МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное

учреждение высшего образования

«Казанский национальный исследовательский технический

университет им. А.Н. Туполева-КАИ»

(КНИТУ-КАИ)

|  |
| --- |
| Отделение СПО ИКТЗИ, КИТ |

(наименование института (факультета)

**КУРСОВАЯ РАБОТА**

по дисциплине «Технология разработки и защиты баз данных»

тема «Учёт заявок на выполнение услуг рекламного агентства»

|  |  |  |  |
| --- | --- | --- | --- |
| Выполнила: | обучающийся группы 4238 | |  |
|  | Бусов В.Р. | |  |
|  | (ФИО) | | (подпись) |
| Проверил: | преподаватель | Григорьева И.В. |  |
|  | (должность) | (Ф.И.О.) | (подпись) |
| Курсовая работа защищена с оценкой: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | |
| Дата защиты «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г. | | | |

Казань 2024г.

**АННОТАЦИЯ**

Курсовая работа 65 страниц, 52 рисунка, 5 таблиц, 7 источников литературы.

В данном курсовом проекте будет спроектирована база данных для учёта заявок на выполнение услуг рекламного агентства.

Для создания базы данных была использована стандартная среда администрирования баз данных СУБД MS SQL Server – Среда администрирования MS SQL Server Management.

База данных «Учёт заявок на выполнение услуг рекламного агентства» включает таблицы, представления, триггеры и хранимые процедуры.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc168338111)

[ГЛАВА 1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 5](#_Toc168338112)

[1.1. Концептуальное проектирование 5](#_Toc168338113)

[1.2. База данных 8](#_Toc168338114)

[1.3. Модели баз данных 9](#_Toc168338115)

[1.3.1 Реляционная модель данных 9](#_Toc168338116)

[1.4. Типы данных MySQL 9](#_Toc168338117)

[1.5. Обоснование выбора СУБД 11](#_Toc168338118)

[ГЛАВА 2. РАБОТА С БАЗОЙ ДАННЫХ 13](#_Toc168338119)

[2.1. Даталогическое проектирование 13](#_Toc168338120)

[2.1.1. Преобразование концептуальной модели в реляционную модель 13](#_Toc168338121)

[2.1.2. Нормализация базы данных 14](#_Toc168338122)

[2.1.3. Определение характеристик атрибутов 15](#_Toc168338123)

[2.2. Создание БД в выбранной СУБД 17](#_Toc168338124)

[2.3. Импорт и экспорт данных 19](#_Toc168338125)

[2.4. Поддержка целостности данных 24](#_Toc168338126)

[2.4.1. Декларативная поддержка ограничений целостности 24](#_Toc168338127)

[2.4.2. Процедурная поддержка ограничений целостности 24](#_Toc168338128)

[2.5. Реализация доступа пользователей к базе данных 34](#_Toc168338129)

[ГЛАВА 3. СОЗДАНИЕ WPF ПРИЛОЖЕНИЯ И ПОДКЛЮЧЕНИЕ БАЗЫ ДАННЫХ... 38](#_Toc168338130)

[ЗАКЛЮЧЕНИЕ 43](#_Toc168338131)

[СПИСОК ЛИТЕРАТУРЫ 44](#_Toc168338132)

[ПРИЛОЖЕНИЕ А 45](#_Toc168338133)

[ПРИЛОЖЕНИЕ Б 58](#_Toc168338134)

# **ВВЕДЕНИЕ**

БД «Учёт заявок на выполнение услуг рекламного агентства» является важным инструментом в сфере рекламы, направленным на организацию эффективной работы исполнителей и клиентов в рекламной деятельности. В ней объединяется информация о заявках, клиентах, исполнителях, категориях и истории выполнения заявок, необходимая для успешной работы и выполнения требований клиентов.

В данном курсовом проекте спроектируем базу данных для хранения информации о заявках, клиентах, исполнителях, категориях и истории выполнения заявок.

**Объектом исследования** является процесс разработки базы данных, а предметом курсовой работы является разработка БД «Учёт заявок на выполнение услуг рекламного агентства».

**Цель курсовой работы** – создание БД «Учёт заявок на выполнение услуг рекламного агентства».

Для достижения поставленной цели необходимо выполнить следующие **задачи**:

1. Ознакомиться с литературой по данной теме.
2. Проанализировать предметную область.
3. Построить ER-модель.
4. Создать объекты БД (таблицы, запросы).
5. Разработать триггеры и хранимые процедуры.

БД «Учёт заявок на выполнение услуг рекламного агентства» разработана с целью повышения эффективности выполнения заявок в рекламной деятельности, улучшения контроля над данными о клиентах и исполнителях, а также для улучшения коммуникации и сотрудничества при выполнении заказов.

**ГЛАВА 1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

* 1. **Концептуальное проектирование**

Выполнение услуг рекламного агентства — это специализированная деятельность, направленная на поддержку малоизвестных проектов и распространение информации о них путем размещения рекламы о данных проектах. В данной предметной области основными аспектами являются планирование и организация размещения рекламы, учет заявок, а также удобное и комфортное взаимодействие клиентов и исполнителей.

Учет выполнения услуг рекламного агентства включает в себя сотрудничество с организациями, размещающими рекламу, размещение рекламы, а также координацию работы исполнителей. Исполнитель отвечает за обсуждение рекламы, предлагает идеи по привлечению клиентов и за размещение рекламы.

Учет клиентов включает в себя сбор и систематизацию информации о каждом клиенте, такую как контактные данные (фамилия, имя, отчество, номер телефона и адрес электронной почты), категория рекламы и описание желаемого результата. Эта информация позволяет исполнителю проанализировать требования клиента и в полной степени выполнить работу, предложив подходящее решение.

Анализ эффективности проведенных рекламных кампаний позволяет руководителю агентства выявлять успешные стратегии работы с клиентами, что способствует улучшению качества предоставляемых услуг. Кроме того, такой анализ помогает выявить потенциальные проблемы и помогать клиентам в их решении.

Взаимодействие с сотрудниками и заказчиками включает в себя участие в совещаниях и совместных мероприятиях, обмен информацией о рекламных проектах и требованиях клиентов, а также согласование действий по выполнению рекламных кампаний.

Для автоматизации работы «Учет заявок на выполнение услуг рекламного агентства» предусматривается создание информационной системы, которая включает в себя базу данных с информацией о клиентах, выполненных проектах, рекламных компаниях и других данных. Программа должна обеспечивать возможность учета клиентов, формирование отчетов о выполненных проектах, анализ эффективности рекламных кампаний, а также обеспечивать взаимодействие с сотрудниками и заказчиками.

Система должна поддерживать следующие функции:

1. Учет клиентов: фамилия, имя, отчество, номер телефона, адрес электронной почты, заявки клиента.

2. Учет исполнителей: фамилия, имя, отчество, адрес электронной почты, выполненные заявки, должность в предприятии.

3. Учет заявок: название заявки, описание, к какому клиенту относится, время размещения рекламы, категория.

4. Категории рекламы: название категории, минимальная стоимость размещения рекламы для данной категории, описание, возможные места размещения рекламы

Программа также должна предоставлять возможность генерации отчетов, таких как:

1. Отчеты по выполненным заявкам: ФИО исполнителя, название и описание заявки.

2. Статистика по количеству обращений: категория заявки, количество обращений.

3. Отчеты для администрации: статистика по времени реализации одной заявки.

Внедрение автоматизированной системы «Учёт заявок на выполнение услуг рекламного агентства» позволит улучшить эффективность размещения рекламы, обеспечив точный учет и анализ деятельности, а также улучшит взаимодействие исполнителей и клиентов в рамках.

На основе проведённого анализа предметной области были выделены следующие сущности:

* *Клиент – хранение сведений о клиенте*
* *Исполнитель – хранение сведений об исполнителе*
* *Заявка – информация о рекламной заявке*
* *Категория – информация о категории рекламы*
* *История – информация об исполнителе и выполненных им заявках*

В таблице 1 представлен список атрибутов каждой сущности.

Таблица 1 – Перечень атрибутов выделенных сущностей

|  |  |  |
| --- | --- | --- |
| Сущность | Атрибут | Описание |
| 1 | 2 | 3 |
| Клиент | Id\_Клиента | Уникальный код клиента |
| Фамилия | Фамилия клиента |
| Имя | Имя клиента |
| Отчество | Отчество клиента |
| Номер телефона | Номер телефона клиента для связи |
| Адрес электронной почты | Адрес электронной почты клиента для связи |
| Исполнитель | ID\_Исполнителя | Уникальный код исполнителя |
| Фамилия | Фамилия исполнителя |
| Имя | Имя исполнителя |
| Отчество | Отчество исполнителя |
| Должность | Должность исполнителя в предприятии |
| Адрес электронной почты | Адрес электронной почты исполнителя для связи |
| Ключ электронной почты | Ключ внешнего приложения |
| Пароль | Пароль исполнителя |
| Категория | ID\_Категории | Уникальный код категории |
| Название | Название категории |
| Описание | Описание категории |
| Стоимость | Минимальная стоимость рекламы в данной категории |
| История | ID\_Исполнителя | Уникальный код исполнителя |
| ID\_Заявки | Уникальный код заявки |
| Заявка | ID\_Заявки | Уникальный код заявки |
| Название | Название заявки |
| Описание | Описание заявки |
| ID\_Категории | Код категории |
| ID\_Клиента | Код заказчика |

Продолжение таблицы 1

|  |  |  |
| --- | --- | --- |
|  | Дата открытия | Дата взятия в работу исполнителем |
| Дата закрытия | Дата завершения работы |

На рисунке 1 представлена ER-диаграмма «сущность-связь».

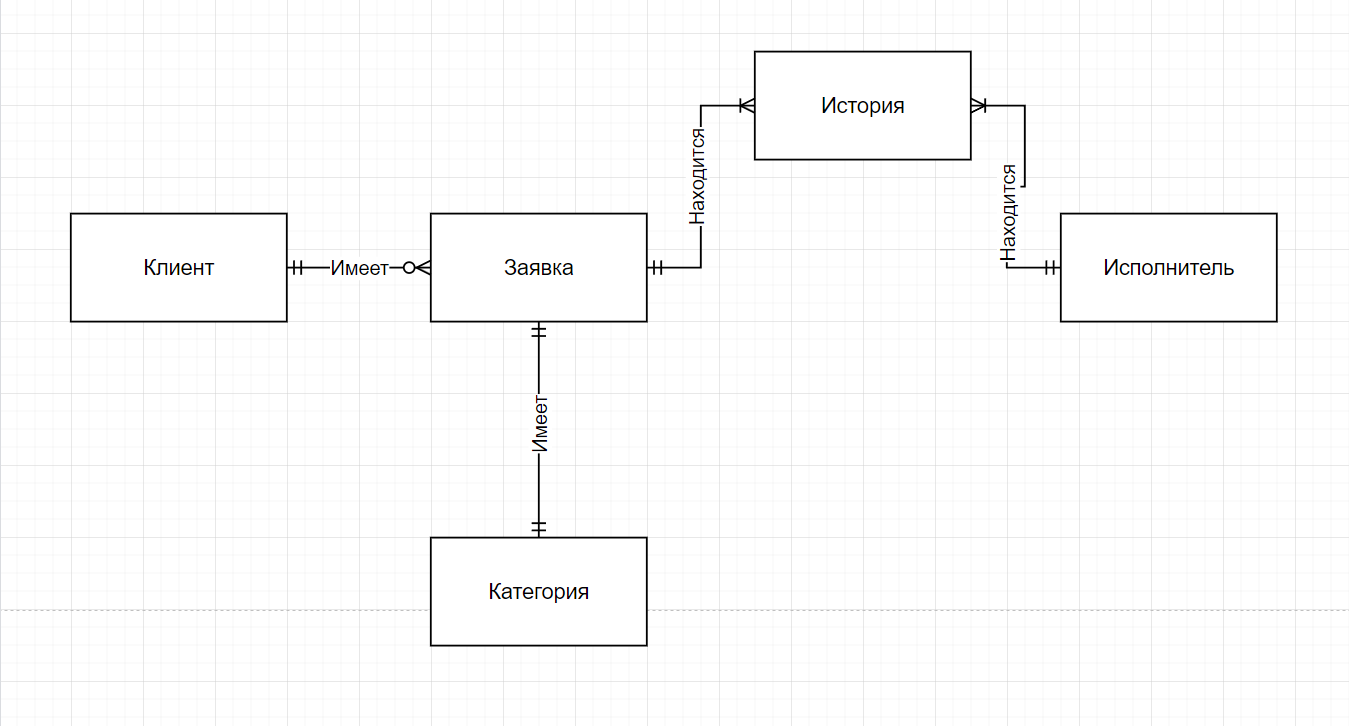


Рисунок 1 – ER-диаграмма «Сущность – связь»

* 1. **База данных**

В наше время данные играют важнейшую роль. Их настолько большое количество, что мы просто обязаны где-то хранить их. Один из способов – это базы данных.

База данных (БД) – это совокупность взаимосвязанных данных, которые хранятся вместе. Чтобы создать её и эксплуатировать нужны системы управления базами данных (СУБД).

СУБД – это совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

Каждая база данных состоит из: сущностей, их свойств (атрибутов) и взаимосвязей.

Сущности – это объект (таблица), в котором мы должны хранить данные.

