Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Ивановский государственный энергетический университет имени В. И. Ленина».

Кафедра систем управления

Отчет по лабораторному практикуму

**«Проектирование баз данных и разработка приложений в СУБД Microsoft SQL Server»**

Дисциплина: Системы управления базами данных

Вариант 10

Выполнили: студенты группы 2–43

Балашов И.

Ковшов Д.

Проверил: ассистент

Трифонов А. Ю.

Иваново 2025

**Цель работы**

Разработка и реализация базы данных для конкретной предметной области с использованием методов моделирования, таких как создание концептуальной, логической и физической моделей. В рамках работы необходимо определить таблицы пользовательских ролей и их функции, а также установить ограничения на вводимые данные, значения по умолчанию и правила удаления. Также требуется разработать и выполнить запросы на модификацию данных (Insert, Update, Delete) и запросы на выборку с использованием различных операторов и функций SQL, включая агрегацию, подзапросы и объединения. В процессе работы также предстоит создать хранимые процедуры и триггеры для автоматизации обработки данных и обеспечения целостности базы данных.

**Задание**

***Оператор сотовой связи***

Оператор сотовой связи МТС предоставляет услуги сотовой связи физическим и юридическим лицам по всей России. Юридические лица могут подключать только корпоративные тарифы, а физические –только некорпоративные. Для каждого клиента известно количество средств на счету. Для каждого тарифа указаны стоимость перехода на тариф, стоимость минуты разговоров (по городу, для междугородних и международных звонков). Все звонки фиксируются: кто звонит, кому звонят, время звонка, продолжительность разговора (целое количество минут), тип соединения (по городу, междугородний, международный). После окончания разговора со счета звонящего списывается определенная сумма. