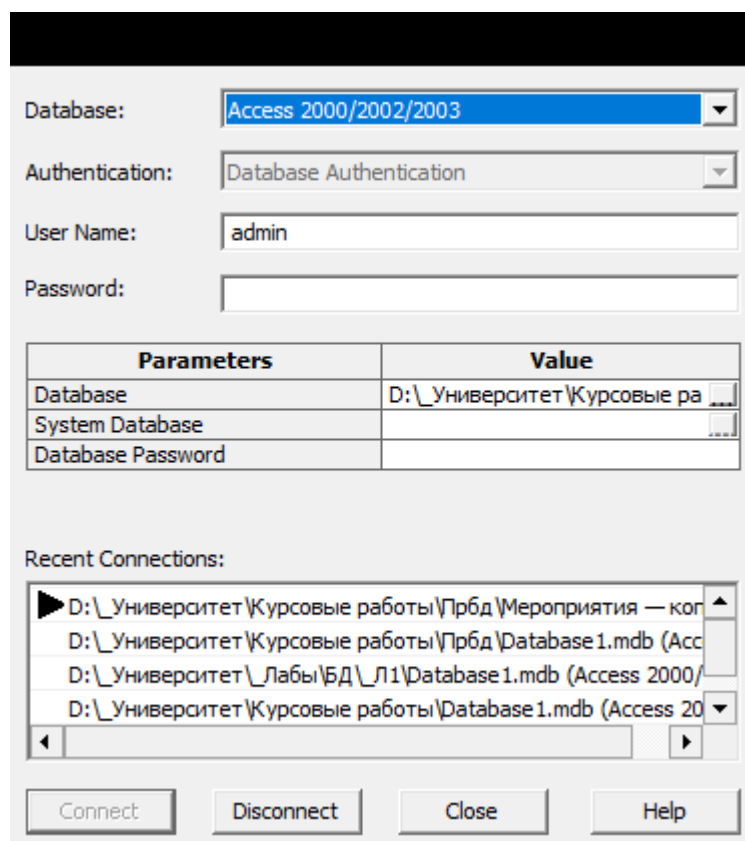


Рис. 4. Подключение к БД

После подключения к файлу нужно во вкладке «Tools» выбрать опцию «Forward Engineer», после – «Shema Generation...». В появившемся окне можно выполнить настройку схемы и для её генерации нажать на кнопку «Generate...» (рис. 5), после нажатия на которую будет показан процесс выполнения запросов (рис. 6).

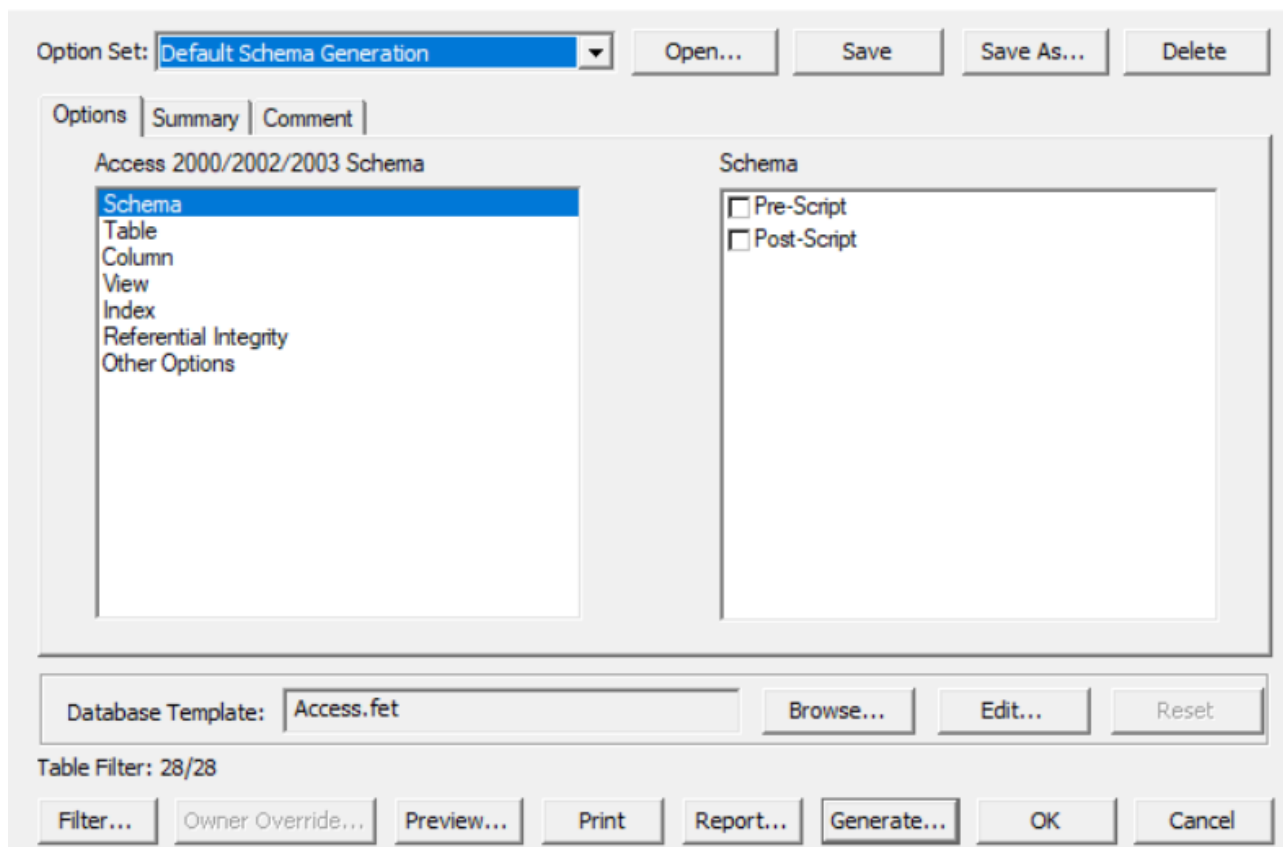


Рис. 5. Окно настройки схемы

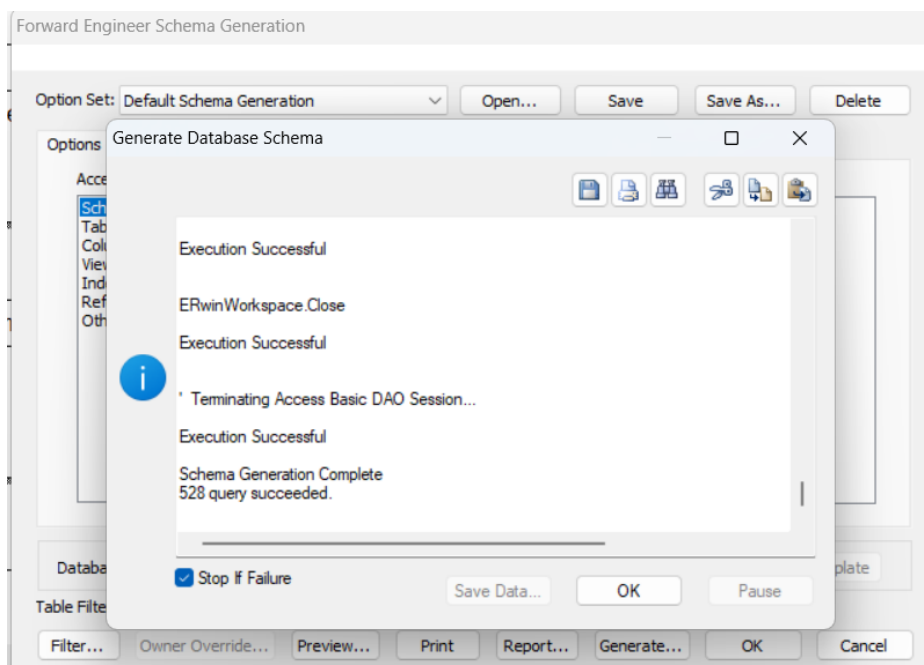


Рис. 6. Окно выполнения запросов

Результатом выполнения генерирования являются созданные таблицы и связи между ними, которые хранятся в файле Access «Cinema.mdb» (рис. 7-8).

Все объекты Access

Поиск...

Таблицы

- Администратор
- Билет
- Возрастные ограничения
- Жанр
- Зал
- Зона ответственности
- Зритель
- Кассир
- Категория зала
- Сеанс
- Смена
- Специализация
- Тариф
- Технический персонал
- Фильм

Зритель

Имя поля	Тип данных	Описание (необязатель)
ID зрителя	Счетчик	
Фамилия	Короткий текст	
Имя	Короткий текст	
Отчество	Короткий текст	
Возраст	Числовой	

Свойства поля

Общие	Подстановка
Размер поля	Длинное целое
Новые значения	Последовательные
Формат поля	

Рис. 7. Результат генерации

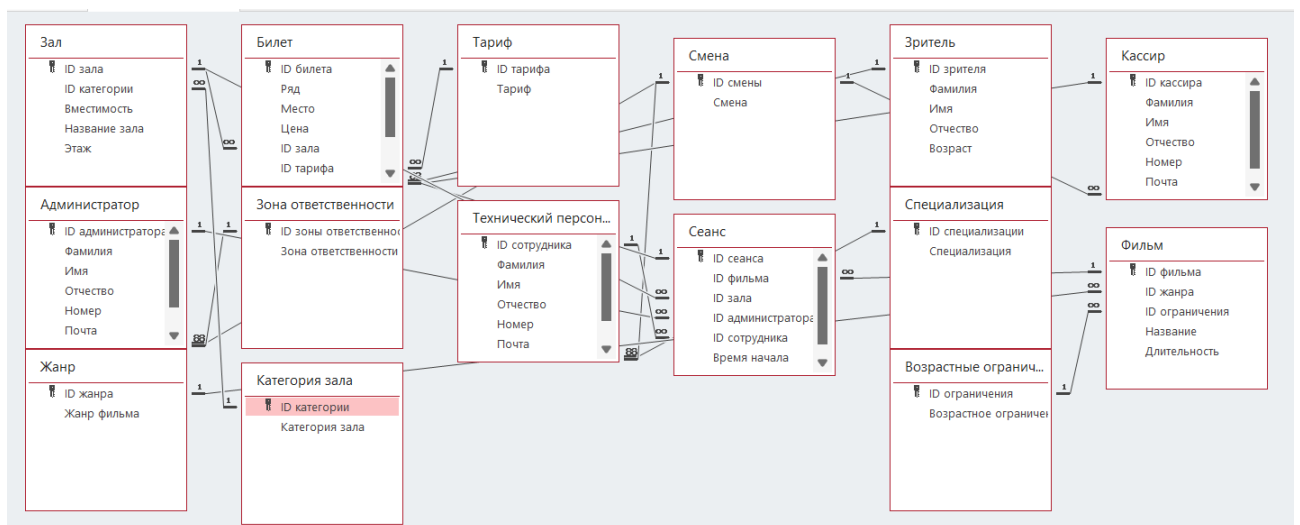


Рис. 8. Схема данных в Access

По завершении этапа даталогического проектирования осуществляется переход к следующей стадии разработки – созданию программного приложения информационной системы.

4. Описание программного обеспечения

Для разработки приложения был выбран язык программирования C# в сочетании с технологией Windows Forms. Такой выбор объясняется тем, что язык предоставляет встроенные средства взаимодействия с базами данных через пространство имён System.Data.OleDb, которое реализует технологию OLE DB. Это решение оказалось наиболее подходящим для десктопного приложения, ориентированного на работу в среде Windows, где требуется подключение к базе данных Access, хранящейся в виде отдельного файла на локальном диске пользователя [13].

Работа над проектом будет вестись в среде Visual Studio 2022. Эта среда предоставляет все необходимые инструменты для разработки, включая визуальный редактор форм, позволяющий наглядно располагать элементы интерфейса - кнопки, поля ввода, таблицы. Благодаря встроенным средствам генерации кода и удобной системе отладки, процесс создания окон, настройки событий и написания логики приложения может быть организован последовательно и удобно [13].

На этапе программирования особое внимание уделялось организации связи между интерфейсом и базой данных. Для этого использовались компоненты OleDbConnection, OleDbCommand и OleDbDataReader. Сначала устанавливалось соединение с файлом базы данных Access, после чего выполнялся SQL-запрос, и результаты считывались построчно. Полученные данные загружались в интерфейс приложения и отображались с помощью элемента DataGridView. Для упрощения привязки данных использовался компонент BindingSource, который позволял связать элементы управления с источником данных и выполнять базовые операции фильтрации и обновления [13].

В процессе работы были разработаны оконные интерфейсы, отражающие предполагаемую структуру и функциональность будущего приложения. Интерфейс, предназначенный для выполнения пользовательских запросов и отображения таблиц на экране, представлен в приложении информационной системы кинотеатра (рис. 8). Отдельное окно реализовано для добавления новой записи, где пользователь может вводить необходимые данные (рис. 9). Также разработан интерфейс редактирования, предназначенный для изменения уже существующих записей в системе (рис. 10). Программный код, реализующий указанные интерфейсы, приведён в листинге (П2), а описание работы с интерфейсами включено в руководство пользователя (П3).

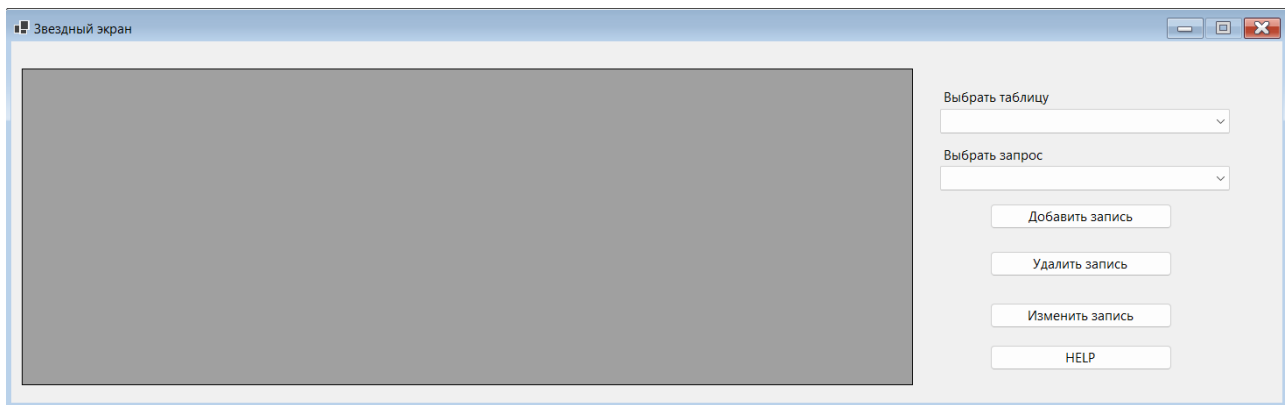


Рис. 8. Окно выполнения запросов и отображения таблиц

Рис. 9. Окно добавления новой записи

Рис. 10. Окно редактирования существующей записи

В ходе выполнения данной курсовой работы была спроектирована база данных на тему кинотеатра, отражающая ключевые сущности и процессы, характерные для данной предметной области. На основе этой базы было разработано программное решение с графическим интерфейсом, позволяющее пользователю выполнять операции добавления, редактирования и удаления записей, а также осуществлять выборку данных с помощью встроенных запросов.

Реализация данной системы позволяет значительно сократить время на обработку информации о сеансах и продаже билетов, снизить количество ошибок, связанных с ручным вводом данных, а также повысить прозрачность и контролируемость всех этапов работы кинотеатра. Благодаря автоматизации ключевых процессов, программа представляет собой эффективный инструмент для повышения качества обслуживания посетителей и оптимизации работы персонала, что делает её ценной для руководства и сотрудников кинотеатра.

1. LiveJournal <https://kak-eto-sdelano.livejournal.com/13170.html> Дата доступа: 22.09.2025
2. Сидорова, Н. П. Базы данных: практикум по проектированию реляционных баз данных : учебное пособие : [16+] / Н. П. Сидорова ; Технологический университет, Институт техники и цифровых технологий, Факультет инфокоммуникационных систем и технологий. – Москва ; Берлин : Директ-Медиа, 2020. – 93 с, Дата доступа: 22.09.2025.
3. Лысенко К. Д. Инфологическое и даталогическое проектирование информационной системы спортивной школы с помощью ERwin / К. Д. Лысенко // Вестник науки. – 2023. Том 5, №7 (64). – С. 236-240:
https://www.elibrary.ru/download/elibrary_54235792_69645821.pdf Дата доступа: 23.09.2025
4. Боева Н. В. Базы данных и системы управления базами данных. Модели баз данных. Сравнительные характеристики и особенности. Реляционные модели баз данных / Н. В. Боева // Форум ученых. – 2017. Том 9, №13. – С. 127-130: <https://cyberleninka.ru/article/n/bazy-dannyh-i-sistemy-upravleniya-bazami-dannyh-modeli-baz-dannyh-sravnitelnye-harakteristiki-i-osobennosti-relyatsionnye-modeli/viewer> Дата доступа: 23.09.2025
5. Бурнаев А. Н., Проценко И. Г. Физическая модель и нормализация БД ИСК (КТЕСТ) / А. Н. Бурнаев, И. Г. Проценко // Природные ресурсы, их современное состояние, охрана, промысловое и техническое использование. – 2024. – С. 181-185: https://www.elibrary.ru/download/elibrary_75191872_45265786.pdf Дата доступа: 24.09.2025
6. Карпова Т. С. Базы данных: модели, разработка, реализация. СПб.: Питер, 2002 – 304 Дата доступа: 24.09.2025
7. Маклаков, С.А. ВРwin и ERwin. CASE-средства для разработки информационных систем. — М.: Диалог-МИФИ, 2013 – 456 с Дата доступа: 24.09.2025
8. Физическое проектирование базы данных [Электронный ресурс]: <https://scienceforum.ru/2017/article/2017030333> Дата доступа 24.09.2025
9. MYSQL – что это такое простыми словами [Электронный ресурс]: <https://mchost.ru/articles/chto-takoe-mysql/> Дата доступа: 24.09.2025
10. Что такое MariaDB [Электронный ресурс]: <https://workspace.ru/tools/database/mariadb/> Дата доступа: 25.09.2025
11. PostgreSQL: все что нужно знать для быстрого старта [Электронный ресурс]: <https://www.reg.ru/blog/postgresql-vsyo-chto-nuzhno-znat-dlya-bystrogo-starta/> Дата доступа: 25.09.2025
12. MS Access – один из лучших инструментов для обработки данных [Электронный ресурс]: <https://zaman.ru/articles/MS-Access/> Дата доступа: 25.09.2025
13. Горелов С. В. «Современные технологии программирования: разработка Windows-приложений на языке C#. Том 1». - Дата доступа: 25.09.2025

Приложение 1. Техническое задание

МИНОБРАНАУКИ РОССИИ
федеральное государственное бюджетное
образовательное учреждение высшего образования
ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Институт информационных технологий
наименование института (факультета)
Кафедра математического и программного обеспечения 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 «Звездный экран»
Техническое задание на курсовую работу
Листов 6

Руководитель: доцент Селяничев О.Л.
Исполнитель: студент гр. 1ИСб-01-2оп-22
Сухарев Е.С.

Череповец, 2025 г.

В данном документе описаны требования к созданию базы данных для кинотеатра. База данных предназначена для хранения информации о фильмах, залах, сеансах, билетах, зрителях и персонале, обеспечивающем работу кинотеатра. Основная цель разработки базы данных – автоматизация процессов управления кинотеатром, упрощение работы с данными и повышение эффективности обслуживания зрителей. Система позволит быстро получать актуальную информацию о репертуаре, расписании сеансов, доступных местах, а также обеспечит надежное хранение данных о посетителях и сотрудниках кинотеатра.

1. Основание для разработки

Основанием для разработки является задание на курсовую работу по дисциплине "Проектирование баз данных", выданное на кафедре МПО ЭВМ ИИТ ЧГУ.

Дата утверждения: 10 февраля 2025 года.

Наименование темы разработки: «Проектирование базы данных «Звездный экран».

2. Назначение разработки

Проектирование базы данных осуществляется в рамках учебной работы с целью углубления и закрепления знаний по курсу «Проектирование баз данных».

3. Требования к программе

3.1. Требования к функциональным характеристикам

База данных должна обладать изменяемыми таблицами, которые содержат информацию о работе кинотеатра:

- фильмы;
- сеансы;
- залы;
- билеты;
- зрители;
- кассиры;
- администраторы;
- технический персонал;
- тарифы;
- категории залов.

Приложение должно выполнять запросы:

- Зрители старше 18 лет;
- Фильмы длительностью больше 120 минут;

- Билеты по стоимости;
- Фильмы по длительности;
- Количество зрителей по возрастам;
- Средняя цена билета;
- Администраторы дневной смены;
- Проданные билеты по тарифу;
- Количество проданных билетов по каждому залу;
- Средняя цена билета по тарифам;
- Общая выручка по каждому тарифу;
- Максимальная вместимость зала;
- Количество фильмов по жанру;
- Технический персонал со специализацией "Электрик";
- Самый дорогой билет;
- Технический персонал ночной смены;
- Самые вместительные залы;
- Количество техников по сменам;
- Почты всех сотрудников;
- Номера телефонов всех сотрудников;
- Количество сеансов у каждого техника;
- Количество сеансов у каждого администратора;
- Количество администраторов по каждой зоне ответственности;
- Билеты по категориям залов;
- Сеансы с указанием фильма и зала.

На главном окне программы (рис.П1.1) необходимо расположить:

1. окно для вывода таблиц;
2. выбор отображаемых данных;
3. действия над базой данных (добавление, удаление и изменение).

Рис. П1.1. Главное окно приложения

На окне добавления записи (рис.П1.2) необходимо расположить:

1. поля ввода данных;
2. кнопка добавления записи;

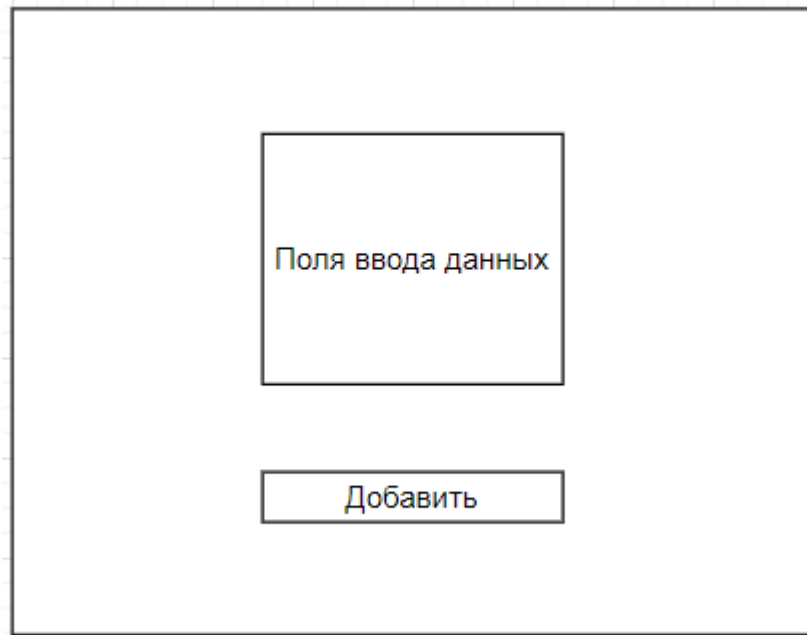


Рис.П1.2. Окно добавления записи

На окне изменения записи (рис.П1.3) необходимо расположить:

1. поля ввода данных;
2. кнопка сохранения изменений;
3. кнопка отмены действия

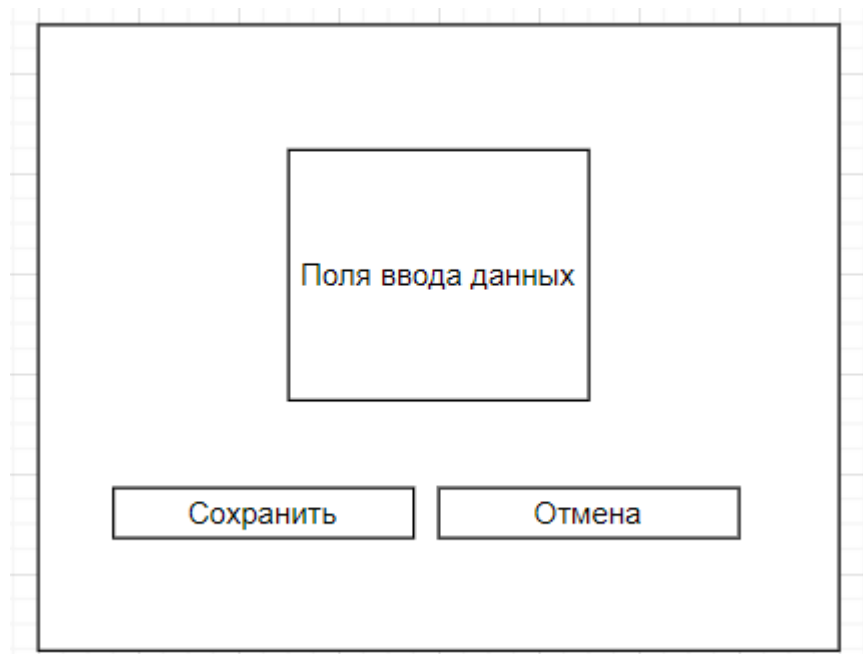


Рис.П1.3. Окно изменения записи

3.2. Требования к надёжности

Для надёжности программы необходима исправность работы всех функций. Требуется защита от действий пользователя, которые могут привести приложение в неисправное состояние. Рекомендуется закрыть все открытые программы. Программа должна выполнять валидацию данных.