* 1. **Модели баз данных**

При создании базы данных, необходимо упорядочить информацию по различным признакам, чтобы потом извлекать из нее необходимые данные в любом сочетании. Это возможно, только когда все данные структурированы.

Существует несколько видов моделей БД:

* Табличные (реляционные)
* Иерархические
* Сетевые
  + 1. **Реляционная модель данных**

Совокупность данных, состоящая из набора двумерных таблиц. В теории множеств таблице соответствует термин отношение (relation), физическим представлением которого является таблица, отсюда и название модели – реляционная. Эта структура должна обязательно иметь такие объекты, как отношение, атрибут, домен, кортеж, степень, кардинальность, первичный ключ. На рисунке 2 показаны все эти понятия.

****

Рисунок 2 – Реляционная модель данных

* 1. **Типы данных MySQL**

MySQL предоставляет следующие типы данных, которые можно разбить на ряд групп, в которых я выделю основные.

**Символьные типы:**

* **CHAR**: представляет строку фиксированной длины.
* **VARCHAR**: представляет стоку переменной длины.

**Числовые типы:**

* **TINYINT**: представляет целые числа от -127 до 128, занимает 1 байт
* **BOOL**: фактически не представляет отдельный тип, а является лишь псевдонимом для типа TINYINT(1) и может хранить два значения 0 и 1. Однако данный тип может также в качестве значения принимать встроенные константы **TRUE** (представляет число 1) и **FALSE** (предоставляет число 0).

Также имеет псевдоним **BOOLEAN**.

* **TINYINT UNSIGNED**: представляет целые числа от 0 до 255, занимает 1 байт
* **INT**: представляет целые числа от -2147483648 до 2147483647, занимает 4 байта
* **BIGINT**: представляет целые числа от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807, занимает 8 байт
* **DECIMAL**: хранит числа с фиксированной точностью. Данный тип может принимать два параметра precision и scale: DECIMAL (precision, scale).
* **FLOAT**: хранит дробные числа с плавающей точкой одинарной точности от -3.4028 \* 1038 до 3.4028 \* 1038, занимает 4 байта
* **DOUBLE**: хранит дробные числа с плавающей точкой двойной точности от -1.7976 \* 10308 до 1.7976 \* 10308, занимает 8 байт. Также может принимать форму DOUBLE (M, D), где M - общее количество цифр, а D - количество цифр после запятой.

**Типы для работы с датой и временем:**

* **DATE**: хранит даты с 1 января 1000 года до 31 декабря 9999 года (c "1000-01-01" до "9999-12-31"). По умолчанию для хранения используется формат yyyy-mm-dd. Занимает 3 байта.
* **TIME**: хранит время от -838:59:59 до 838:59:59. По умолчанию для хранения времени применяется формат «hh:mm:ss». Занимает 3 байта.

**Денежный тип:**

* **MONEY** – денежная величина от -263 (-922,337,203,685,477.5808) до 263-1 (+922,337,203,685,477.5807), с точностью до десятитысячной части денежной единицы.
  1. **Обоснование выбора СУБД**

СУБД (система управления базами данных) – это информационная среда для создания баз данных и управления хранимыми в них сведениями об объектах реального мира.

Выбор СУБД определяется многими факторами, но главный из них – возможность работы с построенной моделью данных. Поэтому одной из важнейших характеристик является тип модели (иерархический, сетевой, реляционный), который поддерживается СУБД. База данных для решения поставленной задачи является реляционной. Помимо модели данных важным показателем является стоимость лицензии для разработки базы данных и стоимость поддержки данной СУБД. Также необходимо обращать внимание на минимальные технические требования для оперативной и эффективной работы СУБД.

В настоящее время на рынке представлен огромный выбор СУБД и каждая СУБД «заточена» под определённый круг задач, которые она может выполнять. Проведём сравнительный анализ СУБД для разработки информационной системы складского учёта.

Для сравнения выберем такие СУБД, как:

* MS Access;
* MySQL;
* MS SQL Server.

Сравнительный анализ СУБД представлен в таблице 2.

Таблица 2 – Сравнение СУБД

|  |  |  |  |
| --- | --- | --- | --- |
| Критерии оценки | MS Access | MySQL | MS SQL Server |
| 1 | 2 | 3 | 4 |
| Размер базы данных | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| несколько Мб | + | + |  |

Продолжение таблицы 2

|  |  |  |  |
| --- | --- | --- | --- |
| до 100Мб | + | + |  |
| несколько ГБ |  | + | + |
| 100Гб и более |  |  | + |
| Количество одновременных пользователей, работающих с базой данных напрямую | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| 1 пользователь | + | + |  |
| до 10 | + | + |  |
| более 100 |  | + | + |
| более 1000 |  |  | + |
| Стоимость лицензии | | | |
| бесплатно (только лицензия на пакет) | + | + |  |
| дорого |  |  | + |
| Платформа, на которой способна работать СУБД | | | |
| Windows | + |  | + |
| Windows+Linux |  | + |  |
| Кластеры |  |  | + |
| Встроенный язык программирования | | | |
| Microsoft | + |  | + |
| Borland | + |  | + |
| Языки для Linux |  |  | + |
| Защита данных | | | |
| слабая | + |  |  |
| сильная |  | + | + |
| Мощность языка SQL и возможности СУБД | | | |
| слабые | + |  |  |
| развитые |  | + |  |
| мощные |  |  | + |
| Требования к техническим характеристикам ПК | | | |
| неприхотливые | + | + |  |
| необходимы мощные сервера |  |  | + |

Окончательный выбор был сделан в пользу MS SQL Server, так как данная СУБД обладает высокими показателями производительности, способна хранить большие объёмы информации без потери быстродействия при реализации доступа к данным из внешних источников.

**ГЛАВА 2. РАБОТА С БАЗОЙ ДАННЫХ**

# **2.1.** **Даталогическое проектирование**

# **2.1.1. Преобразование концептуальной модели в реляционную модель**

При переходе от абстрактной модели предметной области к схеме базы данных, существует несколько подходов.

Один из них предусматривает ручное выполнение преобразований, когда каждый шаг осуществляется вручную.

Другой подход более автоматизированный и заключается в компиляции абстрактной модели в реляционную схему, что позволяет создать базу данных, структурированную согласно третьей нормальной форме.

В рамках курсового проекта будем использовать первый подход. При этом сущности модели «сущность-связь» соответствует отношение (таблица) реляционной БД. Связи между сущностями типа «один – к – одному (1:1)» и «один – ко – многим (1:М)» соответствуют связям между отношениями. Атрибуты, которые идентифицируют, определяют или моделируют сущности в модели «сущность-связь» соответствуют атрибутам отношений.

На рисунке 3 представлена схема функциональных связей.

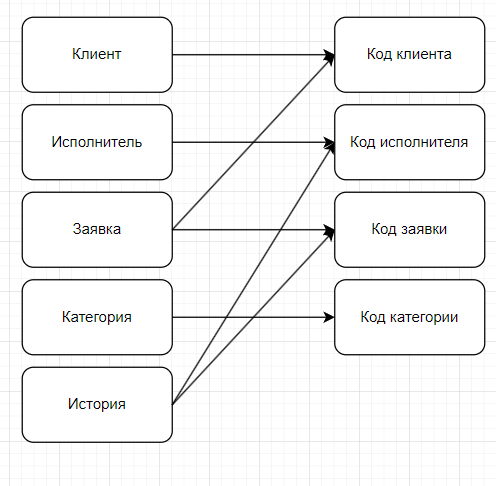


Рисунок 3 – Схема функциональных связей

При разработке реляционной базы данных «Учёт заявок на выполнение услуг рекламного агентства» проанализированы функциональные зависимости между атрибутами всех отношений и выбраны идентифицирующие атрибуты (первичные ключи):

– ID\_Клиента.

– ID\_Исполнителя.

– ID\_Заявки.

– ID\_Категории.

# **2.1.2. Нормализация базы данных**

Изначально все рассматриваемые отношения в реляционном подходе находятся в 1НФ, так как элементы доменов отношений не является множествами (атомарными).

Отношения находятся во 2НФ, так как отношения находятся в 1НФ, все таблицы имеют первичный ключ и каждый не первичный атрибут формально полно зависит от каждого значения первичного ключа.

Отношения находятся в 3НФ, если оно находиться в 2НФ и в нем нет зависимости атрибутов от возможных ключей.

Таблица 3 – 3НФ данных

|  |  |  |  |
| --- | --- | --- | --- |
| Клиент | Исполнитель | Заявка | Категория |
| ID\_Клиента | ID\_Исполнителя | ID\_Заявки | ID\_Категории |
| Фамилия | Фамилия | Название | Название |
| Имя | Имя | Описание | Описание |
| Отчество | Отчество | ID\_Категории | Стоимость |
| Номер телефона | Электронная почта | Дата создания |  |
| Электронная почта | Ключ электронной почты | Дата закрытия |  |
|  | Должность | ID\_Клиента |  |
|  | Пароль |  |  |
| История |  |  |  |
| ID\_Заявки |  |  |  |
| ID\_Исполнителя |  |  |  |

Теперь между неключевыми столбцами всех таблиц отсутствует транзитивная зависимость, что позволяет сделать вывод о том, что база данных «Учёт заявок на выполнение услуг рекламного агентства» приведена к 3НФ.

На рисунке 4 представлена логическая модель базы данных, отражающая связи таблиц БД, разработанная с помощью «drawio».

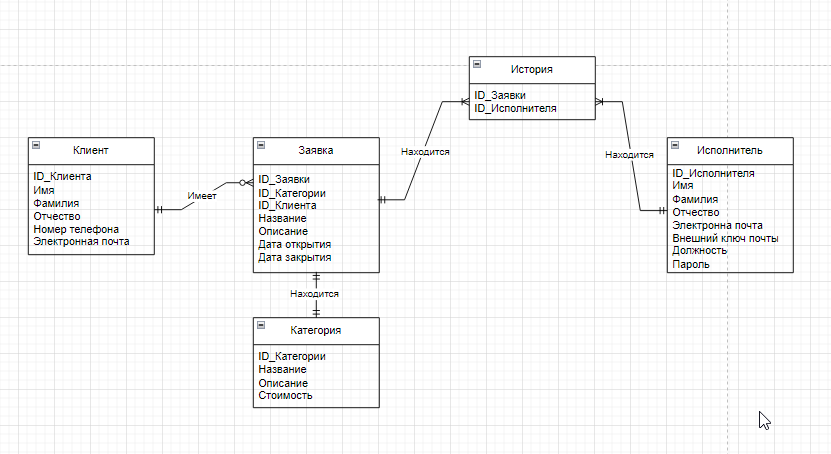


Рисунок 4 – Логическая модель базы данных

# **2.1.3. Определение характеристик атрибутов**

Описание даталогической модели БД представлено в таблице 4.

Таблица 4 – Описание даталогической модели БД

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблица | Поле | Тип поля | Размер поля | Допустимость  неопределенных значений |
| Клиент | Id\_Клиента | Счетчик |  | NOT NULL |
| Фамилия | TEXT | 100 | NOT NULL |
| Имя | TEXT | 100 | NOT NULL |
| Отчество | TEXT | 100 | NOT NULL |
| Номер телефона | TEXT | 11 | NOT NULL |
| Адрес электронной почты | TEXT | 100 | NOT NULL |
| Исполнитель | ID\_Исполнителя | Счетчик |  | NOT NULL |
| Фамилия | TEXT | 100 | NOT NULL |
| Имя | TEXT | 100 | NOT NULL |
| Отчество | TEXT | 100 | NOT NULL |
| Должность | TEXT | 50 | NOT NULL |
| Адрес электронной почты | TEXT | 100 | NOT NULL |
| Ключ электронной почты | TEXT | 100 | NOT NULL |
| Пароль | TEXT | 100 | NOT NULL |
| Заявка | ID\_Заявки | Счетчик |  | NOT NULL |
| Название | TEXT | 100 | NOT NULL |

Продолжение таблицы 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Описание | TEXT | 255 | NOT NULL |
| ID\_Категории | Счетчик |  | NOT NULL |
| ID\_Клиента | Счетчик |  | NOT NULL |
| Дата открытия | DATE |  |  |
| Дата закрытия | DATE |  |  |
| Категория | ID\_Категории | Счетчик |  | NOT NULL |
| Название | TEXT | 100 | NOT NULL |
| Описание | TEXT | 255 | NOT NULL |
| Стоимость | INT |  | NOT NULL |
| История | ID\_Исполнителя | INT |  | NOT NULL |
| ID\_Заявки | INT |  | NOT NULL |

На рисунке 5 представлена даталогическая модель БД.

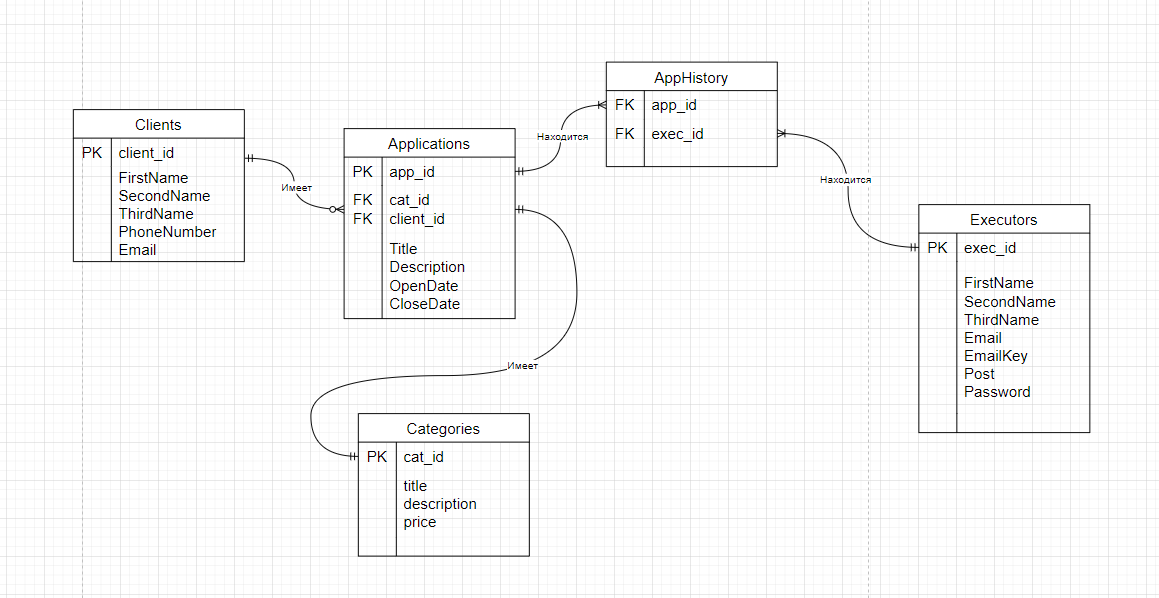


Рисунок 5 – Даталогическая модель БД

# **2.2. Создание БД в выбранной СУБД**

Для создания базы данных была использована стандартная среда администрирования баз данных выбранной СУБД MS SQL Server – Среда администрирования MS SQL Server Management.

На рисунках 6-10 представлена структура созданных таблиц БД.

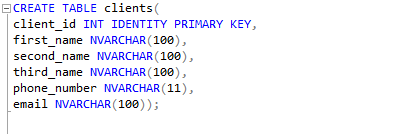


Рисунок 6 – Структура таблицы «clients»

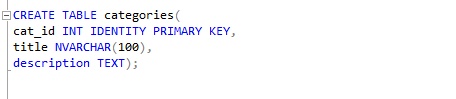


Рисунок 7 – Структура таблицы «categories»

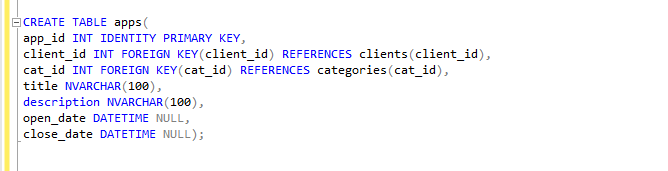


Рисунок 8 – Структура таблицы «apps»



Рисунок 9 – Структура таблицы «executors»

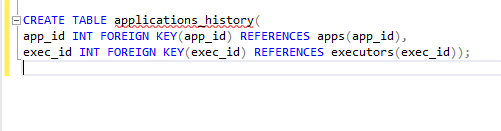


Рисунок 10 – Структура таблицы «applications\_history»

На рисунке 11 представлена модель базы данных, отражающая индексы и связи таблиц БД.

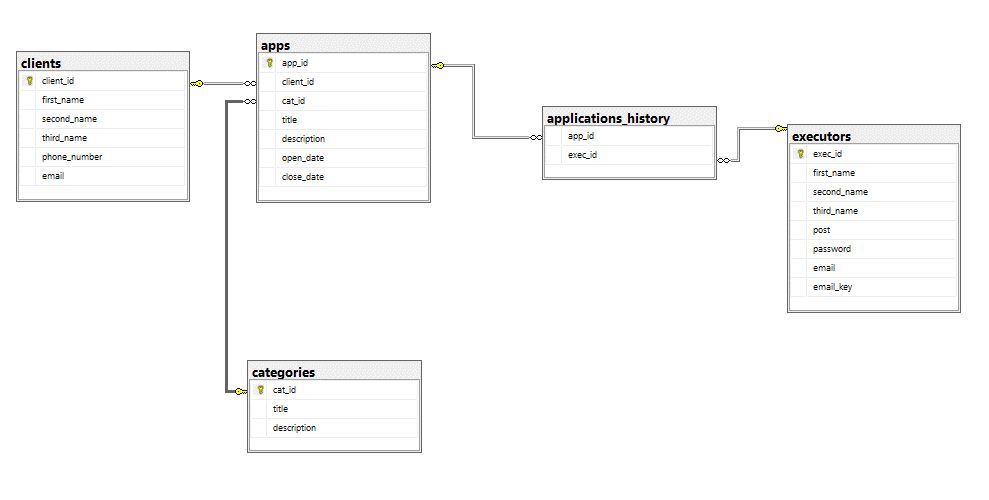


Рисунок 11 – Модель связей таблиц базы данных

# **2.3. Импорт и экспорт данных**

Импорт данных реализуем с помощью SQL Server Import and Export Wizard.

Создаем Excel файл, в котором данные будут импортированы в базу данных SQL Server. Убедившись, что данные в файле соответствуют структуре таблицы, в которую мы планируем импортировать данные, реализуем импорт.

Результат импорта предоставлен на рисунке 12.

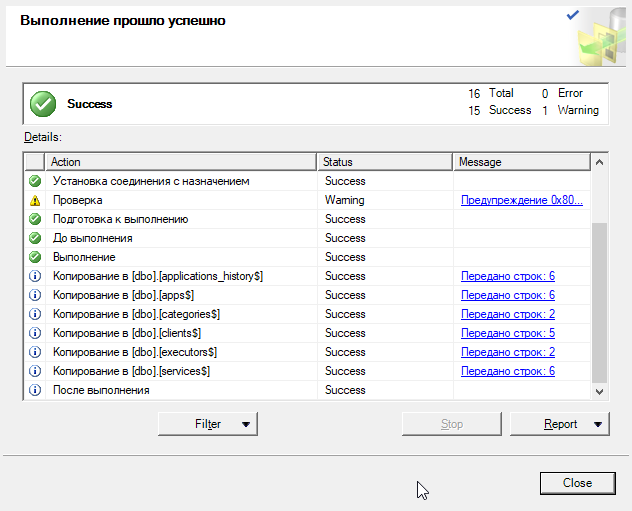


Рисунок 12 – Импорт данных из SQL

На рисунках 13-17 показано заполнение таблиц с помощью импорта данных из Excel файла.

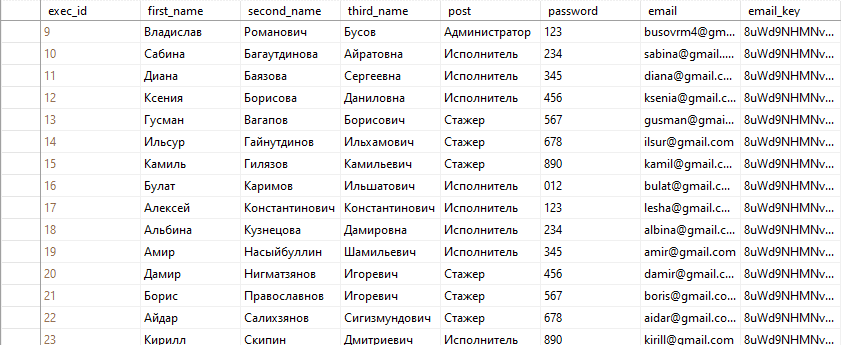


Рисунок 13 – Вывод таблицы Executors

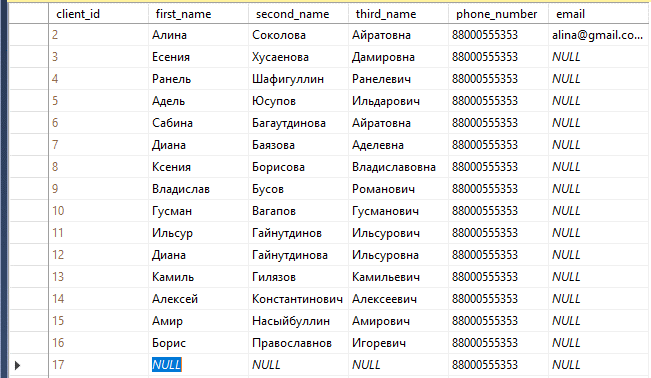


Рисунок 14 – Вывод таблицы Clients

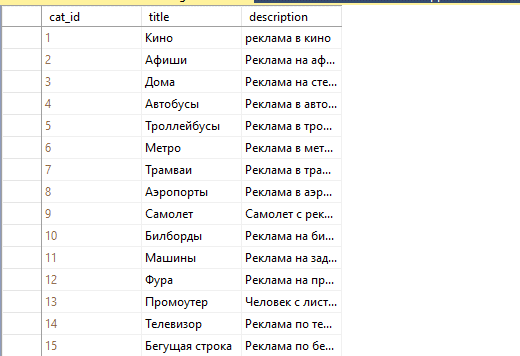


Рисунок 15 – Вывод таблицы Categories

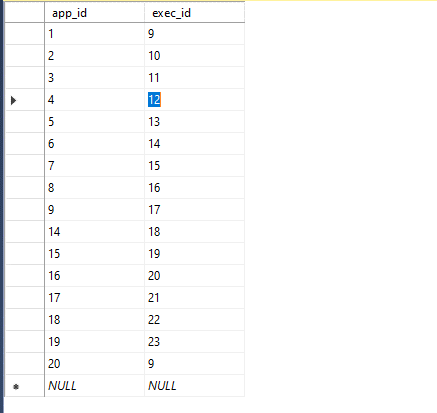


Рисунок 16 – Вывод таблицы ApplicationsHistory

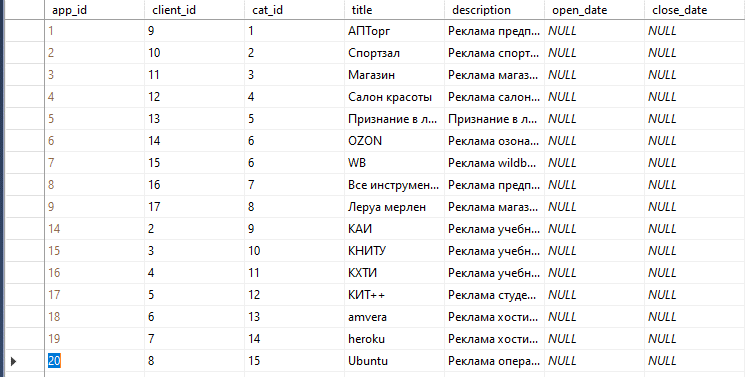


Рисунок 17 – Вывод таблицы Applications

После успешного выполнения импорта данных перейдем к экспорту данных из SQL в Excel (Рисунок 18).

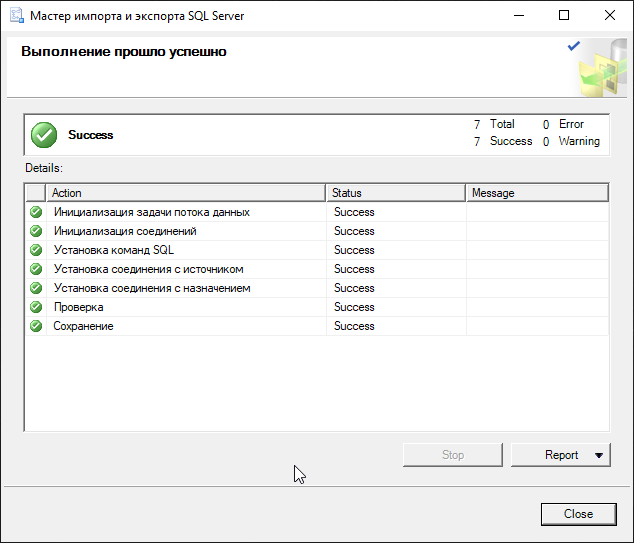


Рисунок 18 – Экспорт данных

Далее проверяем наличие данных в Excel файле. Все успешно экспортировалось (Рисунок 19).