3.3. Условия эксплуатации

Для работы программы потребуется:

- исправный компьютер;
- съемный накопитель без повреждений.

3.4. Требования к составу и параметрам технических средств

Минимальные требования необходимые для компьютера, чтобы человек мог пользоваться программой:

- процессор с тактовой частотой не менее 1,2 ГГц;
- оперативная память 4 Гб;
- свободное место на жестком диске: до 1 Гб;
- интегрированная видеокарта;
- монитор;
- клавиатура;
- компьютерная мышь;
- порт USB 2.0.

3.5. Требования к информационной и программной совместимости

Для корректной работы программного обеспечения необходимо использовать компьютер с операционной системой Windows 10 и с установленными библиотеками .NET Framework.

3.6. Требования к маркировке и упаковке

Съемный накопитель должен быть с USB-интерфейсом не ниже версии 2.0.

3.7. Требования к хранению и транспортированию

Для корректной работы программы файлы нужно хранить на компьютере или съемном накопителе в одной папке.

3.8. Специальные требования

Реализовать интерфейс в виде, привычном для любого пользователя – в стиле Microsoft Office.

4. Требования к программной документации

4.1. Содержание расчётно-пояснительной записки

Программная документация должна содержать расчётно-пояснительную записку (РПЗ) с содержанием:

- Титульный лист;
 Оглавление;
 Введение;
 1. Описание предметной области;
 2. Инфологическое проектирование;
 3. Даталогическое проектирование;
 3.1. Нормализация;
 3.1.1. Ненормализованная таблица;
 3.1.2. Первая нормальная форма;
 3.1.3. Вторая нормальная форма;
 3.1.4. Третья нормальная форма;
 3.2. Логическая модель;
 3.3. Физическая модель;
 4. Описание программного обеспечения;
 Заключение;
 Список литературы;
 Приложения.

4.2. Требования к оформлению

Нужно, чтобы соблюдались все требования к оформлению на протяжении выполнения всей работы (табл. П1.1).

Таблица П1.1

Требования к оформлению

Документ	Печать на отдельных листах формата А4 (210х297 мм); обратная 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 — выравнены по левому краю, числа, даты — по правому.

5. Стадии и этапы разработки

Информация о стадиях и этапах разработки представлена в табличном виде (табл. П1.2).

Таблица П1.2

Стадии и этапы разработки

Наименование этапа разработки	Сроки разработки	Результат выполнения	Отметка о выполнении
1.Выбор темы	10 февраля 2025 г.	Выбранная тема	
2. Оформление технического задания	10 февраля – 20 февраля 2025 г.	Оформленное техническое задание	
3.Описание предметной области	21 февраля – 12 февраля 2025 г.	Обозначены характеристики объекта для проектирования, функционирование предметной области	
4.Проектирование базы данных	13 февраля – 29 апреля 2025 г.	Инфологическое и даталогическое проектирование	
5.Написание программы	29 апреля – 16 мая 2025 г.	Созданная программа и описание её работы	
6. Оформление руководства пользователя	18 мая 2025 г.	Написано руководство	
7.Оформление сопроводительной документации - РПЗ	25 апреля – 22 мая 2025 г.	Оформленная РПЗ	

6. Порядок контроля и приемки

Данные о порядке контроля и приемки находятся в 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
Разработка технического задания	19 февраля 2025 г.	Оформленное техническое задание	
Проектирование базы данных	29 апреля 2025 г.	Создана схема БД	
Разработка программы	16 мая 2025 г.	Создана конечная версия программы	
Оформление руководства пользователя	18 мая 2025 г.	Оформлено руководство	
Оформление РПЗ	22 мая 2025 г.	Оформленная РПЗ	
Сдача РПЗ	29 мая 2025 г.	Сданная РПЗ	

Файл "Form1.cs":

```
using System;
using System.Data;
using System.Data.OleDb;
using System.Windows.Forms;
using System.IO;
using System.Reflection;
using static
System.Windows.Forms.VisualStyles.VisualStyleElement;

namespace KR
{
    public partial class Form1 : Form
    {
        // База данных в папке приложения
        private string connectionString =
        @"Provider=Microsoft.ACE.OLEDB.12.0;Data
        Source={Path.Combine(Application.StartupPath,
        "CinemaDB.mdb")};";

        public Form1()
        {
            InitializeComponent();

            // Проверяем существование файла базы данных при
            запуске
            CheckDatabaseExists();
        }

        // Метод для проверки существования базы данных
        private void CheckDatabaseExists()
        {
            // Извлекаем путь к базе данных из строки подключения
            string dbPath = ExtractDatabasePath(connectionString);

            if (!File.Exists(dbPath))
            {
                MessageBox.Show($"Файл базы данных не найден по
                пути:\n{dbPath}\n\nПожалуйста, убедитесь что файл находится в
                правильной директории.",
                "Ошибка", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
            }
        }

        // Метод для извлечения пути к базе данных из строки
        подключения
        private string ExtractDatabasePath(string connectionString)
        {
            string[] parts = connectionString.Split(';');
            foreach (string part in parts)
            {
                if (part.Trim().StartsWith("Data Source=",
                StringComparison.OrdinalIgnoreCase))
                {
                    return part.Substring(part.IndexOf('=') + 1);
                }
            }
            return "";
        }

        private void add_Click(object sender, EventArgs e)
        {
            if (comboBoxTable.SelectedItem == null)
            {
                MessageBox.Show("Пожалуйста, выберите таблицу
                перед добавлением записи.", "Ошибка", MessageBoxButtons.OK,
                MessageBoxIcon.Warning);
                return;
            }

            string selectedTable =
            comboBoxTable.SelectedItem.ToString();
            TableAddForm f = new TableAddForm(connectionString,
            selectedTable);
        }
    }
}
```

Приложение 2. Текст программы

```
f.ShowDialog();
PrintTable(selectedTable);
}

private void delete_Click(object sender, EventArgs e)
{
    if (comboBoxTable.SelectedItem == null)
    {
        MessageBox.Show("Пожалуйста, выберите таблицу
        перед удалением записи.", "Ошибка", MessageBoxButtons.OK,
        MessageBoxIcon.Warning);
        return;
    }

    if (dataGridView1.SelectedRows.Count == 0)
    {
        MessageBox.Show("Пожалуйста, выберите строку для
        удаления.", "Ошибка", MessageBoxButtons.OK,
        MessageBoxIcon.Warning);
        return;
    }

    string selectedTable =
    comboBoxTable.SelectedItem.ToString();
    try
    {
        using (OleDbConnection connection = new
        OleDbConnection(connectionString))
        {
            connection.Open();
            string primaryKeyColumn =
            dataGridView1.Columns[0].Name;
            string primaryKeyValue =
            dataGridView1.SelectedRows[0].Cells[0].Value.ToString();
            string query = $"DELETE FROM [{selectedTable}]
            WHERE [{primaryKeyColumn}] = @id";

            using (OleDbCommand command = new
            OleDbCommand(query, connection))
            {
                command.Parameters.AddWithValue("@id",
                primaryKeyValue);
                int rowsAffected = command.ExecuteNonQuery();

                if (rowsAffected > 0)
                {
                    MessageBox.Show("Запись успешно удалена.",
                    "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);
                }
                else
                {
                    MessageBox.Show("Не удалось удалить запись.",
                    "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }
            PrintTable(selectedTable);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("Ошибка при удалении: " +
        ex.Message, "Ошибка", MessageBoxButtons.OK,
        MessageBoxIcon.Error);
    }
}

private void edit_Click(object sender, EventArgs e)
{
    if (comboBoxTable.SelectedItem == null)
    {
        MessageBox.Show("Пожалуйста, выберите таблицу
        перед редактированием записи.", "Ошибка",
        MessageBoxButtons.OK, MessageBoxIcon.Warning);
        return;
    }
}
```

```

        if (dataGridView1.SelectedRows.Count == 0)
        {
            MessageBox.Show("Пожалуйста, выберите строку для редактирования.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);
            return;
        }

        string selectedTable =
        comboBoxTable.SelectedItem.ToString();
        DataRow selectedRow =
        ((DataRowView)dataGridView1.SelectedRows[0].DataBoundItem).Row;

        TableEdit f = new TableEdit(connectionString, selectedTable, selectedRow);
        f.ShowDialog();
        PrintTable(selectedTable);
    }

    public void PrintTable(string tableName)
    {
        try
        {
            using (OleDbConnection connection = new OleDbConnection(connectionString))
            {
                connection.Open();
                string query = $"SELECT * FROM [{tableName}]";
                OleDbDataAdapter dataAdapter = new OleDbDataAdapter(query, connection);
                DataTable dataTable = new DataTable();
                dataAdapter.Fill(dataTable);
                dataGridView1.DataSource = dataTable;
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show("Ошибка: " + ex.Message);
        }
    }

    public void ExecuteCustomQuery(string query)
    {
        try
        {
            using (OleDbConnection connection = new OleDbConnection(connectionString))
            {
                connection.Open();
                OleDbDataAdapter dataAdapter = new OleDbDataAdapter(query, connection);
                DataTable dataTable = new DataTable();
                dataAdapter.Fill(dataTable);
                dataGridView1.DataSource = dataTable;
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show("Ошибка: " + ex.Message);
            dataGridView1.DataSource = null;
            dataGridView1.Rows.Clear();
            dataGridView1.Columns.Clear();
        }
    }

    private void comboBoxZapros_SelectedIndexChanged(object sender, EventArgs e)
    {
        if (comboBoxZapros.SelectedItem == null)
            return;

        string selectedQuery =
        comboBoxZapros.SelectedItem.ToString();
        ExecuteQueryFromAccess(selectedQuery);
    }

    public void ExecuteQueryFromAccess(string queryName)

```

```

    {
        try
        {
            using (OleDbConnection connection = new OleDbConnection(connectionString))
            {
                connection.Open();
                string query = $"SELECT * FROM [{queryName}]";

                OleDbDataAdapter dataAdapter = new OleDbDataAdapter(query, connection);
                DataTable dataTable = new DataTable();
                dataAdapter.Fill(dataTable);
                dataGridView1.DataSource = dataTable;
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show("Ошибка при выполнении запроса: " + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
            dataGridView1.DataSource = null;
            dataGridView1.Rows.Clear();
            dataGridView1.Columns.Clear();
        }
    }

    private void button1_Click(object sender, EventArgs e)
    {
        MessageBox.Show(
            "Программа для управления базой данных кинотеатра\n\n" +
            "Версия: 1.0\n" +
            "Автор: студент ИИСб-01-2оп-22 Сухарев Евгений\n" +
            "Дата создания: 2 мая 2025\n\n" +
            "Описание:\n" +
            "Данная программа предназначена для управления базой данных.\n" +
            "связанной с кинотеатром.\n" +
            "Основные функции:\n" +
            "- Просмотр таблиц базы данных\n" +
            "- Добавление новых записей\n" +
            "- Редактирование существующих записей\n" +
            "- Удаление записей\n" +
            "- Выполнение пользовательских запросов\n\n" +
            "Для работы с программой выберите таблицу из выпадающего\n" +
            "списка и используйте кнопки для выполнения нужных действий.",
            "О программе",
            MessageBoxButtons.OK,
            MessageBoxIcon.Information
        );
    }
}

```

Файл "TableAddForm.cs"

```

using System;
using System.Collections.Generic;
using System.Data;
using System.Data.OleDb;
using System.Drawing;
using System.Linq;
using System.Windows.Forms;

namespace KR
{
    public partial class TableAddForm : Form
    {
        private string connectionString;
        private string tableName;
        private List<Control> inputControls = new List<Control>();
        private List<string> columnNames = new List<string>();
        private List<bool> isPrimaryKey = new List<bool>();
        private List<Type> columnTypes = new List<Type>();
    }
}

```

```

private List<bool> isAutoNumber = new List<bool>();
private List<bool> isForeignKey = new List<bool>();
private List<string> referencedTables = new List<string>();
private List<bool> isNullable = new List<bool>();

// Словарь начальных значений AutoNumber для каждой
таблицы
private Dictionary<string, int> autoNumberStartValues = new
Dictionary<string, int>()
{
    {"Жанр", 100000},
    {"Администратор", 200000},
    {"Билет", 300000},
    {"Возрастные ограничения", 400000},
    {"Зал", 500000},
    {"Зона ответственности", 600000},
    {"Зритель", 700000},
    {"Кассир", 800000},
    {"Категория зала", 900000},
    {"Сеанс", 1000000},
    {"Смена", 1100000},
    {"Специализация", 1200000}, // Исправлено с 12000 на
1200000
    {"Тариф", 1300000},
    {"Технический персонал", 1400000},
    {"Фильм", 1500000}
};

public TableAddForm(string connStr, string tableName)
{
    InitializeComponent();
    this.connectionString = connStr;
    this.tableName = tableName;
    this.Text = $"Добавление записи в таблицу: {tableName}";
    this.Font = new Font("Segoe UI", 10);
    this.StartPosition = FormStartPosition.CenterParent;

    // Устанавливаем начальное значение AutoNumber для
таблицы
    SetAutoNumberStartValue(tableName);

    LoadFields();
}

private void SetAutoNumberStartValue(string tableName)
{
    // Проверяем, есть ли настройка для этой таблицы
    if (!autoNumberStartValues.ContainsKey(tableName))
        return;

    try
    {
        using (OleDbConnection conn = new
OleDbConnection(connectionString))
        {
            conn.Open();

            // Проверяем, есть ли уже записи в таблице
            string checkQuery = $"SELECT COUNT(*) FROM
[{tableName}]";
            using (OleDbCommand checkCmd = new
OleDbCommand(checkQuery, conn))
            {
                int recordCount =
Convert.ToInt32(checkCmd.ExecuteScalar());

                // Если таблица пустая, устанавливаем начальное
значение
                if (recordCount == 0)
                {
                    int startValue =
autoNumberStartValues[tableName];

                    // Находим имя поля AutoNumber (обычно это
первичный ключ)
                    string autoNumberField =
GetAutoNumberFieldName(conn, tableName);

```

```

        if (!string.IsNullOrEmpty(autoNumberField))
        {
            System.Diagnostics.Debug.WriteLine($"Попытка установить
начальное значение {startValue} для поля {autoNumberField} в
таблице {tableName}");

            // Метод 1: Попробуем ALTER TABLE
            try
            {
                string alterQuery = $"ALTER TABLE
[{tableName}] ALTER COLUMN [{autoNumberField}]
COUNTER({startValue},1)";
                using (OleDbCommand alterCmd = new
OleDbCommand(alterQuery, conn))
                {
                    alterCmd.ExecuteNonQuery();

                    System.Diagnostics.Debug.WriteLine($"Успешно
установлено
через ALTER TABLE");
                    return;
                }
            }
            catch (Exception ex1)
            {
                System.Diagnostics.Debug.WriteLine($"ALTER TABLE не
сработал: {ex1.Message}");
            }

            // Метод 2: Вставляем запись с нужным ID,
затем удаляем
            try
            {
                // Получаем список всех полей таблицы
(кроме AutoNumber)
                var nonAutoFields =
GetNonAutoNumberFields(conn, tableName, autoNumberField);

                // Создаем INSERT запрос с явным
указанием AutoNumber значения
                string insertQuery;
                if (nonAutoFields.Count > 0)
                {
                    // Если есть другие поля, вставляем
минимальные значения
                    string fieldsStr = $"[{autoNumberField}], "
+ string.Join(", ", nonAutoFields.Select(f => $"[{f.Key}]"));
                    string valuesStr = "?," + string.Join(", ",
nonAutoFields.Select(f => "?"));
                    insertQuery = $"INSERT INTO
[{tableName}] ({fieldsStr}) VALUES ({valuesStr})";

                    using (OleDbCommand insertCmd = new
OleDbCommand(insertQuery, conn))
                    {
                        insertCmd.Parameters.Add("?",
OleDbType.Integer).Value = startValue - 1;

                        // Добавляем параметры для остальных
полей
                        foreach (var field in nonAutoFields)
                        {
                            if (field.Value == typeof(string))

                                insertCmd.Parameters.AddWithValueValue("?", "temp");
                            else if (field.Value == typeof(int))

                                insertCmd.Parameters.AddWithValueValue("?", 0);
                            else if (field.Value ==
typeof(DateTime))

                                insertCmd.Parameters.AddWithValueValue("?", DateTime.Now);
                            else

                                insertCmd.Parameters.AddWithValueValue("?", DBNull.Value);
                        }

```

```

        insertCmd.ExecuteNonQuery();
    }
    else
    {
        // Если только AutoNumber поле
        insertQuery = $"INSERT INTO
[{tableName}] [{autoNumberField}] VALUES (?";
        using (OleDbCommand insertCmd = new
OleDbCommand(insertQuery, conn))
        {
            insertCmd.Parameters.Add("?",
OleDbType.Integer).Value = startValue - 1;
            insertCmd.ExecuteNonQuery();
        }

        // Удаляем вставленную запись
        string deleteQuery = $"DELETE FROM
[{tableName}] WHERE [{autoNumberField}] = ?";
        using (OleDbCommand deleteCmd = new
OleDbCommand(deleteQuery, conn))
        {
            deleteCmd.Parameters.Add("?",
OleDbType.Integer).Value = startValue - 1;
            deleteCmd.ExecuteNonQuery();
        }

        System.Diagnostics.Debug.WriteLine($"Успешно установлено
через INSERT/DELETE");
    }
    catch (Exception ex2)
    {
        System.Diagnostics.Debug.WriteLine($"INSERT/DELETE не
сработал: {ex2.Message}");
    }
    }
    else
    {
        System.Diagnostics.Debug.WriteLine($"Таблица
{tableName} не пустая ({recordCount} записей), пропускаем
установку начального значения");
    }
    }
    catch (Exception ex)
    {
        System.Diagnostics.Debug.WriteLine($"Ошибка
установки начального значения AutoNumber для {tableName}:
{ex.Message}");
    }
}

private Dictionary<string, Type>
GetNonAutoNumberFields(OleDbConnection conn, string
tableName, string autoNumberField)
{
    var fields = new Dictionary<string, Type>();

    try
    {
        OleDbCommand cmd = new OleDbCommand($"SELECT
* FROM [{tableName}] WHERE 1=0", conn);
        OleDbDataReader reader = cmd.ExecuteReader();
        DataTable schemaTable = reader.GetSchemaTable();
        reader.Close();

        foreach (DataRow row in schemaTable.Rows)
        {
            string colName = row["ColumnName"].ToString();
            if (colName != autoNumberField)
            {
                Type colType = (Type)row["DataType"];
            }
        }
    }
}

```

```

        bool allowsNull =
Convert.ToBoolean(row["AllowDBNull"]);

        // Добавляем только поля, которые могут быть
        NULL или имеют значения по умолчанию
        if (allowsNull || colType == typeof(string) || colType
== typeof(int) || colType == typeof(DateTime))
        {
            fields[colName] = colType;
        }
    }
    }
    catch (Exception ex)
    {
        System.Diagnostics.Debug.WriteLine($"Ошибка
получения полей: {ex.Message}");
    }

    return fields;
}

private string GetAutoNumberFieldName(OleDbConnection
conn, string tableName)
{
    try
    {
        // Получаем схему таблицы
        OleDbCommand cmd = new OleDbCommand($"SELECT
* FROM [{tableName}] WHERE 1=0", conn);
        OleDbDataReader reader = cmd.ExecuteReader();
        DataTable schemaTable = reader.GetSchemaTable();
        reader.Close();

        // Получаем информацию о первичных ключах
        DataTable primaryKeysTable =
conn.GetOleDbSchemaTable(OleDbSchemaGuid.Indexes,
new object[] { null, null, null, null, tableName });
        HashSet<string> primaryKeyColumns =
new HashSet<string>();
        foreach (DataRow row in primaryKeysTable.Rows)
        {
            bool isPrimary =
Convert.ToBoolean(row["PRIMARY_KEY"]);
            if (isPrimary)
            {
                string colName =
row["COLUMN_NAME"].ToString();
                primaryKeyColumns.Add(colName);
            }
        }

        // Ищем AutoNumber поле (первичный ключ типа
int/long)
        foreach (DataRow row in schemaTable.Rows)
        {
            string colName = row["ColumnName"].ToString();
            Type colType = (Type)row["DataType"];
            bool isKey = primaryKeyColumns.Contains(colName);

            if (isKey && (colType == typeof(int) || colType ==
typeof(long)))
            {
                return colName;
            }
        }
    }
    catch (Exception ex)
    {
        System.Diagnostics.Debug.WriteLine($"Ошибка
поиска AutoNumber поля: {ex.Message}");
    }

    return null;
}

private void LoadFields()
{
}

```

```

this.SuspendLayout();
this.Controls.Clear();
inputControls.Clear();
columnNames.Clear();
isPrimaryKey.Clear();
columnTypes.Clear();
isAutoNumber.Clear();
isForeignKey.Clear();
referencedTables.Clear();
isNullable.Clear();

using (OleDbConnection conn = new
OleDbConnection(connectionString))
{
    conn.Open();

    // Получаем информацию о внешних ключах
    var foreignKeys = GetForeignKeys(conn, tableName);

    // Получаем схему таблицы
    OleDbCommand cmd = new OleDbCommand($"SELECT
* FROM [{tableName}] WHERE 1=0", conn);
    OleDbDataReader reader = cmd.ExecuteReader();
    DataTable schemaTable = reader.GetSchemaTable();

    // Получаем информацию о первичных ключах
    DataTable primaryKeysTable =
conn.GetOleDbSchemaTable(OleDbSchemaGuid.Indexes, new
object[] { null, null, null, null, tableName });
    HashSet<string> primaryKeyColumns = new
HashSet<string>();
    foreach (DataRow row in primaryKeysTable.Rows)
    {
        bool isPrimary =
Convert.ToBoolean(row["PRIMARY_KEY"]);
        if (isPrimary)
        {
            string colName =
row["COLUMN_NAME"].ToString();
            primaryKeyColumns.Add(colName);
        }
    }

    int y = 20;
    int labelWidth = 180;
    int controlWidth = 250;
    int rowHeight = 40;

    foreach (DataRow row in schemaTable.Rows)
    {
        string colName = row["ColumnName"].ToString();
        bool isKey = primaryKeyColumns.Contains(colName);
        Type colType = (Type)row["DataType"];
        bool allowsNull =
Convert.ToBoolean(row["AllowDBNull"]);

        // Для Access AutoNumber - это обычно первичный
        // ключ типа int/long
        bool isAutoNum = isKey && (colType == typeof(int) ||
colType == typeof(long));

        bool isForeignKeyColumn =
foreignKeys.ContainsKey(colName);
        string referencedTable = isForeignKeyColumn ?
foreignKeys[colName] : "";

        // Диагностика для отладки
        if (tableName == "Специализация")
        {
            System.Diagnostics.Debug.WriteLine($"Поле:
{colName}");
            System.Diagnostics.Debug.WriteLine($" -
Первичный ключ: {isKey}");
            System.Diagnostics.Debug.WriteLine($" - Тип:
{colType}");
            System.Diagnostics.Debug.WriteLine($" - Внешний
ключ: {isForeignKeyColumn}");
        }

        System.Diagnostics.Debug.WriteLine($" -
AutoNumber: {isAutoNum}");
        System.Diagnostics.Debug.WriteLine($" - Будет
скрыт: {isKey && (colType == typeof(int) || colType ==
typeof(long)) && !isForeignKeyColumn}");
    }

    columnNames.Add(colName);
    isPrimaryKey.Add(isKey);
    columnTypes.Add(colType);
    isAutoNumber.Add(isAutoNum);
    isForeignKey.Add(isForeignKeyColumn);
    referencedTables.Add(referencedTable);
    isNullable.Add(allowsNull);

    // AutoNumber поля (первичные ключи) не
    // показываем пользователю - они генерируются автоматически
    if (isKey && (colType == typeof(int) || colType ==
typeof(long)) && !isForeignKeyColumn)
    {
        System.Diagnostics.Debug.WriteLine($"Скрываем
поле {colName} в таблице {tableName} (AutoNumber)");
        continue; // Пропускаем AutoNumber первичные
        // ключи
    }

    // Определяем тип метки
    string labelText = colName;
    if (isForeignKeyColumn)
    {
        labelText += " (выберите)";
    }
    else
    {
        labelText += ":";
    }

    Label label = new Label()
    {
        Text = labelText,
        Location = new Point(20, y),
        Width = labelWidth,
        TextAlign = ContentAlignment.MiddleRight,
        Font = new Font("Segoe UI", 10),
        ForeColor = Color.Black
    };

    Control inputControl;

    if (isForeignKeyColumn)
    {
        // Для внешних ключей - ComboBox для выбора
        ComboBox cb = new ComboBox()
        {
            Location = new Point(20 + labelWidth + 10, y),
            Width = controlWidth,
            Font = new Font("Segoe UI", 10),
            DropDownStyle = ComboBoxStyle.DropDownList,
            BackColor = Color.LightYellow,
            DrawMode = DrawMode.OwnerDrawFixed,
            ItemHeight = 35
        };
        cb.DrawItem += ComboBox_DrawItem;
        cb.SelectedIndexChanged +=
ComboBox_SelectedIndexChanged;
        LoadComboBoxData(cb, referencedTable, conn,
colName);
        inputControl = cb;
    }
    else
    {
        // Обычное текстовое поле
        TextBox tb = new TextBox()
        {
            Location = new Point(20 + labelWidth + 10, y),
            Width = controlWidth,
            Font = new Font("Segoe UI", 10),
            BackColor = Color.White
        };
    }
}