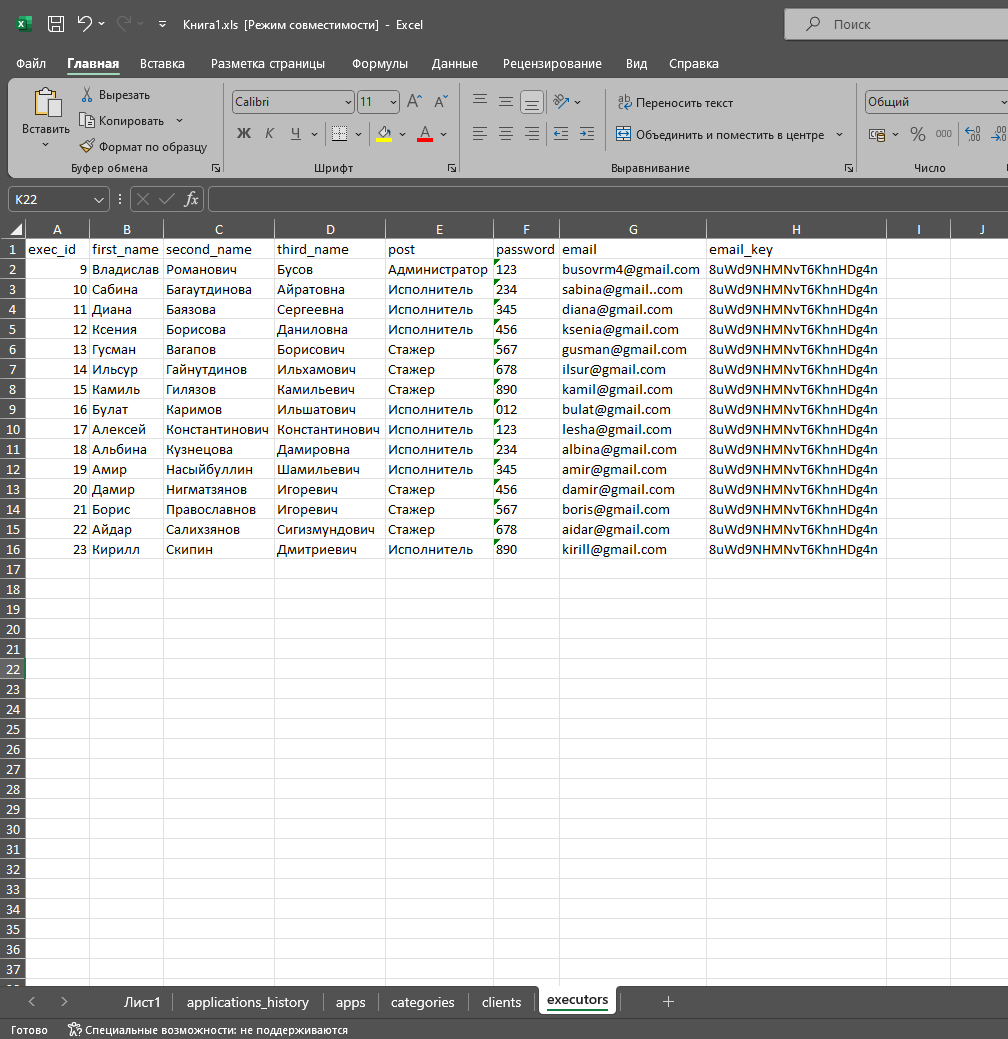


Рисунок 19 – Результат экспорта в Excel

# **2.4. Поддержка целостности данных**

# **2.4.1. Декларативная поддержка ограничений целостности**

Обеспечение целостности базы данных включает в себя соблюдение ряда ограничений, которые необходимы для сохранения непротиворечивости данных. Среди этих ограничений можно выделить два основных типа: ограничения диапазонов значений атрибутов и структурные ограничения на кортежи отношений.

Ограничения диапазонов значений атрибутов контролируют значения атрибутов отношений. Например, можно задать ограничение на диапазон значений числового атрибута или на формат вводимых данных.

Структурные ограничения включают требования целостности сущностей и ссылочной целостности. Первое требование состоит в том, что каждый кортеж отношения должен быть уникальным, то есть каждому экземпляру сущности должен соответствовать только один кортеж. Второе требование связано с понятием внешнего ключа и состоит в том, что для каждого значения внешнего ключа в родительской таблице должна существовать соответствующая строка в дочерней таблице с таким же значением первичного ключа.

В MS SQL Server поддержка ссылочной целостности обычно осуществляется с помощью определения связей между таблицами, например, через использование внешних ключей и настройку связей в диаграмме базы данных.

# **2.4.2. Процедурная поддержка ограничений целостности**

Для организации процедурной поддержки ограничения целостности данных разработаны хранимые процедуры, триггеры и представления.

**Триггер 1.** Триггер trg\_DeleteCat сработает в случае, если из таблицы categories будет удалена запись. Результат (Рисунок 20).

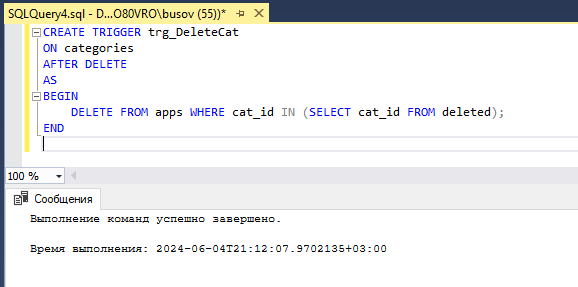


Рисунок 20 – Триггер №1

CREATE TRIGER trg\_DeleteCat

ON categories

AFTER DELETE

AS

BEGIN

DELETE FROM services WHERE category IN (SELECT cat\_id FROM deleted);

END

**Триггер 2.** Триггер trg\_ChangeAppsCatOnDelete предназначен для обновления значения столбца cat\_id в таблице apps. При удалении записи из таблицы categories столбец cat\_id в таблице apps получит значение NULL. Результат (Рисунок 21).

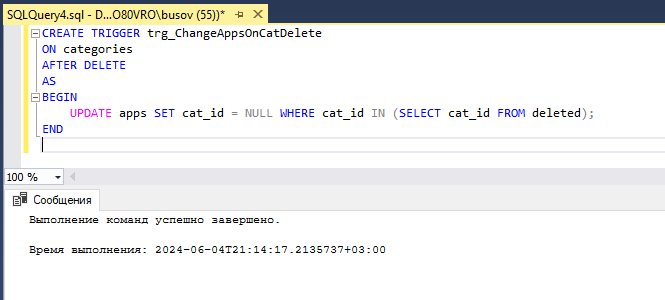


Рисунок 21 – Триггер №2

CREATE TRIGER trg\_ChangeApsOnCatDelete

ON categories

AFTER DELETE

AS

BEGIN

UPDATE aps SET cat\_id = NUL WHERE cat\_id IN (SELECT cat\_id FROM deleted);

END

**Триггер 3.** Триггер trg\_ServiceUpdate сработает после удаления записи из таблицы apps. Триггер предназначен для удаления заявки из истории заявок. Результат (Рисунок 22).

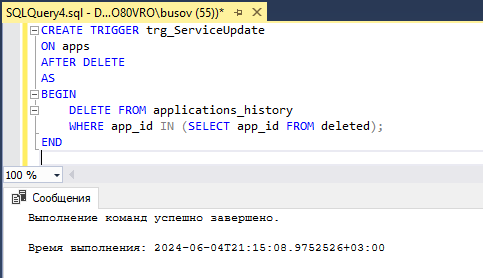


Рисунок 22 – Триггер №3

CREATE TRIGER trg\_ServiceUpdate

ON apps

AFTER DELETE

AS

BEGIN

DELETE FROM aplications\_history

WHERE ap\_id IN (SELECT ap\_id FROM deleted);

END

**Триггер 4.** Триггер trg\_ExecDelete сработает в случае удаления записи из таблицы executors. При удалении исполнителя триггер автоматически удаляет все записи в таблице applications\_history, которые связаны с удаленной записью по внешнему ключу. Результат (Рисунок 23)

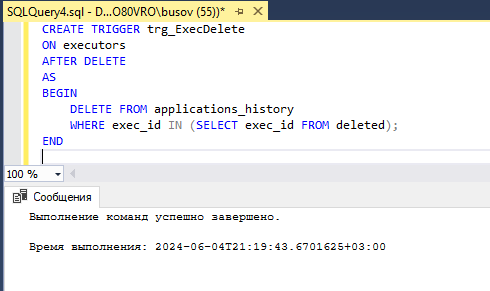


Рисунок 23 – Триггер №4

CREATE TRIGER trg\_ExecDelete

ON executors

AFTER DELETE

AS

BEGIN

DELETE FROM aplications\_history

WHERE exec\_id IN (SELECT exec\_id FROM deleted);

END

**Триггер 5.** Триггер trg\_ClientsDelete сработает после удаления из таблицы clients одной из записей. После удаления из таблицы apps удалятся все записи, которые по внешнему ключу связаны в этим клиентом. Результат (Рисунок 24).

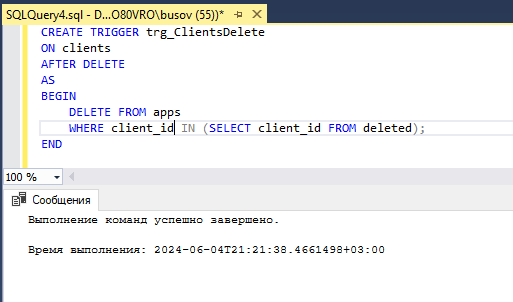


Рисунок 24 – Триггер №5

CREATE TRIGER trg\_ClientsDelete

ON clients

AFTER DELETE

AS

BEGIN

DELETE FROM aps

WHERE client\_id IN (SELECT client\_id FROM deleted);

END

**Процедура 1.** Даная процедура позволяет нам получить краткую информацию для связи с пользователями. Результат (Рисунок 25).

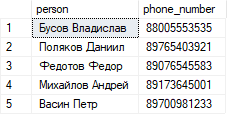


Рисунок 25 – Процедура №1

CREATE PROCEDURE p1 AS

SELECT phone\_number, person FROM clients;

**Процедура 2.** Процедура p2 позволяет нам получить информацию о связях между категориями и услугами. Результат (Рисунок 26).

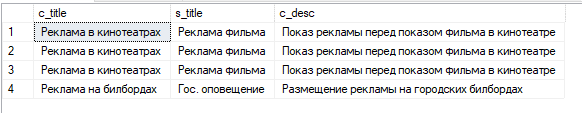


Рисунок 26 – Процедура №2

CREATE PROCEDURE p2 AS

BEGIN

SELECT

Categories.title AS c\_title,

Services.title AS s\_title,

Categories.description as c\_desc

FROM

Services JOIN categories ON

Services.category = categories.cat\_id

WHERE services.cost <= 500;

END

**Процедура 3.** Процедура P3 позволяет узнать полную информацию о заявке и о том, какой исполнитель взял е в работу. Результат (Рисунок 27).

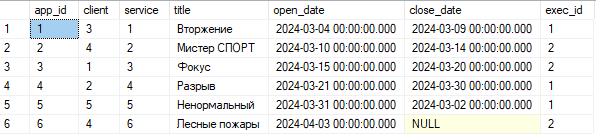


Рисунок 27 – Процедура №3

CREATE PROCEDURE p3 AS

BEGIN

SELECT apps.\*, applications\_history.exec\_id

FROM aps

LEFT JOIN aplications\_history ON

Aps.ap\_id = applications\_history.ap\_id;

END

**Процедура 4.** Процедура P4 дает информацию о заявке и о типе услуг, к которому она принадлежит. Результат (Рисунок 28).

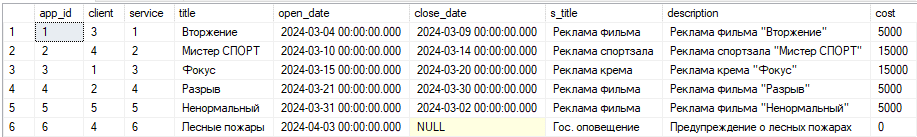


Рисунок 28 – Процедура №4

CREATE PROCEDURE p4 AS

BEGIN

SELECT apps.\*, services.title as s\_title, services.description, services.cost

FROM aps JOIN services

ON aps.service = services.service\_id;

END

**Процедура 5.** Даная процедура даёт информацию о том, когда была открыта и закрыта заявка. Заявки, которые сейчас в работе, не учитываются. Результат (Рисунок 29).

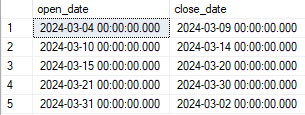


Рисунок 29 – Процедура №5

CREATE PROCEDURE p5 AS

BEGIN

SELECT open\_date, close\_date

FROM aps

WHERE close\_date IS NOT NUL;

END

**Представление 1.** Первое представление V1 позволяет нам получить краткую информацию для связи с пользователями. Результат на рисунке 30.

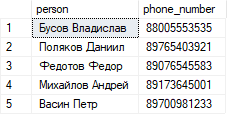


Рисунок 30 – Представление №1

CREATE VIEW v1 AS

SELECT clients.person, phone\_number FROM clients;

**Представление 2.** Второе представление V2 позволяет нам получить информацию о связях между категориями и услугами. Результат (Рисунок 31).

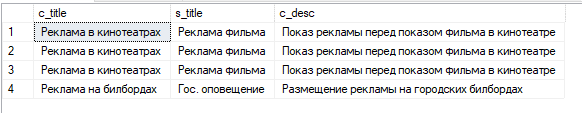


Рисунок 31 – Представление №2

CREATE VIEW v2 AS

SELECT

Categories.title AS c\_title,

Services.title AS s\_title,

Categories.description as c\_desc

FROM

Services JOIN categories ON

Services.category = categories.cat\_id

WHERE services.cost <= 50;

**Представление 3**. Третье представление V3 позволяет узнать полную информацию о заявке и о том, какой исполнитель взял е в работу. Результат на рисунке 32.

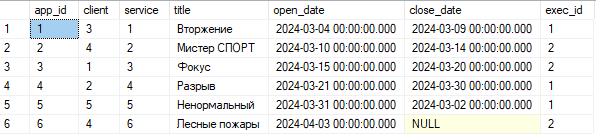


Рисунок 32 – Представление №3

CREATE VIEW v3 AS

SELECT apps.\*, applications\_history.exec\_id

FROM apps

LEFT JOIN aplications\_history ON

Aps.ap\_id = applications\_history.app\_id;

**Представление 4.** Четвертое представление V4 дает информацию о заявке и о типе услуг, к которому она принадлежит. Результат на рисунке 33.