```

```

    };
    inputControl = tb;
}

inputControls.Add(inputControl);
this.Controls.Add(label);
this.Controls.Add(inputControl);
y += rowHeight;
}

Button btn = new Button()
{
    Text = "Добавить запись",
    Location = new Point(20 + labelWidth + 10, y + 20),
    Size = new Size(150, 35),
    Font = new Font("Segoe UI", 10, FontStyle.Bold),
    BackColor = Color.LightGreen
};
btn.Click += Btn_Click;
this.Controls.Add(btn);

int formWidth = 20 + labelWidth + 10 + controlWidth + 20;
int formHeight = y + 80;
this.ClientSize = new Size(formWidth, formHeight);
}

this.ResumeLayout(true);
}

private Dictionary<string, string>
GetForeignKeys(OleDbConnection conn, string tableName)
{
    var foreignKeys = new Dictionary<string, string>();

    try
    {
        DataTable foreignKeysTable =
            conn.GetOleDbSchemaTable(OleDbSchemaGuid.Foreign_Keys,
            new object[] { null, null, null, null, null, tableName });

        foreach (DataRow row in foreignKeysTable.Rows)
        {
            string fkColumnName =
                row["FK_COLUMN_NAME"].ToString();
            string pkTableName =
                row["PK_TABLE_NAME"].ToString();
            foreignKeys[fkColumnName] = pkTableName;
        }
    }
    catch (Exception ex)
    {
        System.Diagnostics.Debug.WriteLine($"Ошибка
        получения внешних ключей: {ex.Message}");
        foreignKeys = GetForeignKeysByNaming(conn,
        tableName);
    }

    return foreignKeys;
}

private Dictionary<string, string>
GetForeignKeysByNaming(OleDbConnection conn, string
tableName)
{
    var foreignKeys = new Dictionary<string, string>();

    DataTable tablesTable =
        conn.GetOleDbSchemaTable(OleDbSchemaGuid.Tables,
        new object[] { null, null, null, "TABLE" });
    List<string> allTables = new List<string>();
    foreach (DataRow row in tablesTable.Rows)
    {
        allTables.Add(row["TABLE_NAME"].ToString());
    }

    // Получаем информацию о первичных ключах для этой
    таблицы

    DataTable primaryKeysTable =
        conn.GetOleDbSchemaTable(OleDbSchemaGuid.Indexes,
        new object[] { null, null, null, null, tableName });
    HashSet<string> primaryKeyColumns =
        new HashSet<string>();
    foreach (DataRow row in primaryKeysTable.Rows)
    {
        bool isPrimary =
            Convert.ToBoolean(row["PRIMARY_KEY"]);
        if (isPrimary)
        {
            string colName = row["COLUMN_NAME"].ToString();
            primaryKeyColumns.Add(colName);
        }
    }

    OleDbCommand cmd = new OleDbCommand($"SELECT *
    FROM [{tableName}] WHERE 1=0", conn);
    OleDbDataReader reader = cmd.ExecuteReader();
    DataTable schemaTable = reader.GetSchemaTable();

    foreach (DataRow row in schemaTable.Rows)
    {
        string colName = row["ColumnName"].ToString();
        Type colType = (Type)row["DataType"];

        // Диагностика для "Специализация"
        if (tableName == "Специализация")
        {
            System.Diagnostics.Debug.WriteLine($"GetForeignKeysByNaming
            - Поле: {colName}, Первичный ключ:
            {primaryKeyColumns.Contains(colName)}");

            // НЕ обрабатываем первичные ключи как внешние!
            if (primaryKeyColumns.Contains(colName))
            {
                continue;
            }

            if (colType == typeof(int) || colType == typeof(long)) //
            AutoNumber обычно int или long
            {
                if (colName.StartsWith("Код") && colName != "Код
                клиента" && colName != "Код типа клиента")
                {
                    string possibleTableName =
                        colName.Substring(3).Trim();

                    var matchingTable = allTables.FirstOrDefault(t =>
                    t.Equals(possibleTableName,
                    StringComparison.OrdinalIgnoreCase) ||
                    t.Equals(possibleTableName + "ы",
                    StringComparison.OrdinalIgnoreCase) ||
                    t.Equals(possibleTableName + "а",
                    StringComparison.OrdinalIgnoreCase));

                    if (!string.IsNullOrEmpty(matchingTable))
                    {
                        foreignKeys[colName] = matchingTable;
                    }
                }
                else if ((tableName == "Физлица" || tableName ==
                "Юрлица") && colName == "Код клиента")
                {
                    foreignKeys[colName] = "Клиенты";
                }
                else if (tableName == "Клиенты" && colName == "Код
                типа клиента")
                {
                    foreignKeys[colName] = "Тип клиента";
                }
            }
        }
    }

    return foreignKeys;
}

```

```

private void LoadComboBoxData(ComboBox comboBox, string
referencedTable, OleDbConnection conn, string columnName)
{
    try
    {
        OleDbCommand cmd = new OleDbCommand($"SELECT
* FROM [{referencedTable}] WHERE 1=0", conn);
        OleDbDataReader reader = cmd.ExecuteReader();
        DataTable schemaTable = reader.GetSchemaTable();
        reader.Close();

        string primaryKeyColumn = null;
        string displayColumn = null;

        foreach (DataRow row in schemaTable.Rows)
        {
            string colName = row["ColumnName"].ToString();
            Type colType = (Type)row["DataType"];

            // Ищем первичный ключ (обычно AutoNumber для
Access)
            if ((colType == typeof(int) || colType == typeof(long))
&& primaryKeyColumn == null)
            {
                primaryKeyColumn = colName;
            }
            else if (colType == typeof(string) && displayColumn ==
null)
            {
                displayColumn = colName;
            }
        }

        if (primaryKeyColumn == null)
        {
            comboBox.Items.Add("Нет доступных данных");
            return;
        }

        if (displayColumn == null)
            displayColumn = primaryKeyColumn;

        string query = $"SELECT [{primaryKeyColumn}],
[{displayColumn}] FROM [{referencedTable}]";
        cmd = new OleDbCommand(query, conn);
        reader = cmd.ExecuteReader();

        var items = new List<ComboBoxItem>();

        while (reader.Read())
        {
            int id = reader.GetInt32(0);
            string display = displayColumn == primaryKeyColumn ?
id.ToString() : reader.GetString(1);

            if (referencedTable == "Клиенты")
            {
                string clientInfo = GetClientDisplayInfo(conn, id);
                if (!string.IsNullOrEmpty(clientInfo))
                {
                    display = clientInfo;
                }
            }

            items.Add(new ComboBoxItem { Value = id, Text =
display, Code = id.ToString() });
        }
        reader.Close();

        comboBox.DisplayMember = "Text";
        comboBox.ValueMember = "Value";
        comboBox.DataSource = items;

        if (items.Count == 0)
        {
            comboBox.DataSource = null;
            comboBox.Items.Add("Нет данных для выбора");
        }
    }
}

```

```

    }
}
catch (Exception ex)
{
    System.Diagnostics.Debug.WriteLine($"Ошибка загрузки
данных для ComboBox: {ex.Message}");
    comboBox.Items.Add("Ошибка загрузки данных");
}

private string GetClientDisplayInfo(OleDbConnection conn, int
clientId)
{
    try
    {
        string query = "SELECT Фамилия, Имя FROM [Физлица]
WHERE [Код клиента] = ?";
        using (OleDbCommand cmd = new
OleDbCommand(query, conn))
        {
            cmd.Parameters.Add("?", OleDbType.Integer).Value =
clientId;
            using (OleDbDataReader reader = cmd.ExecuteReader())
            {
                if (reader.Read())
                {
                    string surname = reader.IsDBNull(0) ? "" :
reader.GetString(0);
                    string name = reader.IsDBNull(1) ? "" :
reader.GetString(1);
                    return $"{surname} {name} (Физ.лицо)".Trim();
                }
            }
        }

        query = "SELECT [Наименование организации] FROM
[Юрлица] WHERE [Код клиента] = ?";
        using (OleDbCommand cmd = new
OleDbCommand(query, conn))
        {
            cmd.Parameters.Add("?", OleDbType.Integer).Value =
clientId;
            using (OleDbDataReader reader = cmd.ExecuteReader())
            {
                if (reader.Read())
                {
                    string orgName = reader.IsDBNull(0) ? "" :
reader.GetString(0);
                    return $"{orgName} (Юр.лицо)";
                }
            }
        }
    }
    catch (Exception ex)
    {
        System.Diagnostics.Debug.WriteLine($"Ошибка
получения информации о клиенте: {ex.Message}");
    }

    return null;
}

public class ComboBoxItem
{
    public int Value { get; set; }
    public string Text { get; set; }
    public string Code { get; set; }

    public override string ToString()
    {
        return Text;
    }
}

private void ComboBox_DrawItem(object sender,
DrawItemEventArgs e)
{
    if (e.Index < 0) return;
}

```

```

        ComboBox cb = sender as ComboBox;
        if (cb.Items.Count <= e.Index) return;

        object item = cb.Items[e.Index];
        ComboBoxItem comboItem = item as ComboBoxItem;

        if (comboItem == null)
        {
            e.DrawBackground();
            if ((e.State & DrawItemState.Selected) ==
                DrawItemState.Selected)
            {
                e.Graphics.DrawString(item.ToString(), cb.Font,
                    SystemBrushes.HighlightText, e.Bounds.Left + 2,
                    e.Bounds.Top + 2);
            }
            else
            {
                e.Graphics.DrawString(item.ToString(), cb.Font,
                    SystemBrushes.ControlText, e.Bounds.Left + 2,
                    e.Bounds.Top + 2);
            }
            e.DrawFocusRectangle();
            return;
        }

        e.DrawBackground();

        if ((e.State & DrawItemState.ComboBoxEdit) !=
            DrawItemState.ComboBoxEdit)
        {
            Font nameFont = new Font("Segoe UI", 11,
                FontStyle.Bold);
            Font codeFont = new Font("Segoe UI", 8,
                FontStyle.Regular);

            Color nameColor = Color.Black;
            Color codeColor = Color.Gray;

            if ((e.State & DrawItemState.Selected) ==
                DrawItemState.Selected)
            {
                nameColor = Color.White;
                codeColor = Color.LightGray;
            }

            using (SolidBrush nameBrush = new
                SolidBrush(nameColor))
            {
                e.Graphics.DrawString(comboItem.Text, nameFont,
                    nameBrush, e.Bounds.Left + 2, e.Bounds.Top + 2);
            }

            using (SolidBrush codeBrush = new
                SolidBrush(codeColor))
            {
                e.Graphics.DrawString(comboItem.Code, codeFont,
                    codeBrush, e.Bounds.Left + 2, e.Bounds.Top + 18);
            }
        }
        else
        {
            Font codeFont = new Font("Segoe UI", 10,
                FontStyle.Regular);
            Color codeColor = Color.Black;

            using (SolidBrush codeBrush = new
                SolidBrush(codeColor))
            {
                e.Graphics.DrawString(comboItem.Value.ToString(),
                    codeFont, codeBrush, e.Bounds.Left + 2, e.Bounds.Top + 8);
            }
        }

        e.DrawFocusRectangle();
    }

```

```

        private void ComboBox_SelectedIndexChanged(object sender,
            EventArgs e)
        {
            ComboBox cb = sender as ComboBox;
            if (cb != null && cb.SelectedItem is ComboBoxItem)
            {
                cb.Invalidate();
            }
        }

        private void Btn_Click(object sender, EventArgs e)
        {
            try
            {
                // Проверяем заполнение всех полей (кроме
                AutoNumber)
                List<string> emptyFields = new List<string>();
                int controlIndex = 0;

                for (int i = 0; i < columnNames.Count; i++)
                {
                    // AutoNumber поля не включаем в INSERT - они
                    генерируются автоматически
                    if (isPrimaryKey[i] && (columnTypes[i] == typeof(int) ||
                        columnTypes[i] == typeof(long)) && !isForeignKey[i])
                    {
                        continue; // Пропускаем AutoNumber поля
                    }

                    string columnName = columnNames[i];
                    Control control = inputControls[controlIndex];
                    controlIndex++;

                    bool isEmpty = false;

                    if (control is TextBox textBox)
                    {
                        string inputValue = textBox.Text.Trim();
                        if (string.IsNullOrEmpty(inputValue))
                        {
                            isEmpty = true;
                        }
                    }
                    else if (control is ComboBox comboBox)
                    {
                        if (comboBox.SelectedItem == null)
                        {
                            isEmpty = true;
                        }
                        else
                        {
                            string itemText =
                                comboBox.SelectedItem?.ToString() ?? "";
                            if (itemText == "Нет данных для выбора" ||
                                itemText == "Ошибка загрузки данных")
                            {
                                isEmpty = true;
                            }
                        }
                    }

                    if (isEmpty)
                    {
                        emptyFields.Add(columnName);
                    }
                }

                if (emptyFields.Count > 0)
                {
                    string message = "Все поля должны быть заполнены.
                    Следующие поля пусты:\n\n" +
                        string.Join("\n", emptyFields.Select(f => $"•
                    {f}")) +
                        "\n\nПожалуйста, заполните все поля перед
                    добавлением записи.";

                    MessageBox.Show(message, "Не все поля заполнены",

```

```

        MessageBoxButtons.OK,
        MessageBoxIcon.Warning);
        return;
    }

    // Добавляем запись
    using (OleDbConnection conn = new
OleDbConnection(connectionString))
    {
        conn.Open();

        var parameters = new Dictionary<string, object>();
        var columns = new List<string>();
        controlIndex = 0;

        for (int i = 0; i < columnNames.Count; i++)
        {
            string columnName = columnNames[i];

            // AutoNumber поля не включаем в INSERT
            if (isPrimaryKey[i] && (columnTypes[i] ==
typeof(int) || columnTypes[i] == typeof(long)) && !isForeignKey[i])
            {
                continue; // Пропускаем AutoNumber поля
            }

            Control control = inputControls[controlIndex];
            controlIndex++;

            bool allowsNull = isNullable[i];
            object value = null;

            if (control is TextBox textBox)
            {
                string inputValue = textBox.Text.Trim();
                if (!string.IsNullOrEmpty(inputValue))
                {
                    value = inputValue;
                }
                else if (allowsNull)
                {
                    value = DBNull.Value;
                }
            }
            else if (control is ComboBox comboBox)
            {
                if (comboBox.SelectedItem is ComboBoxItem
selectedItem)
                {
                    value = selectedItem.Value;
                }
                else if (allowsNull)
                {
                    value = DBNull.Value;
                }
            }

            if (value != null || value == DBNull.Value)
            {
                parameters[columnName] = value;
                columns.Add(columnName);
            }
        }

        string sql;

```

Файл TableEdit.cs

```

using System;
using System.Collections.Generic;
using System.Data;
using System.Data.OleDb;
using System.Drawing;
using System.Linq;
using System.Windows.Forms;

namespace KR
{
    public partial class TableEdit : Form

```

```

    {
        if (columns.Count > 0)
        {
            string columnsStr = string.Join(", ", columns.Select(c
=> $"[{c}]"));
            string valuesStr = string.Join(", ",
Enumerable.Repeat("?", columns.Count));
            sql = $"INSERT INTO [{tableName}] ({columnsStr})
VALUES ({valuesStr})";
        }
        else
        {
            sql = $"INSERT INTO [{tableName}] DEFAULT
VALUES";
        }

        using (OleDbCommand cmd = new OleDbCommand(sql,
conn))
        {
            foreach (var columnName in columns)
            {
                var value = parameters[columnName];
                int colIndex =
columnNames.IndexOf(columnName);

                if (isForeignKey[colIndex] && value !=
DBNull.Value)
                {
                    cmd.Parameters.Add("?",
OleDbType.Integer).Value = value;
                }
                else
                {
                    cmd.Parameters.AddWithValue("?", value);
                }
            }

            int result = cmd.ExecuteNonQuery();
            if (result > 0)
            {
                MessageBox.Show("Запись успешно добавлена!",
"Успех",
                MessageBoxButtons.OK,
                MessageBoxIcon.Information);
                this.DialogResult = DialogResult.OK;
                this.Close();
            }
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Ошибка при добавлении записи:
{ex.Message}", "Ошибка",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
    }

    private void label1_Click(object sender, EventArgs e)
    {
    }

    {
        private string connectionString;
        private string tableName;
        private DataRow selectedRow;
        private List<KeyValuePair<string, Control>> fields;
        private Dictionary<string, bool> readOnlyFields;
        private Dictionary<string, ComboBox>
foreignKeyComboBoxes;

        public TableEdit(string connectionString_param, string
tableName_param, DataRow selectedRow_param)
        {
            InitializeComponent();

```

```

        this.connectionString = connectionString_param ?? throw new
        ArgumentNullException(nameof(connectionString_param));
        this.tableName = tableName_param ?? throw new
        ArgumentNullException(nameof(tableName_param));
        this.selectedRow = selectedRow_param ?? throw new
        ArgumentNullException(nameof(selectedRow_param));
        fields = new List<KeyValuePair<string, Control>>();
        readOnlyFields = new Dictionary<string, bool>();
        foreignKeyComboBoxes = new Dictionary<string,
        ComboBox>();
        InitializeForm();
    }

    private void InitializeForm()
    {
        this.Text = $"Редактирование записи: {tableName}";
        this.Size = new Size(600, 500);
        this.FormBorderStyle = FormBorderStyle.FixedDialog;
        this.MaximizeBox = false;
        this.StartPosition = FormStartPosition.CenterParent;
        this.Font = new Font("Segoe UI", 10);

        FlowLayoutPanel fieldsPanel = new FlowLayoutPanel
        {
            Dock = DockStyle.Fill,
            AutoScroll = true,
            FlowDirection = FlowDirection.TopDown,
            WrapContents = false,
            Padding = new Padding(20)
        };

        Panel buttonsPanel = new Panel
        {
            Dock = DockStyle.Bottom,
            Height = 60,
            Padding = new Padding(20)
        };

        this.Controls.Add(fieldsPanel);
        this.Controls.Add(buttonsPanel);

        // Получаем информацию о внешних ключах
        Dictionary<string, ForeignKeyInfo> foreignKeys =
        GetForeignKeys();

        using (OleDbConnection connection = new
        OleDbConnection(connectionString))
        {
            connection.Open();

            DataTable schemaTable =
            connection.GetOleDbSchemaTable(OleDbSchemaGuid.Columns,
            new object[] { null, null, tableName, null });

            foreach (DataRow column in schemaTable.Rows)
            {
                string columnName =
                column["COLUMN_NAME"].ToString();
                bool isAutoIncrement =
                IsAutoIncrementColumn(column);
                bool isPrimaryKey =
                IsPrimaryKeyColumn(columnName);

                Panel rowPanel = new Panel
                {
                    Width = fieldsPanel.ClientSize.Width - 40,
                    Height = 40,
                    Margin = new Padding(0, 0, 0, 10)
                };

                Label label = new Label
                {
                    Text = columnName + ":",
                    Width = 150,
                    TextAlign = ContentAlignment.MiddleRight,
                    Location = new Point(0, 10),
                    Font = new Font("Segoe UI", 10)
                };
            }

```

```

        rowPanel.Controls.Add(label);

        Control inputControl;

        // Если это внешний ключ, создаем ComboBox
        if (foreignKeys.ContainsKey(columnName))
        {
            ComboBox comboBox = new ComboBox
            {
                Name = $"cbo_{columnName}",
                Location = new Point(160, 8),
                Width = 300,
                Font = new Font("Segoe UI", 10),
                DropDownStyle = ComboBoxStyle.DropDownList
            };

            LoadForeignKeyData(comboBox,
            foreignKeys[columnName]);

            // Устанавливаем текущее значение
            string currentValue =
            selectedRow[columnName]?.ToString();
            if (!string.IsNullOrEmpty(currentValue))
            {
                foreach (ComboBoxItem item in comboBox.Items)
                {
                    if (item.Value.ToString() == currentValue)
                    {
                        comboBox.SelectedItem = item;
                        break;
                    }
                }
            }

            inputControl = comboBox;
            foreignKeyComboBoxes[columnName] = comboBox;
        }
        else
        {
            // Обычное текстовое поле
            TextBox textBox = new TextBox
            {
                Name = $"txt_{columnName}",
                Location = new Point(160, 8),
                Width = 300,
                Font = new Font("Segoe UI", 10),
                Text = selectedRow[columnName]?.ToString() ??
                ""
            };

            // ИСПРАВЛЕННАЯ ЛОГИКА: Делаем поле
            // только для чтения только если это явно автоинкремент
            // И исключаем поля, которые должны быть
            // редактируемыми
            bool shouldBeReadOnly =
            ShouldFieldBeReadOnly(columnName, isAutoIncrement,
            isPrimaryKey);

            if (shouldBeReadOnly)
            {
                textBox.ReadOnly = true;
                textBox.BackColor = SystemColors.Control;
                readOnlyFields[columnName] = true;
            }
            else
            {
                readOnlyFields[columnName] = false;
            }

            inputControl = textBox;
        }

        rowPanel.Controls.Add(inputControl);
        fieldsPanel.Controls.Add(rowPanel);
        fields.Add(new KeyValuePair<string,
        Control>(columnName, inputControl));
    }
}

```

```

Button btnSave = new Button
{
    Text = "Сохранить",
    Width = 105,
    Height = 30,
    Font = new Font("Segoe UI", 10),
    Location = new Point(buttonsPanel.Width - 230, 15)
};
btnSave.Click += BtnSave_Click;
buttonsPanel.Controls.Add(btnSave);

Button btnCancel = new Button
{
    Text = "Отмена",
    Width = 100,
    Height = 30,
    Font = new Font("Segoe UI", 10),
    Location = new Point(buttonsPanel.Width - 120, 15)
};
btnCancel.Click += (s, e) => this.Close();
buttonsPanel.Controls.Add(btnCancel);
}

// ИСПРАВЛЕННЫЙ МЕТОД: Улучшенная логика
определения полей только для чтения
private bool ShouldFieldBeReadOnly(string columnName, bool
isAutoIncrement, bool isPrimaryKey)
{
    string columnNameLower = columnName.ToLower();

    // Поля, которые всегда должны быть редактируемыми
    (независимо от других условий)
    var alwaysEditableFields = new
HashSet<string>(StringComparer.OrdinalIgnoreCase)
{
    "дата", "date", "дата платежа", "дата платежа",
    "сумма", "amount", "sum", "сумма платежа",
    "описание", "description", "комментарий", "comment",
    "количество", "quantity", "count",
    "телефон", "phone", "номер телефона", "номер
телефона", "phone_number",
    "email", "почта", "электронная почта",
    "имя", "name", "фамилия", "отчество", "surname",
    "firstname", "lastname",
    "адрес", "address", "город", "city", "улица", "street",
    "должность", "position", "звание", "rank",
    "зарплата", "salary", "оклад", "wage",
    "стаж", "experience", "опыт", "years",
    "вес", "weight", "объем", "volume", "размер", "size"
};

    // Если поле содержит любое из всегда редактируемых
слов - разрешаем редактирование
    if (alwaysEditableFields.Any(field =>
columnNameLower.Contains(field.ToLower())))
    {
        return false;
    }

    // Поля, которые точно должны быть только для чтения
(только точные совпадения или четкие идентификаторы)
    var alwaysReadOnlyFields = new
HashSet<string>(StringComparer.OrdinalIgnoreCase)
{
    "id", "код", "code", "key", "ключ", "идентификатор",
    "identifier"
};

    // Проверяем точное совпадение с именами полей только
для чтения
    if (alwaysReadOnlyFields.Contains(columnName))
    {
        return true;
    }

    // Проверяем, начинается ли имя поля с id, код и т.д., но
исключаем составные имена