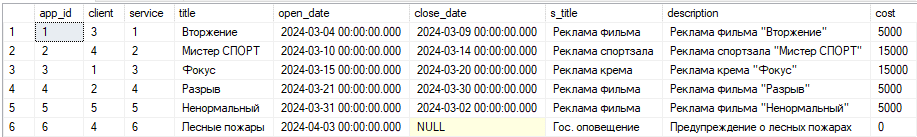


Рисунок 33 – Представление №4

CREATE VIEW v4 AS

SELECT apps.\*, services.title as s\_title, services.description, services.cost

FROM apps JOIN services

ON aps.service = services.service\_id;

**Представление 5.** Пятое представление V5 даёт информацию о том, когда была открыта и закрыта заявка. Заявки, которые сейчас в работе, не учитываются . Результат на рисунке 34.

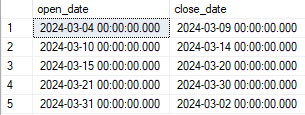


Рисунок 34 – Представление №5

CREATE VIEW v5 AS

SELECT open\_date, close\_date

FROM apps

WHERE close\_date IS NOT NULL;

# **2.5. Реализация доступа пользователей к базе данных**

При помощи Activity Monitor собираем и обрабатываем информацию о текущем состоянии SQL Server (Рисунок 35).

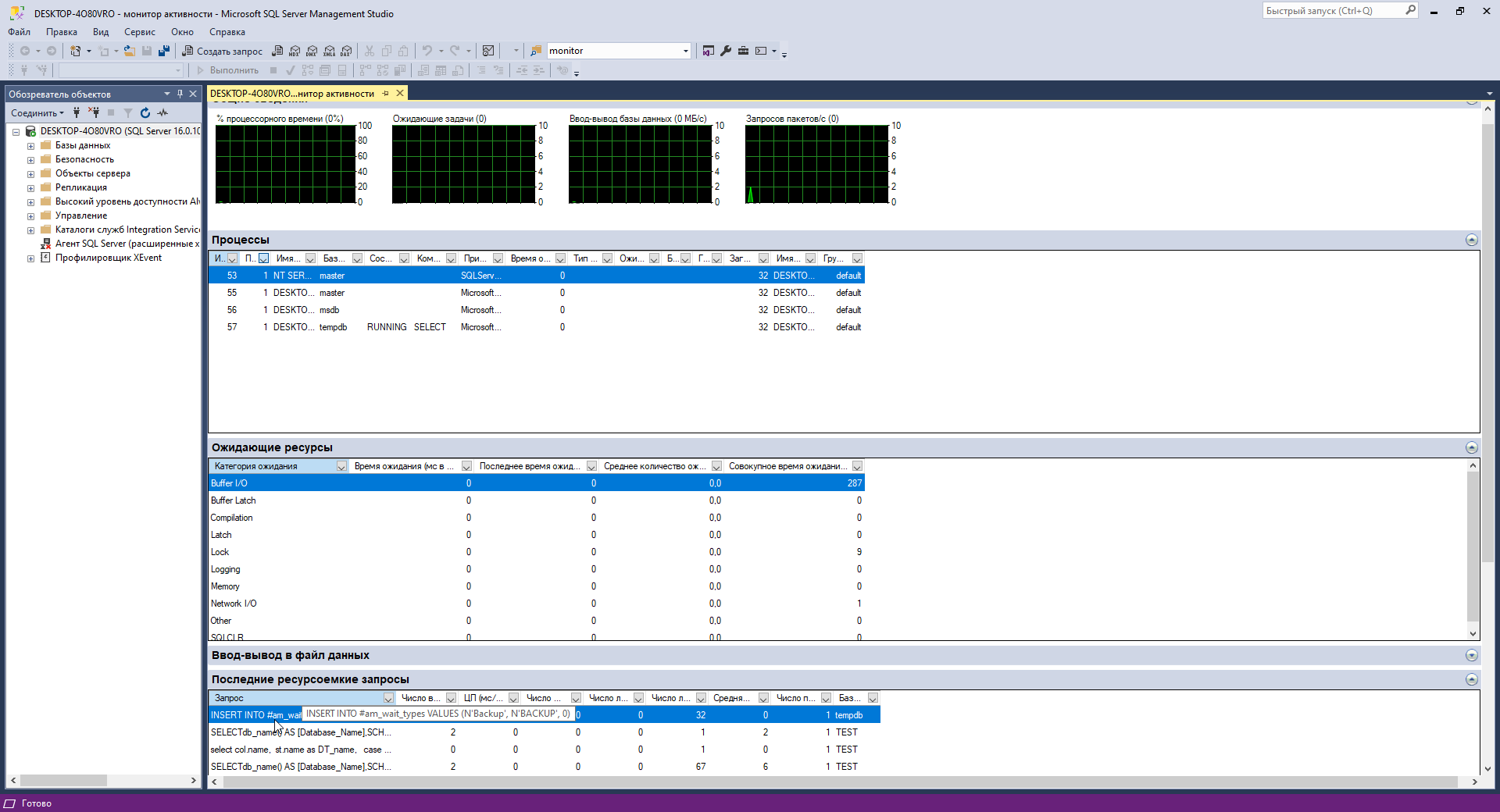


Рисунок 35 – Activity Monitor

Создаем 3 логина для входа, задав пароли 12345678 для пользователей admin, exec, client (Рисунок 36).



Рисунок 36 – Логины

Создадим роль Admin с логином VladAdmin, у которой будут все разрешения для взаимодействия с БД (Рисунок 37).

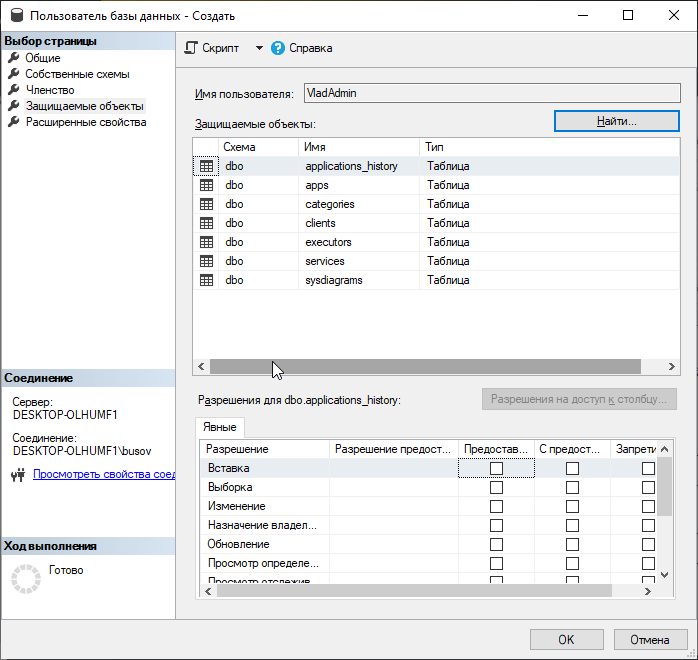


Рисунок 37 – Роль Admin

Создадим роль Exec с логином AntonExec. Данный пользователь имеет право просматривать записи и влиять на некоторые таблицы в базе данных (Рисунок 38).

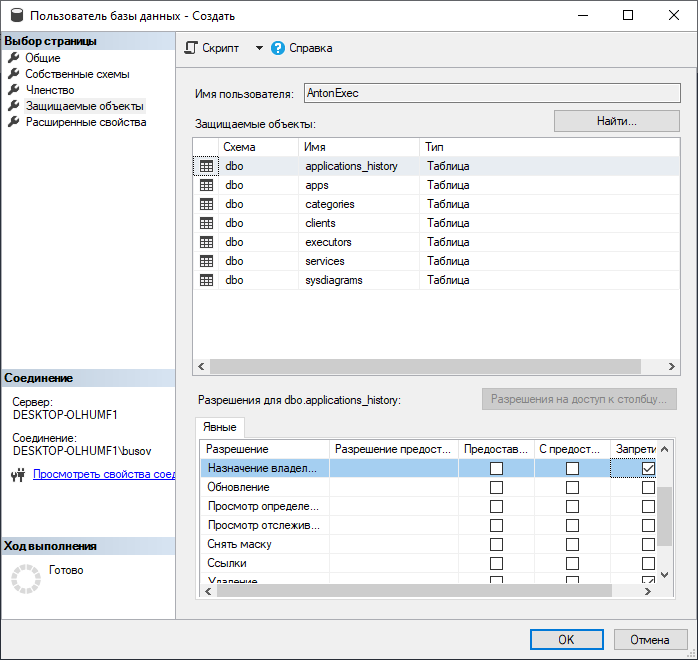


Рисунок 38 – Роль Exec

Создадим роль Client с логином DimaClient. Данный пользователь может просматривать данные о заявке (Рисунок 39).

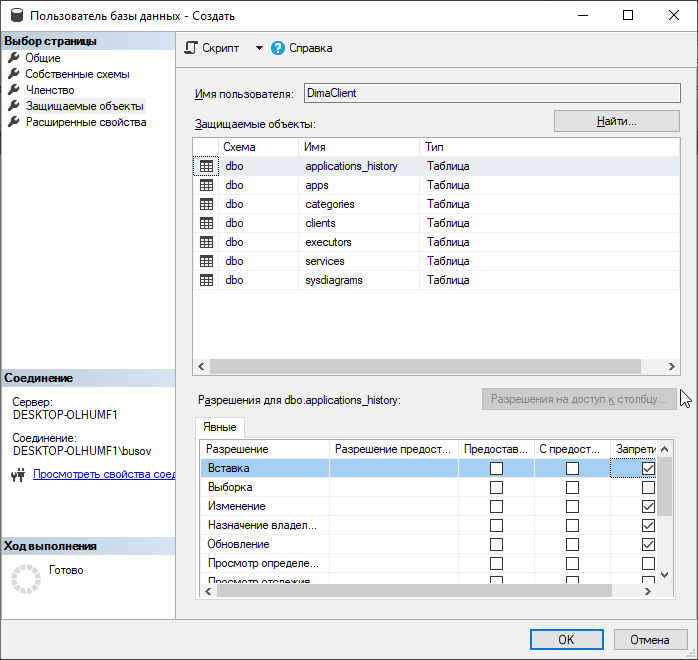


Рисунок 39 – Роль Client

# **ГЛАВА 3. СОЗДАНИЕ WPF ПРИЛОЖЕНИЯ И ПОДКЛЮЧЕНИЕ БАЗЫ ДАННЫХ**

В WPF-приложении реализованы интерфейсы взаимодействия для клиента, исполнителя и администратора системы. Клиент имеет возможность создать и отправить заявку. Исполнитель может просматривать заявки, брать их в работу и завершать. Администратор имеет право напрямую редактировать, создавать и удалять записи. Листинг разметки представлен в ПРИЛОЖЕНИИ А.

Для реализации функционала таблиц были созданы окна. Окна программы и ее работа представлена на рисунках 40 – 52.

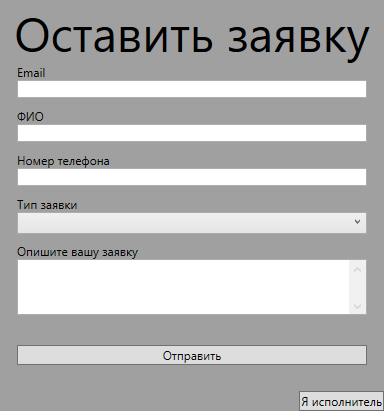


Рисунок 40 – Главный экран

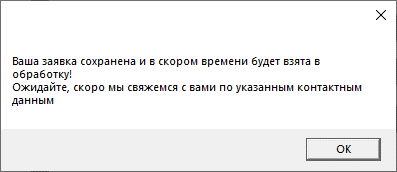


Рисунок 41 – Создание заявки клиентом сервиса

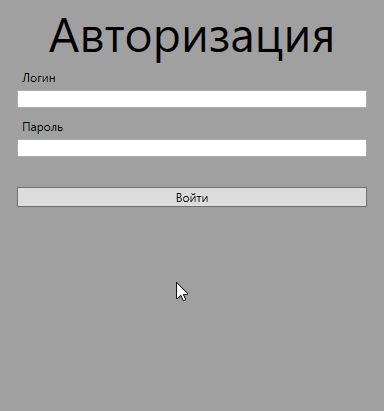


Рисунок 42 – После нажатия на кнопку «Я исполнитель» пользователь должен авторизоваться в системе под логином и паролем, которые задает администратор системы

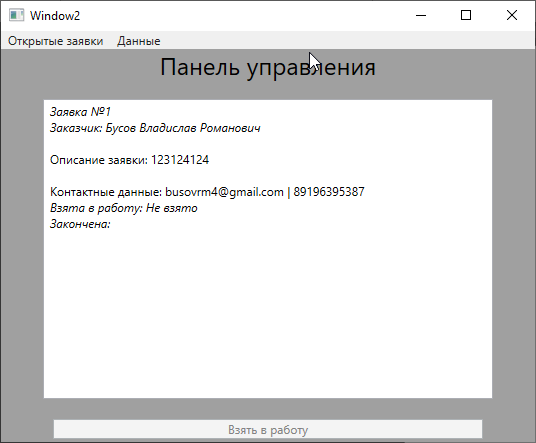


Рисунок 43 – Окно управления заявками исполнителя

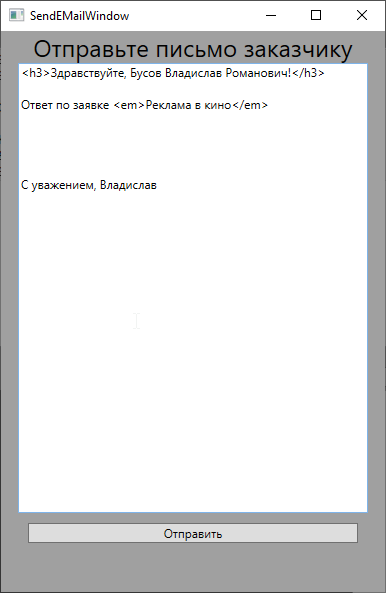


Рисунок 44 – Отправка сообщения по электронной почте при взятии в работу заявки исполнителем