```

```

        if (columnNameLower.StartsWith("id") &&
columnNameLower.Length <= 4)
        {
            return true;
        }

        if (columnNameLower.StartsWith("код") &&
columnNameLower.Length <= 5)
        {
            return true;
        }

        // Блокируем только явно автоинкрементные поля
        return isAutoIncrement;
    }

    private Dictionary<string, ForeignKeyInfo> GetForeignKeys()
    {
        Dictionary<string, ForeignKeyInfo> foreignKeys = new
Dictionary<string, ForeignKeyInfo>();

        try
        {
            using (OleDbConnection connection = new
OleDbConnection(connectionString))
            {
                connection.Open();

                // Получаем информацию о внешних ключах
                DataTable foreignKeysTable =
connection.GetOleDbSchemaTable(OleDbSchemaGuid.Foreign_Ke
ys,
                    new object[] { null, null, null, null, null, tableName });

                foreach (DataRow row in foreignKeysTable.Rows)
                {
                    string fkColumnName =
row["FK_COLUMN_NAME"].ToString();
                    string pkTableName =
row["PK_TABLE_NAME"].ToString();
                    string pkColumnName =
row["PK_COLUMN_NAME"].ToString();

                    foreignKeys[fkColumnName] = new ForeignKeyInfo
                    {
                        ReferencedTable = pkTableName,
                        ReferencedColumn = pkColumnName
                    };
                }
            }
        }
        catch (Exception ex)
        {
            // Если не удастся получить информацию о внешних
ключах, продолжаем без них
            Console.WriteLine($"Не удалось получить информацию
о внешних ключах: {ex.Message}");
        }

        return foreignKeys;
    }

    private void LoadForeignKeyData(ComboBox comboBox,
ForeignKeyInfo fkInfo)
    {
        try
        {
            using (OleDbConnection connection = new
OleDbConnection(connectionString))
            {
                connection.Open();

                // Получаем первые два столбца из связанной таблицы
для отображения
                DataTable schemaTable =
connection.GetOleDbSchemaTable(OleDbSchemaGuid.Columns,
                    new object[] { null, null, fkInfo.ReferencedTable, null
});
            }
        }
    }

```

```

var columns = schemaTable.Rows.Cast<DataRow>()
    .OrderBy(r
        =>
        Convert.ToInt32(r["ORDINAL_POSITION"]))
    .Take(2)
    .Select(r => r["COLUMN_NAME"].ToString())
    .ToList();

string displayColumn = columns.Count > 1 ? columns[1]
: columns[0];

string query = $"SELECT [{fkInfo.ReferencedColumn}],
[{displayColumn}] FROM [{fkInfo.ReferencedTable}] ORDER BY
[{displayColumn}]";

OleDbCommand command = new
OleDbCommand(query, connection);
OleDbDataReader reader = command.ExecuteReader();

comboBox.Items.Clear();
comboBox.Items.Add(new ComboBoxItem("", "<Не
выбрано>"));

while (reader.Read())
{
    object keyValue = reader[fkInfo.ReferencedColumn];
    object displayValue = reader[displayColumn];

    comboBox.Items.Add(new
    ComboBoxItem(keyValue, displayValue?.ToString() ?? ""));
}

reader.Close();
}
catch (Exception ex)
{
    MessageBox.Show($"Ошибка при загрузке данных для
поля {comboBox.Name}: {ex.Message}",
        "Ошибка",
        MessageBoxButtons.OK,
        MessageBoxIcon.Warning);
}

private bool IsAutoIncrementColumn(DataRow columnInfo)
{
    try
    {
        return columnInfo["COLUMN_FLAGS"] != DBNull.Value
        &&
        (Convert.ToInt32(columnInfo["COLUMN_FLAGS"])
        & 0x10) != 0;
    }
    catch
    {
        return false;
    }
}

private bool IsPrimaryKeyColumn(string columnName)
{
    try
    {
        using (OleDbConnection connection = new
        OleDbConnection(connectionString))
        {
            connection.Open();
            DataTable primaryKeys =
            connection.GetOleDbSchemaTable(OleDbSchemaGuid.Primary_Keys,
            new object[] { null, null, tableName });

            foreach (DataRow row in primaryKeys.Rows)
            {
                if
                (row["COLUMN_NAME"].ToString().Equals(columnName,
                StringComparison.OrdinalIgnoreCase))
                {

```

```

return true;
        }
    }
}
catch
{
    // Если не удастся определить первичный ключ, НЕ
    считаем первый столбец первичным
    return false;
}

return false;
}

private void BtnSave_Click(object sender, EventArgs e)
{
    try
    {
        using (OleDbConnection connection = new
        OleDbConnection(connectionString))
        {
            connection.Open();

            List<string> setClauses = new List<string>();
            OleDbCommand command = new OleDbCommand();
            command.Connection = connection;

            // Получаем первичный ключ для WHERE условия
            string primaryKeyColumn = GetPrimaryKeyColumn();
            object primaryKeyValue =
            selectedRow[primaryKeyColumn];

            foreach (var pair in fields)
            {
                // Пропускаем поля только для чтения и первичный
                ключ
                if (readOnlyFields.ContainsKey(pair.Key) &&
                readOnlyFields[pair.Key])
                    continue;

                if (pair.Key.Equals(primaryKeyColumn,
                StringComparison.OrdinalIgnoreCase))
                    continue;

                setClauses.Add($"[{pair.Key}] = ?");

                object value;
                if (foreignKeyComboBoxes.ContainsKey(pair.Key))
                {
                    // Для внешних ключей берем значение из
                    ComboBox
                    ComboBoxItem selectedItem =
                    foreignKeyComboBoxes[pair.Key].SelectedItem as ComboBoxItem;
                    value = selectedItem?.Value ?? DBNull.Value;
                    if (value.ToString() == "")
                        value = DBNull.Value;
                }
                else
                {
                    // Для обычных полей
                    string textValue = ((TextBox)pair.Value).Text;
                    OleDbType paramType =
                    GetOleDbType(pair.Key);
                    value = ConvertParameterValue(textValue,
                    paramType);
                }

                command.Parameters.Add($"?[{pair.Key}]",
                GetOleDbType(pair.Key)).Value = value;
            }

            if (setClauses.Count == 0)
            {
                MessageBox.Show("Нет полей для обновления.",
                "Предупреждение",
                MessageBoxButtons.OK,
                MessageBoxIcon.Warning);
            }
        }
    }
}

```

```

        return;
    }

    // Добавляем параметр для WHERE условия
    command.Parameters.Add($"?{primaryKeyColumn}",
        GetOleDbType(primaryKeyColumn))
        .Value =
        ConvertParameterValue(primaryKeyValue?.ToString(),
            GetOleDbType(primaryKeyColumn));

    string query = $"UPDATE [{tableName}] SET
    {string.Join(", ", setClauses)} WHERE [{primaryKeyColumn}] = ?";
    command.CommandText = query;

    int rowsAffected = command.ExecuteNonQuery();
    if (rowsAffected > 0)
    {
        MessageBox.Show("Запись успешно обновлена.",
            "Успех",
            MessageBoxButtons.OK,
            MessageBoxIcon.Information);
        this.DialogResult = DialogResult.OK;
        this.Close();
    }
    else
    {
        MessageBox.Show("Не удалось обновить запись.",
            "Ошибка",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
catch (Exception ex)
{
    MessageBox.Show("Ошибка при обновлении: " +
        ex.Message, "Ошибка",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
}

private string GetPrimaryKeyColumn()
{
    try
    {
        using (OleDbConnection connection = new
            OleDbConnection(connectionString))
        {
            connection.Open();
            DataTable primaryKeys =
            connection.GetOleDbSchemaTable(OleDbSchemaGuid.Primary_Keys,
                new object[] { null, null, tableName });

            if (primaryKeys.Rows.Count > 0)
            {
                return
                primaryKeys.Rows[0]["COLUMN_NAME"].ToString();
            }
        }
    }
    catch
    {
        // Если не удастся определить первичный ключ,
        используем первый столбец
    }

    return fields.Count > 0 ? fields[0].Key : "";
}

private OleDbType GetOleDbType(string columnName)
{
    using (OleDbConnection connection = new
        OleDbConnection(connectionString))
    {
        connection.Open();
        DataTable schemaTable =
        connection.GetOleDbSchemaTable(OleDbSchemaGuid.Columns,
            new object[] { null, null, tableName, null });

```

```

        foreach (DataRow row in schemaTable.Rows)
        {
            if
            (row["COLUMN_NAME"].ToString().Equals(columnName,
                StringComparison.OrdinalIgnoreCase))
            {
                int dataTypeCode =
                Convert.ToInt32(row["DATA_TYPE"]);
                return (OleDbType)dataTypeCode;
            }
        }
        return OleDbType.VarChar;
    }

    private object ConvertParameterValue(string value, OleDbType
        type)
    {
        if (string.IsNullOrEmpty(value) && type !=
            OleDbType.VarChar && type != OleDbType.LongVarChar)
            return DBNull.Value;

        switch (type)
        {
            case OleDbType.Integer:
            case OleDbType.BigInt:
                return int.TryParse(value, out int intValue) ? intValue :
                    DBNull.Value;
            case OleDbType.Double:
            case OleDbType.Numeric:
                return double.TryParse(value, out double doubleValue) ?
                    doubleValue : DBNull.Value;
            case OleDbType.Date:
                return DateTime.TryParse(value, out DateTime
                    dateValue) ? dateValue : DBNull.Value;
            case OleDbType.Boolean:
                return bool.TryParse(value, out bool boolValue) ?
                    boolValue : DBNull.Value;
            case OleDbType.Guid:
                return Guid.TryParse(value, out Guid guidValue) ?
                    guidValue : DBNull.Value;
            default:
                return value;
        }
    }

    private void InitializeComponent()
    {
        SuspendLayout();
        //
        // TableEdit
        //
        AutoScaleDimensions = new.SizeF(8F, 20F);
        AutoScaleMode = AutoScaleMode.Font;
        ClientSize = new.Size(495, 282);
        Name = "TableEdit";
        Text = "Редактирование записи";
        ResumeLayout(false);
    }

    private void label1_Click(object sender, EventArgs e)
    {
    }

    // Вспомогательные классы
    public class ForeignKeyInfo
    {
        public string ReferencedTable { get; set; }
        public string ReferencedColumn { get; set; }
    }

    public class ComboBoxItem
    {
        public object Value { get; set; }
        public string Display { get; set; }
    }

```

```
public ComboBoxItem(object value, string display)
{
    Value = value;
    Display = display;
}
```

```
public override string ToString()
{
    return Display;
}
}
```

Руководство пользователя

Программа предназначена для работы с базой данных кинотеатра. С её помощью пользователь может эффективно управлять данными: добавлять новые записи о фильмах, сеансах, билетах, зрителях и сотрудниках, вносить изменения в существующие сведения и удалять устаревшую информацию. Также предусмотрена возможность выполнения 30 предустановленных запросов для получения аналитической и справочной информации, включая расписание сеансов, статистику посещаемости и продажи билетов.

Для запуска приложения необходимо дважды щёлкнуть по файлу CinemaApp.exe (рис. ПЗ.1). Файл базы данных находится в одной папке с программой.

 CinemaApp	01.10.2025 18:09	Приложение	136 КБ
 CinemaDB	01.10.2025 19:48	Microsoft Access ...	1 120 КБ

Рис. ПЗ.1. Запуск приложения

После запуска программы открывается главное окно, где центральная часть предназначена для отображения данных, а справа расположена панель управления (рис. ПЗ.2). В нижнем блоке отображается справочная информация, содержащая подсказки и описание выполняемых действий. Чтобы выбрать таблицу базы данных, необходимо воспользоваться выпадающим списком «Выбрать таблицу», расположенным на панели управления (рис. ПЗ.3). Аналогично, для выполнения запроса используется список «Выбрать запрос», позволяющий получить результаты предустановленных запросов (рис. ПЗ.4). Для редактирования данных предусмотрены кнопки «Добавить запись», «Удалить запись» и «Изменить запись», а вызов окна со справочной информацией осуществляется через кнопку «HELP» (рис. ПЗ.5).