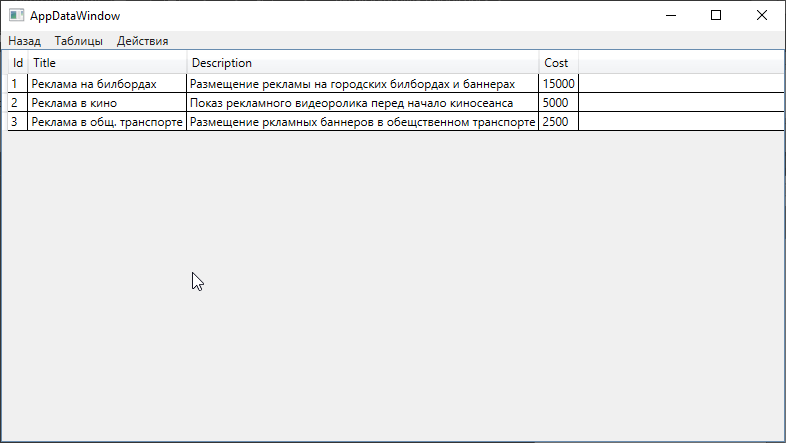


Рисунок 45 – Таблица «Categories»

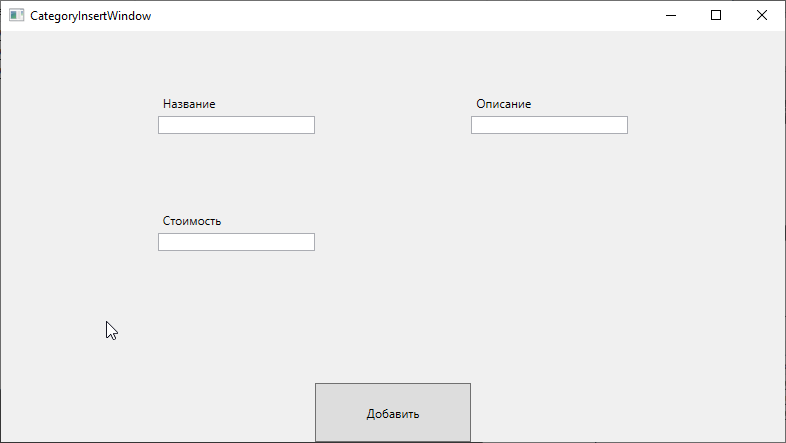


Рисунок 46 – Окно добавления данных в таблицу Categories

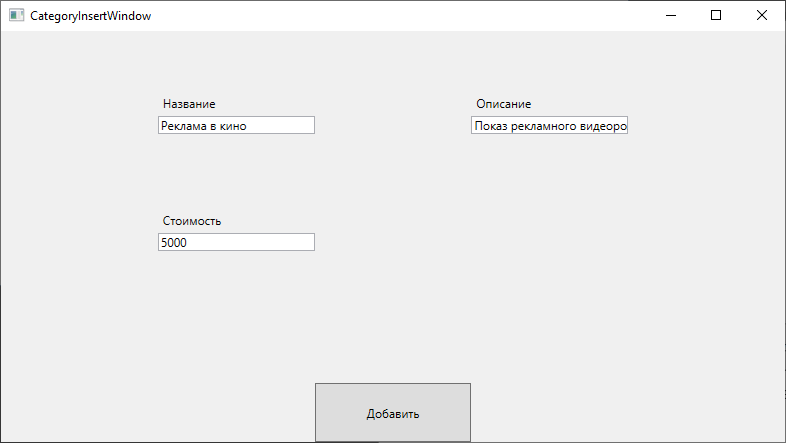


Рисунок 47 – Окно изменения данных

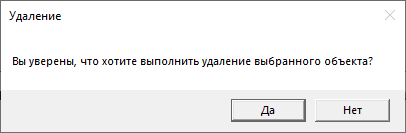


Рисунок 48 – Подтверждение удаления данных

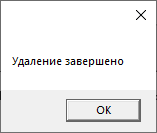


Рисунок 49 – Сообщение об успешном удалении данных из таблицы

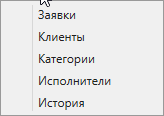


Рисунок 50 – Выбор таблицы

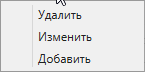


Рисунок 51 – Выбор действия с записями в таблице

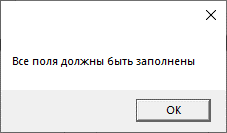


Рисунок 52 – Ошибка при заполнении данных

В случае, если пользователь введет некорректные данные, ему выйдет сообщение об ошибке. Результат данной операции показан на рисунке 52.

# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсовой работы была создана база данных «Учёт заявок на выполнение услуг рекламного агентства», направленным на организацию эффективной работы исполнителей и клиентов в рекламной деятельности. В ней объединяется информация о заявках, клиентах, исполнителях, категориях и истории выполнения заявок, необходимая для успешной работы и выполнения требований клиентов.

Была проведена детальная аналитика предметной области, изучены особенности работы кураторов в образовательных учреждениях и задачи, стоящие перед ними.

Описана задача проекта, определены ключевые моменты, которые необходимо учесть при проектировании базы данных, такие как учет заявок, клиентов и исполнителей, информация о заявках клиентах, категории заявок и другие важные данные.

Рассмотрен список выделенных сущностей и их атрибутов, включая данные о клиентах, исполнителях, заявках, категориях и истории выполнения заявок исполнителями. Создана модель "Сущность-связь", где определены взаимосвязи между сущностями.

База данных включает в себя набор таблиц с определенными связями между ними для удобного доступа к информации и обеспечения целостности данных. Исследованы функциональные зависимости между атрибутами сущностей, и произведена нормализация базы данных до 3NF для исключения избыточности и обеспечения эффективного хранения данных.

Создана инфологическая и даталогическая модели БД.

Для проектирования базы данных была выбрана СУБД MS SQL Server.

Также были рассмотрены механизмы защиты данных и требования к техническому обеспечению для обеспечения безопасности и надежности хранения информации.

В процессе выполнения курсового проекта были получены ценные навыки в проектировании баз данных, разработке запросов и обеспечении функциональности системы, что может быть использовано для реализации более сложных проектов.

# **СПИСОК ЛИТЕРАТУРЫ**

1. Коротков Э.М. Разработка баз данных: учебник. - 2-е изд. – М.: ИНФРА - М, 2016. – 398 с.
2. Кузин А. В. Базы данных: учеб. Пособие для студентов ВУЗов / А. В. Кузин, С. В. Левонисова. – 5-е изд., испр. – М.: издательский центр «Академия», 2012. – 320 с.
3. Кузнецов С. Д. Базы данных: учебник для студентов учреждений высшего профессионального образования / С. Д. Кузнецов – – М.: издательский центр «Академия», 2012. – 496 с.
4. Мирошниченко Г. А. Реляционные базы данных: практические приемы оптимальных решений. – СПб.: БХВ-Петербург, 2005. – 400 с.
5. Мухин К.А. Проектирование баз данных, М.: Горячая линия – Телеком, 2017. – 240 с.
6. Морган, С. Проектирование и оптимизация доступа к базам данных Microsoft SQL Server 2005 / С. Морган. - М.: Русская редакция, 2008. - 480 c.
7. <https://metanit.com/sql/sqlserver/> – Руководство по MS SQL Server 2022

# **ПРИЛОЖЕНИЕ А**

**XAML**

**Окно «Window1»**

<Window x:Class="WpfApp1.Window1"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:WpfApp1"

mc:Ignorable="d"

Title="Window1" Height="450" Width="400">

<DockPanel Background="#A0A0A0">

<StackPanel DockPanel.Dock="Top">

<TextBlock HorizontalAlignment="Center" FontSize="48" TextWrapping="Wrap" Text="Авторизация"/>

</StackPanel>

<StackPanel DockPanel.Dock="Top" Width="350">

<Label Content="Логин"/>

<TextBox Name="Login" Margin="0px 0px 0px 5px"/>

<Label Content="Пароль"/>

<TextBox Name="Password" Margin="0px 0px 0px 30px"/>

<Button Name="EnterButton" Content="Войти" Click="EnterButton\_Click"/>

</StackPanel>

</DockPanel>

</Window>

**Окно «Window2»**

<Window x:Class="WpfApp1.Window2"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:WpfApp1"

mc:Ignorable="d"

Title="Window2" Height="450" Width="550">

<DockPanel Background="#A0A0A0">

<StackPanel DockPanel.Dock="Top">

<Menu>

<MenuItem Name="AppsMenuItem" Header="\_Заявки">

<MenuItem Header="\_Мои заявки" Click="MenuItem\_Click"/>

<MenuItem Header="\_Открытые заявки" Click="MenuItem\_Click\_1"/>

</MenuItem>

<MenuItem Header="\_Данные" Click="DataMenuItem\_Click">

</MenuItem>

</Menu>

<TextBlock HorizontalAlignment="Center" TextWrapping="Wrap" FontSize="24" Text="Панель управления"/>

</StackPanel>

<StackPanel DockPanel.Dock="Bottom" Width="450">

<Menu Name="CloseApp">

<MenuItem Click="CloseApp\_Click" Header="\_Завершить выполнение"/>

</Menu>

<ListBox ScrollViewer.VerticalScrollBarVisibility="Auto" x:Name="MyApps" Height="300" SelectionChanged="AppSelect">

<ListBox.ItemTemplate>

<DataTemplate>

<StackPanel>

<TextBlock Width="400" TextWrapping="Wrap" FontStyle="Italic" Text="{Binding Title}"/>

<TextBlock Width="400" TextWrapping="Wrap" Text="{Binding Content}"/>

<TextBlock Width="400" TextWrapping="Wrap" Text="{Binding Contacts}"/>

<TextBlock Width="400" TextWrapping="Wrap" FontStyle="Italic" Text="{Binding Dates}"/>

</StackPanel>

</DataTemplate>

</ListBox.ItemTemplate>

</ListBox>

<Button Margin="10 20 10 0" Name="SubmitAppButton" Click="SubmitAppButton\_Click" Content="Взять в работу"/>

</StackPanel>

</DockPanel>

</Window>

**Окно «SendEMailWindow»**

<Window x:Class="WpfApp1.SendEMailWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:WpfApp1"

mc:Ignorable="d"

Title="SendEMailWindow" Height="600" Width="400">

<DockPanel Background="#A0A0A0">

<StackPanel DockPanel.Dock="Top">

<TextBlock Text="Отправьте письмо заказчику" FontSize="24" TextWrapping="Wrap" HorizontalAlignment="Center"/>

</StackPanel>

<StackPanel DockPanel.Dock="Top" Width="350">

<TextBox Name="EmailTextBox" Height="450" TextWrapping="Wrap" AcceptsReturn="True" VerticalScrollBarVisibility="Auto"/>

<Button Click="SendEmail" Margin="10 10 10 0" Content="Отправить"/>

</StackPanel>

</DockPanel>

</Window>

**Окно «ExecutorInsertWindow»**

<Window x:Class="WpfApp1.InsertWindows.ExecutorInsertWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:WpfApp1.InsertWindows"

mc:Ignorable="d"

Title="ExecutorInsertWindow" Height="450" Width="800">

<Grid Background="#F0F0F0">

<Grid.RowDefinitions>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition/>

<ColumnDefinition/>

<ColumnDefinition/>

<ColumnDefinition/>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<StackPanel Grid.Row="1" Grid.Column="1">

<Label Content="Имя"/>

<TextBox Name="ExecName"/>

</StackPanel>

<StackPanel Grid.Row="1" Grid.Column="3">

<Label Content="Фамилия"/>

<TextBox Name="ExecSurname"/>

</StackPanel>

<StackPanel Grid.Row="3" Grid.Column="1">

<Label Content="Должность"/>

<TextBox Name="ExecPost"/>

</StackPanel>

<StackPanel Grid.Row="3" Grid.Column="3">

<Label Content="Пароль"/>

<TextBox Name="ExecPassword"/>

</StackPanel>

<StackPanel Grid.Row="5" Grid.Column="1">

<Label Content="Почта"/>

<TextBox Name="ExecEmail"/>

</StackPanel>

<StackPanel Grid.Row="5" Grid.Column="3">

<Label Content="Ключ почты"/>

<TextBox Name="ExecEmailKey"/>

</StackPanel>

<Button Grid.Row="6" Grid.Column="2" Content="Добавить" Click="Button\_Click"/>

</Grid>

</Window>

**Окно «ClientInsertWindow»**

<Window x:Class="WpfApp1.InsertWindows.ClientInsertWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:WpfApp1.InsertWindows"

mc:Ignorable="d"

Title="ClientInsertWindow" Height="450" Width="800">

<Grid Background="#F0F0F0">

<Grid.RowDefinitions>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition/>

<ColumnDefinition/>

<ColumnDefinition/>

<ColumnDefinition/>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<StackPanel Grid.Row="1" Grid.Column="1">

<Label Content="Имя"/>

<TextBox Name="ClientName"/>

</StackPanel>

<StackPanel Grid.Row="1" Grid.Column="3">

<Label Content="Номер телефона"/>

<TextBox Name="ClientPhoneNumber"/>

</StackPanel>

<StackPanel Grid.Row="3" Grid.Column="1">

<Label Content="Почта"/>

<TextBox Name="ClientEmail"/>

</StackPanel>

<Button Grid.Row="6" Grid.Column="2" Content="Добавить" Click="Button\_Click"/>

</Grid>

</Window>

**Окно «CategoryInsertWindow»**

<Window x:Class="WpfApp1.InsertWindows.CategoryInsertWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:WpfApp1.InsertWindows"

mc:Ignorable="d"

Title="CategoryInsertWindow" Height="450" Width="800">

<Grid Background="#F0F0F0">

<Grid.RowDefinitions>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition/>

<ColumnDefinition/>

<ColumnDefinition/>

<ColumnDefinition/>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<StackPanel Grid.Row="1" Grid.Column="1">

<Label Content="Название"/>

<TextBox Name="CatTitle"/>

</StackPanel>

<StackPanel Grid.Row="1" Grid.Column="3">

<Label Content="Описание"/>

<TextBox Name="CatDesc"/>

</StackPanel>

<StackPanel Grid.Row="3" Grid.Column="1">

<Label Content="Стоимость"/>

<TextBox Name="CatPrice"/>

</StackPanel>

<Button Grid.Row="6" Grid.Column="2" Content="Добавить" Click="Button\_Click"/>

</Grid>

</Window>

**Окно «AppInsertWindow»**

<Window x:Class="WpfApp1.InsertWindows.AppInsertWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:WpfApp1.InsertWindows"

mc:Ignorable="d"

Title="AppInsertWindow" Height="450" Width="800">

<Grid Background="#F0F0F0">

<Grid.RowDefinitions>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

<RowDefinition/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition/>

<ColumnDefinition/>

<ColumnDefinition/>

<ColumnDefinition/>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<StackPanel Grid.Row="1" Grid.Column="1">

<Label Content="Id клиета"/>

<TextBox Name="AppClientId"/>

</StackPanel>

<StackPanel Grid.Row="1" Grid.Column="3">

<Label Content="Категория"/>

<ComboBox x:Name="CategoriesDropDown" Margin="0px 0px 0px 10px">

</ComboBox>

</StackPanel>

<StackPanel Grid.Row="3" Grid.Column="1">

<Label Content="Название"/>

<TextBox Name="AppTitle"/>

</StackPanel>

<StackPanel Grid.Row="3" Grid.Column="3">

<Label Content="Описание"/>

<TextBox Name="AppDescription"/>

</StackPanel>

<StackPanel Grid.Row="5" Grid.Column="1">

<Label Content="Дата открытия"/>

<TextBox Name="AppOpenDate"/>

</StackPanel>

<StackPanel Grid.Row="5" Grid.Column="3">

<Label Content="Дата закрытия"/>

<TextBox Name="AppCloseDate"/>

</StackPanel>

<Button Grid.Row="6" Grid.Column="2" Content="Добавить" Click="Button\_Click"/>

</Grid>

</Window>

**Окно «AppDataWindow»**

<Window x:Class="WpfApp1.DataWindows.AppDataWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:WpfApp1.DataWindows"

mc:Ignorable="d"

Title="AppDataWindow" Height="450" Width="800">

<DockPanel>

<Menu DockPanel.Dock="Top">

<MenuItem Header="Назад" Click="BackBtn\_Click"/>

<MenuItem Header="Таблицы">

<MenuItem Click="LoadData" Header="Заявки"/>

<MenuItem Click="LoadData" Header="Клиенты"/>

<MenuItem Click="LoadData" Header="Категории"/>

<MenuItem Click="LoadData" Header="Исполнители"/>

<MenuItem Click="LoadData" Header="История"/>

</MenuItem>

<MenuItem Header="Действия">

<MenuItem Header="Удалить" Click="DeleteBtn\_Click"/>

<MenuItem Header="Изменить" Click="UpdateBtn\_Click"/>

<MenuItem Header="Добавить" Click="AddBtn\_Click"/>

</MenuItem>

</Menu>

<DataGrid Name="DataTable" DockPanel.Dock="Top" IsReadOnly="True">

</DataGrid>

</DockPanel>

</Window>

# **ПРИЛОЖЕНИЕ Б**

**XAML.CS**

**Окно «Window1»**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using WpfApp1.Models;

namespace WpfApp1

{

public partial class Window1 : Window

{

Executor exec;

public Window1()

{

InitializeComponent();

}

private void EnterButton\_Click(object sender, RoutedEventArgs e)

{

string login = this.Login.Text;

string psswd = this.Password.Text;

if (login == "" || psswd == "") {

MessageBox.Show("Ошибка! Все поля должны быть заполнены");

return;

}

using (MobileContext db = new MobileContext())

{

foreach (Executor exec in db.Executors)

{

if ($"{exec.Surname}\_{exec.Id}" == login && exec.Password == psswd)

{

this.exec = exec;

break;

}

}

}

if (this.exec != null)

{

Window2 win = new Window2(exec);

this.Close();

win.Show();

} else {

MessageBox.Show("Неверные данные для входа!");

}

}

}

}

**Окно «Window2»**

using WpfApp1.Models;

using System.Windows;

using System.Windows.Controls;

using System.Net.Mail;

using System.Net;

using System.Data.Entity.Migrations;

using Microsoft.VisualBasic;

using WpfApp1.DataWindows;

namespace WpfApp1

{

/// <summary>

/// Логика взаимодействия для Window2.xaml

/// </summary>

public partial class Window2 : Window

{

Executor exec;

public Window2(Executor exec)

{

InitializeComponent();

this.exec = exec;

this.MyApps.Visibility = Visibility.Hidden;

this.SubmitAppButton.Visibility = Visibility.Hidden;

this.CloseApp.Visibility = Visibility.Hidden;

this.SubmitAppButton.IsEnabled = false;

this.CloseApp.IsEnabled = false;

}

private void MenuItem\_Click(object sender, RoutedEventArgs e)

{

this.MyApps.Visibility = Visibility.Visible;

this.SubmitAppButton.Visibility = Visibility.Hidden;

this.CloseApp.Visibility = Visibility.Visible;

this.SubmitAppButton.IsEnabled = false;

this.CloseApp.IsEnabled = false;

this.AppsMenuItem.Header = "\_Мои заявки";

List<object> apps = new List<object>();

using (MobileContext db = new MobileContext())

{

var my\_history = db.Histories.Where(x => x.ExecId == exec.Id);

if (my\_history.Count() != 0)

{

foreach (History his in my\_history)

{

Models.App app = db.Apps.FirstOrDefault(x => x.Id == his.AppId && x.CloseDate == "");

if (app != null)

{

Client client = db.Clients.First(x => x.Id == app.ClientId);

string title = $"Заказчик: {client.Name}\r\n";

string contacts = $"Контактные данные: {client.Email} | {client.PhoneNumber}";

string content = $"Описание заявки: {app.Description}\r\n";

string dates = $"Взята в работу: {app.OpenDate}\r\n" +

$"Закончена: {app.CloseDate}";

apps.Add(new ListBoxElement

{

Content = content,

Title = title,

Dates = dates,

Contacts = contacts,

AppId = app.Id

}

);

}

}

}

else

{

apps.Add(new ListBoxElement { Title = null, Content = "Сейчас нет ни одной открытой заявки" } );

}

}

this.MyApps.ItemsSource = apps;

}

private void MenuItem\_Click\_1(object sender, RoutedEventArgs e)

{

this.MyApps.Visibility = Visibility.Visible;

this.SubmitAppButton.Visibility = Visibility.Visible;

this.CloseApp.Visibility = Visibility.Hidden;

this.SubmitAppButton.IsEnabled = false;

this.CloseApp.IsEnabled = false;

this.AppsMenuItem.Header = "\_Открытые заявки";

List<object> apps = new List<object>();

using (MobileContext db = new MobileContext())

{

var open\_apps = db.Apps.Where(x => x.OpenDate == "Не взято");

if (open\_apps.Count() != 0)

{

foreach (Models.App app in open\_apps)

{

Client client = db.Clients.First(x => x.Id == app.ClientId);

string title = $"Заявка №{app.Id}\r\nЗаказчик: {client.Name}\r\n";

string contacts = $"Контактные данные: {client.Email} | {client.PhoneNumber}";

string content = $"Описание заявки: {app.Description}\r\n";

string dates = $"Взята в работу: {app.OpenDate}\r\n" +

$"Закончена: {app.CloseDate}";

apps.Add( new ListBoxElement{

Content = content,

Title = title,

Dates = dates,

Contacts = contacts,

AppId = app.Id

}

);

}

}

else

{

apps.Add(new ListBoxElement { Title = null, Content = "Сейчас нет ни одной открытой заявки" });

}

}

this.MyApps.ItemsSource = apps;

}

private void SubmitAppButton\_Click(object sender, RoutedEventArgs e)

{

int app\_id = (this.MyApps.SelectedItem as ListBoxElement).AppId;

using (MobileContext db = new MobileContext())

{

Models.App app = db.Apps.First(x => x.Id == app\_id);

Client client = db.Clients.First(x => x.Id == app.ClientId);

SendEMailWindow win = new SendEMailWindow(app, client, this.exec);

win.ShowDialog();

}

}

private void AppSelect(object sender, SelectionChangedEventArgs e)

{

if (this.MyApps.SelectedItem != null)

{

if ((this.MyApps.SelectedItem as ListBoxElement).Title != null)

{

this.SubmitAppButton.IsEnabled = true;

this.CloseApp.IsEnabled = true;

}

}

}

private void UpdateListBox(object sender, EventArgs e)

{

this.MyApps.Visibility = Visibility.Hidden;

this.SubmitAppButton.Visibility = Visibility.Hidden;

this.CloseApp.Visibility = Visibility.Hidden;

this.SubmitAppButton.IsEnabled = false;

this.CloseApp.IsEnabled = false;

}

private void CloseApp\_Click(object sender, RoutedEventArgs e)

{

int app\_id = (this.MyApps.SelectedItem as ListBoxElement).AppId;

using (MobileContext db = new MobileContext())

{

Models.App app = db.Apps.First(x => x.Id == app\_id);

app.CloseDate = $"{DateTime.Now.Day}.{DateTime.Now.Month}.{DateTime.Now.Year}";

Client client = db.Clients.First(x => x.Id == app.ClientId);

db.Apps.AddOrUpdate(app);

db.SaveChanges();

MailAddress from = new MailAddress(exec.Email, exec.Name);

MailAddress to = new MailAddress(client.Email);

MailMessage m = new MailMessage(from, to);

m.Subject = "Заявка закрыта | AdPromouter";

m.Body = $"{client.Name}, ваша заявка {app.Title} выполнена и закрыта." +

$"По всем вопросам обращаться по контактам вашего исполнителя: " +

$"{exec.Email}";

m.IsBodyHtml = true;

SmtpClient smtp = new SmtpClient("smtp.mail.ru", 25);

smtp.Credentials = new NetworkCredential(exec.Email, exec.EmailKey);

smtp.EnableSsl = true;

smtp.Send(m);

}

MessageBox.Show("Заявка успешно завершена!");

this.UpdateListBox(sender, e);

}

private void DataMenuItem\_Click(object sender, RoutedEventArgs e)

{

var win = new AppDataWindow(this.exec);

win.Show();

this.Close();

}

}

public class ListBoxElement

{

public string Content {get; set; }

public string Title {get; set; }

public string Dates {get; set; }

public string Contacts {get; set; }

public int AppId { get; set; }

}

}

**Окно «SendEMailWindow»**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net.Mail;

using System.Net;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using WpfApp1.Models;

using System.Data.Entity.Migrations;

namespace WpfApp1

{

/// <summary>

/// Логика взаимодействия для SendEMailWindow.xaml

/// </summary>

public partial class SendEMailWindow : Window

{

Executor exec;

Client client;

Models.App app;

public SendEMailWindow(Models.App app, Client client, Executor exec)

{

this.exec = exec;

this.client = client;

this.app = app;

InitializeComponent();

this.EmailTextBox.Text += $"<h3>Здравствуйте, {client.Name}!</h3>\r\n\r\n" +

$"Ответ по заявке <em>{app.Title}</em>\r\n\r\n\r\n\r\n\r\n" +

$"С уважением, {exec.Name}";

}

public void SendEmail(object sender, RoutedEventArgs e)

{

MailAddress from = new MailAddress(exec.Email, exec.Name);

MailAddress to = new MailAddress(client.Email);

MailMessage m = new MailMessage(from, to);

m.Subject = "Ответ по заявке от AdPromouter";

m.Body = this.EmailTextBox.Text;

m.IsBodyHtml = true;

SmtpClient smtp = new SmtpClient("smtp.mail.ru", 25);

smtp.Credentials = new NetworkCredential(exec.Email, exec.EmailKey);

smtp.EnableSsl = true;

smtp.Send(m);

using (MobileContext db = new MobileContext())