Рис. ПЗ.2 Интерфейс программы

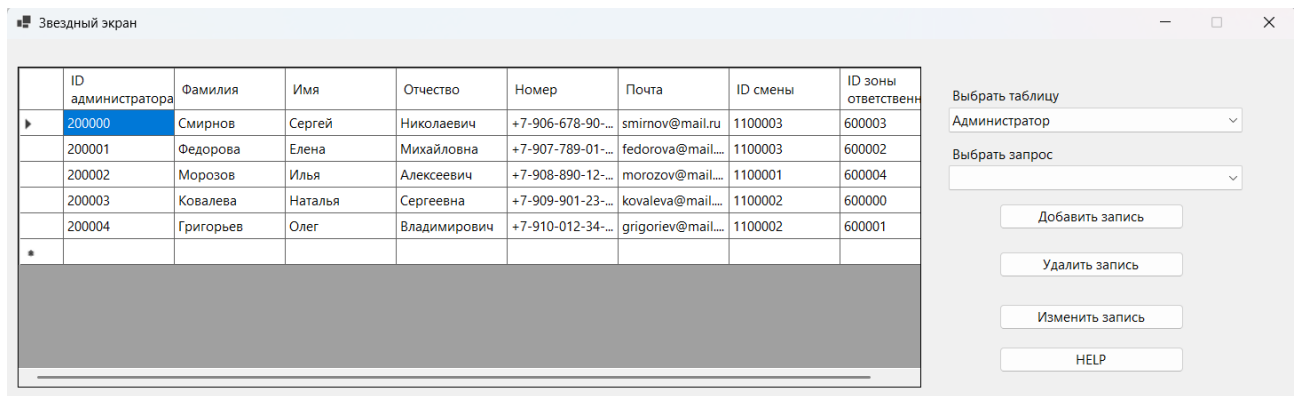


Рис. ПЗ.3 Выбор таблицы

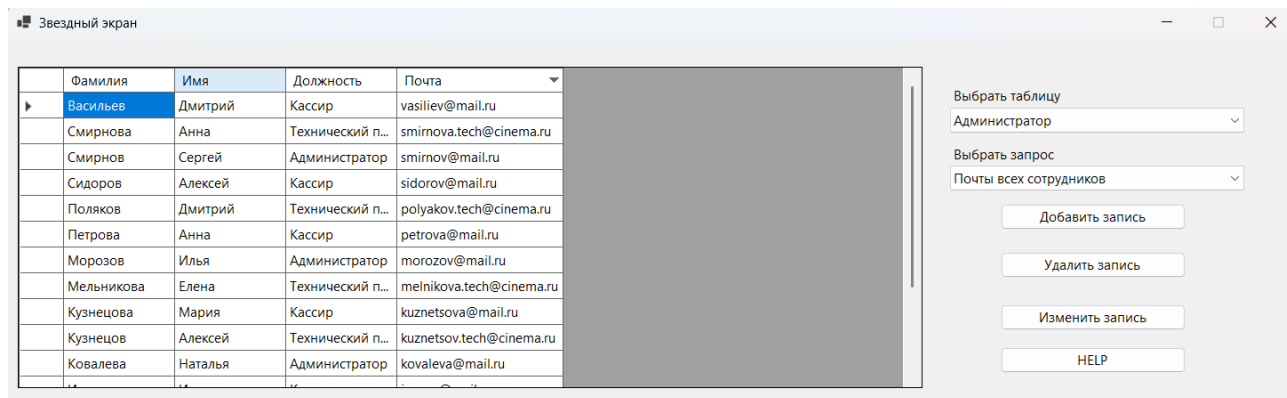


Рис. ПЗ.4. Выбор запроса

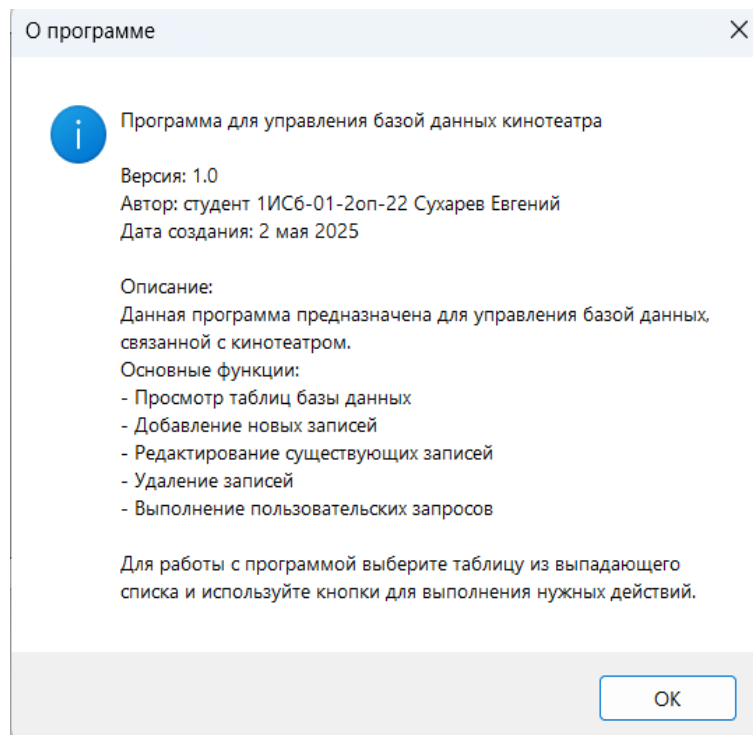


Рис. ПЗ.5. Окно «HELP»

Если необходимо внести изменения в таблицу, воспользуйтесь кнопками управления, расположенными ниже основной области с данными. Чтобы

добавить новую запись, нажмите кнопку «Добавить запись» - откроется форма для ввода информации (рис. ПЗ.6). В ней нужно заполнить все поля. Идентификатор записи присваивается автоматически, а значения, связанные с другими таблицами, выбираются из выпадающего списка. После ввода данных нажмите кнопку «Добавить». Для редактирования существующей записи выделите нужную строку в таблице (рис. ПЗ.7) и нажмите кнопку «Изменить запись». Откроется окно, где можно внести необходимые изменения (рис. ПЗ.8). После этого нажмите кнопку «Сохранить» или, при отказе от редактирования, кнопку «Отмена». Чтобы удалить запись, выделите нужную строку и нажмите кнопку «Удалить» (рис. ПЗ.9).

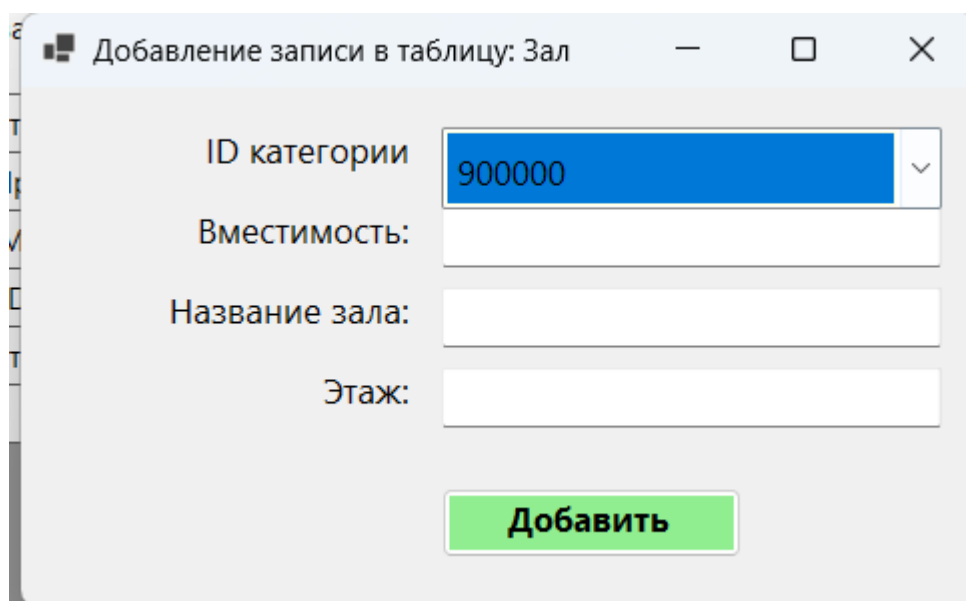


Рис. ПЗ.6. Окно добавления записи

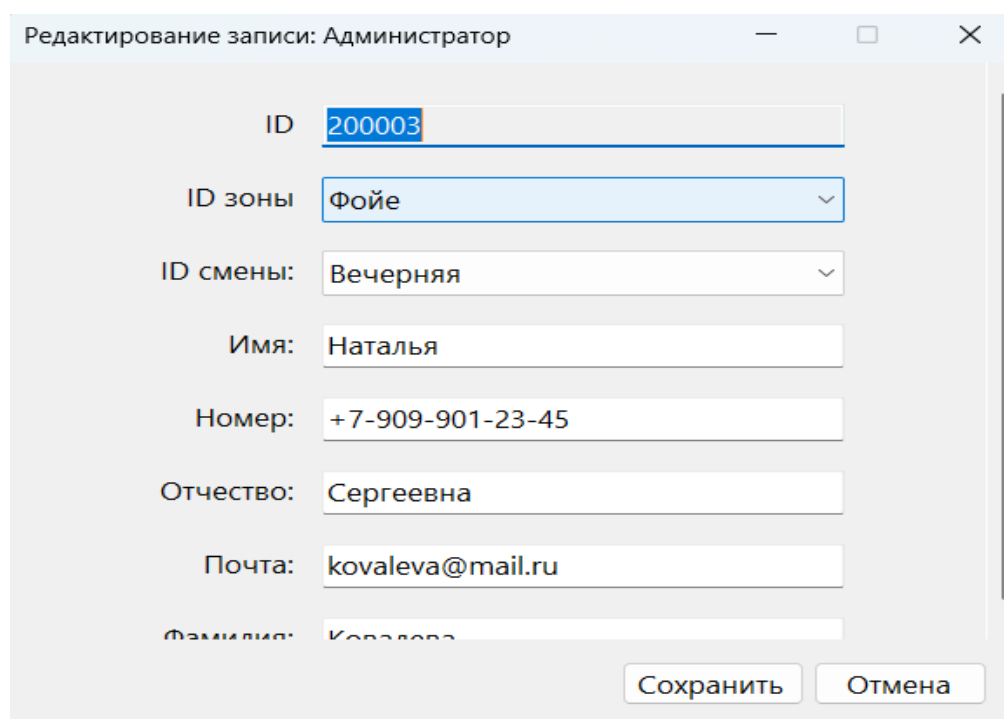


Рис. ПЗ.6. Окно изменения записи

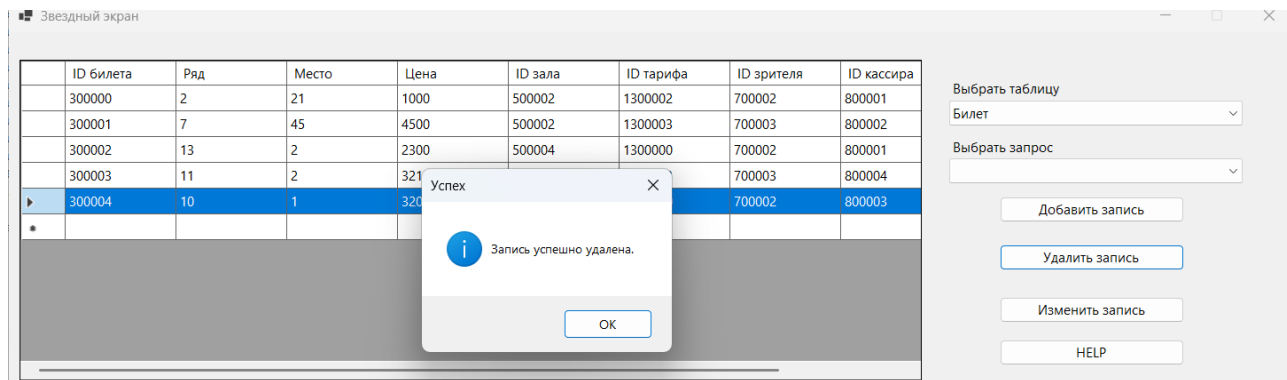


Рис. ПЗ.6. Окно удаления записи

Для закрытия программы нужно в главном окне в правом верхнем углу нажать на «X» (рис. ПЗ.10).