{

db.Histories.Add(new History

{

ExecId = this.exec.Id,

AppId = this.app.Id

});

app.OpenDate = $"{DateTime.Now.Day}.{DateTime.Now.Month}.{DateTime.Now.Year}";

db.Apps.AddOrUpdate(app);

db.SaveChanges();

}

MessageBox.Show("Сообщение отправлено!");

this.Close();

}

}

}

**Окно «AppDataWindow»**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using WpfApp1.InsertWindows;

using WpfApp1.Models;

namespace WpfApp1.DataWindows

{

/// <summary>

/// Логика взаимодействия для AppDataWindow.xaml

/// </summary>

public partial class AppDataWindow : Window

{

Executor exec;

string CurrentTable;

public AppDataWindow(Executor exec)

{

InitializeComponent();

this.exec = exec;

}

private void UpdateData()

{

List<object> items = new List<object>();

using (var db = new MobileContext())

{

if (CurrentTable == "Заявки") { foreach (var el in db.Apps) { items.Add(el); } }

else if (CurrentTable == "Клиенты") { foreach (var el in db.Clients) { items.Add(el); } }

else if (CurrentTable == "Категории") { foreach (var el in db.Categories) { items.Add(el); } }

else if (CurrentTable == "Исполнители") { foreach (var el in db.Executors) { items.Add(el); } }

else if (CurrentTable == "История") { foreach (var el in db.Histories) { items.Add(el); } }

}

DataTable.ItemsSource = items;

}

public void LoadData(object sender, RoutedEventArgs e)

{

List<object> items = new List<object>();

using (var db = new MobileContext()) {

if ((sender as MenuItem).Header.ToString() == "Заявки") { foreach (var el in db.Apps) { items.Add(el); } }

else if ((sender as MenuItem).Header.ToString() == "Клиенты") { foreach (var el in db.Clients) { items.Add(el); } }

else if ((sender as MenuItem).Header.ToString() == "Категории") { foreach (var el in db.Categories) { items.Add(el); } }

else if ((sender as MenuItem).Header.ToString() == "Исполнители") { foreach (var el in db.Executors) { items.Add(el); } }

else if ((sender as MenuItem).Header.ToString() == "История") { foreach (var el in db.Histories) { items.Add(el); } }

CurrentTable = (sender as MenuItem).Header.ToString();

}

DataTable.ItemsSource = items;

}

public void BackBtn\_Click(object sender, RoutedEventArgs e)

{

var win = new Window2(this.exec);

win.Show();

this.Close();

}

public void UpdateBtn\_Click(object sender, RoutedEventArgs e)

{

var selected = DataTable.SelectedItem;

if (selected != null)

{

if (CurrentTable == "Заявки")

{

var win = new AppInsertWindow(this.exec, (Models.App)selected);

win.ShowDialog();

}

else if (CurrentTable == "Клиенты")

{

var win = new ClientInsertWindow(this.exec, (Client)selected);

win.ShowDialog();

}

else if (CurrentTable == "Категории")

{

var win = new CategoryInsertWindow(this.exec, (Category)selected);

win.ShowDialog();

}

else if (CurrentTable == "Исполнители")

{

var win = new ExecutorInsertWindow(this.exec, (Executor)selected);

win.ShowDialog();

}

else if (CurrentTable == "История")

{

var win = new HistoryInsertWindow(this.exec, (History)selected);

win.ShowDialog();

}

} else MessageBox.Show("Не выбрана ни одна строка!");

UpdateData();

}

public void AddBtn\_Click(object sender, RoutedEventArgs e)

{

if (CurrentTable == "Заявки")

{

var win = new AppInsertWindow(this.exec);

win.ShowDialog();

}

else if (CurrentTable == "Клиенты")

{

var win = new ClientInsertWindow(this.exec);

win.ShowDialog();

}

else if (CurrentTable == "Категории")

{

var win = new CategoryInsertWindow(this.exec);

win.ShowDialog();

}

else if (CurrentTable == "Исполнители")

{

var win = new ExecutorInsertWindow(this.exec);

win.ShowDialog();

}

else if (CurrentTable == "История")

{

var win = new HistoryInsertWindow(this.exec);

win.ShowDialog();

}

UpdateData();

}

public void DeleteBtn\_Click(object sender, RoutedEventArgs e)

{

var selected = DataTable.SelectedItem;

if (selected == null) return;

var result = MessageBox.Show(

"Вы уверены, что хотите выполнить удаление выбранного объекта?",

"Удаление",

MessageBoxButton.YesNo

);

if (result == MessageBoxResult.Yes)

{

using (var db = new MobileContext())

{

if (CurrentTable == "Заявки") { db.Apps.Remove(db.Apps.First( x => x.Id == ((Models.App)selected).Id)); }

else if (CurrentTable == "Клиенты") { db.Clients.Remove(db.Clients.First(x => x.Id == ((Client)selected).Id)); }

else if (CurrentTable == "Категории") { db.Categories.Remove(db.Categories.First(x => x.Id == ((Category)selected).Id)); }

else if (CurrentTable == "Исполнители") { db.Executors.Remove(db.Executors.First(x => x.Id == ((Executor)selected).Id)); }

else if (CurrentTable == "История") { db.Histories.Remove(db.Histories.First(x => x.Id == ((History)selected).Id)); }

db.SaveChanges();

UpdateData();

}

MessageBox.Show("Удаление завершено");

}

}

}

}

**Окно «HistoryInsertWindow»**

using System;

using System.Collections.Generic;

using System.Data.Entity.Migrations;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using WpfApp1.Models;

namespace WpfApp1.InsertWindows

{

/// <summary>

/// Логика взаимодействия для HistoryInsertWindow.xaml

/// </summary>

public partial class HistoryInsertWindow : Window

{

History current;

public HistoryInsertWindow(Executor exec, History current = null)

{

this.current = current;

InitializeComponent();

if (current != null)

{

HistoryExecId.Text = current.ExecId.ToString();

HistoryAppId.Text = current.AppId.ToString();

}

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

if (!int.TryParse(HistoryExecId.Text, out int exec\_id)) MessageBox.Show("Id должно быть числом");

else if (!int.TryParse(HistoryAppId.Text, out int app\_id)) MessageBox.Show("Id должно быть числом");

else

{

var hist = new History

{

AppId = app\_id,

ExecId = exec\_id

};

if (current != null) hist.Id = current.Id;

using (var db = new MobileContext())

{

db.Histories.AddOrUpdate(hist);

db.SaveChanges();

}

MessageBox.Show("Запись успешно добавлена!");

this.Close();

}

}

}

}

**Окно «ExecutorInsertWindow»**

using System;

using System.Collections.Generic;

using System.Data.Entity.Migrations;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using WpfApp1.Models;

namespace WpfApp1.InsertWindows

{

/// <summary>

/// Логика взаимодействия для ExecutorInsertWindow.xaml

/// </summary>

public partial class ExecutorInsertWindow : Window

{

Executor current;

public ExecutorInsertWindow(Executor exec, Executor current = null)

{

this.current = current;

InitializeComponent();

if (current != null)

{

ExecEmail.Text = current.Email;

ExecEmailKey.Text = current.EmailKey;

ExecName.Text = current.Name;

ExecPassword.Text = current.Password;

ExecPost.Text = current.Post;

ExecSurname.Text = current.Surname;

}

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

if (ExecEmail.Text == "" || ExecEmail.Text == " ") MessageBox.Show("Все поля должны быть заполнены");

else if (ExecEmailKey.Text == "" || ExecEmailKey.Text == " ") MessageBox.Show("Все поля должны быть заполнены");

else if (ExecName.Text == "" || ExecName.Text == " ") MessageBox.Show("Все поля должны быть заполнены");

else if (ExecPassword.Text == "" || ExecPassword.Text == " ") MessageBox.Show("Все поля должны быть заполнены");

else if (ExecPost.Text == "" || ExecPost.Text == " ") MessageBox.Show("Все поля должны быть заполнены");

else if (ExecSurname.Text == "" || ExecSurname.Text == " ") MessageBox.Show("Все поля должны быть заполнены");

else

{

var exec = new Executor

{

Email = ExecEmail.Text,

EmailKey = ExecEmailKey.Text,

Name = ExecName.Text,

Password = ExecPassword.Text,

Post = ExecPost.Text,

Surname = ExecSurname.Text

};

if (current != null) exec.Id = current.Id;

using (var db = new MobileContext())

{

db.Executors.AddOrUpdate(exec);

db.SaveChanges();

}

MessageBox.Show("Новый исполнитель успешно добавлен!");

this.Close();

}

}

}

}

**Окно «ClientInsertWindow»**

using System;

using System.Collections.Generic;

using System.Data.Entity.Migrations;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using WpfApp1.Models;

namespace WpfApp1.InsertWindows

{

/// <summary>

/// Логика взаимодействия для ClientInsertWindow.xaml

/// </summary>

public partial class ClientInsertWindow : Window

{

Client current;

public ClientInsertWindow(Executor exec, Client current = null)

{

this.current = current;

InitializeComponent();

if (current != null)

{

ClientName.Text = current.Name;

ClientEmail.Text = current.Email;

ClientPhoneNumber.Text = current.PhoneNumber;

}

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

if (ClientName.Text == "" || ClientName.Text == " " ||

ClientEmail.Text == "" || ClientEmail.Text == " "||

ClientPhoneNumber.Text == "" || ClientPhoneNumber.Text == " " ||

!long.TryParse(ClientPhoneNumber.Text, out \_) || ClientPhoneNumber.Text.Length != 11)

MessageBox.Show("Все поля должны быть заполнены");

else

{

var client = new Client

{

Name = ClientName.Text,

Email = ClientEmail.Text,

PhoneNumber = ClientPhoneNumber.Text

};

if (current != null) client.Id = current.Id;

using (var db = new MobileContext())

{

db.Clients.AddOrUpdate(client);

db.SaveChanges();

}

MessageBox.Show("Новый клиент успешно добавлен!");

this.Close();

}

}

}

}

**Окно «CategoryInsertWindow»**

using System;

using System.Collections.Generic;

using System.Data.Entity.Migrations;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Text;

using System.Threading.Tasks;

using System.Transactions;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using WpfApp1.Models;

namespace WpfApp1.InsertWindows

{

/// <summary>

/// Логика взаимодействия для CategoryInsertWindow.xaml

/// </summary>

public partial class CategoryInsertWindow : Window

{

Category current;

public CategoryInsertWindow(Executor exec, Category current = null)

{

this.current = current;

InitializeComponent();

if (current != null)

{

CatTitle.Text = current.Title;

CatPrice.Text = current.Cost.ToString();

CatDesc.Text = current.Description;

}

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

if (!int.TryParse(CatPrice.Text, out int cost)) MessageBox.Show("Ошибка! Цена должна быть в формате числа");

else

{

var cat = new Category

{

Cost = cost,

Description = CatDesc.Text,

Title = CatTitle.Text

};

if (this.current != null) cat.Id = current.Id;

using (var db = new MobileContext())

{

db.Categories.AddOrUpdate(cat);

db.SaveChanges();

}

MessageBox.Show("Новая категория успешно добавлена");

this.Close();

}

}

}

}

**Окно «AppInsertWindow»**

using System;

using System.Collections.Generic;

using System.Data.Entity.Migrations;

using System.Diagnostics.Tracing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using WpfApp1.Models;

namespace WpfApp1.InsertWindows

{

/// <summary>

/// Логика взаимодействия для AppInsertWindow.xaml

/// </summary>

public partial class AppInsertWindow : Window

{

Models.App current;

public AppInsertWindow(Executor exec, Models.App current = null)

{

this.current = current;

InitializeComponent();

List<string> categroies = new List<string>();

using (MobileContext db = new MobileContext())

{

int i = 0;

foreach (Category c in db.Categories)

{

categroies.Add($"{c.Title} | {c.Cost} Рублей");

if (current != null) { if (current.CategoryId == c.Id) CategoriesDropDown.SelectedIndex = i; }

i++;

}

}

this.CategoriesDropDown.ItemsSource = categroies;

if (current != null)

{

AppClientId.Text = current.ClientId.ToString();

AppOpenDate.Text = current.OpenDate;

AppCloseDate.Text = current.CloseDate;

AppDescription.Text = current.Description;

AppTitle.Text = current.Title;

}

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

if (!int.TryParse(AppClientId.Text, out int client\_id)) MessageBox.Show("Ошибка id клиента");

else if (!MainWindow.FKExistsCheck(AppClientId.Text, "Client")) MessageBox.Show("Ошибка! Несуществующий id пользователя");

else if (AppOpenDate.Text != "Не взято" && !MainWindow.DateCheck(AppOpenDate.Text)) MessageBox.Show("Ошибка! Неверный формат даты");

else

{

using (var db = new MobileContext())

{

string cat\_title = CategoriesDropDown.Text.Split(" | ")[0];

int cat\_id = db.Categories.FirstOrDefault(x => x.Title == cat\_title).Id;

var app = new Models.App

{

ClientId = client\_id,

CategoryId = cat\_id,

Description = AppDescription.Text,

Title = AppTitle.Text,

OpenDate = AppOpenDate.Text,

CloseDate = ""

};

if (current != null) app.Id = current.Id;

db.Apps.AddOrUpdate(app);

db.SaveChanges();

}

MessageBox.Show("Заявка успешно добавлена");

this.Close();

}

}

}

}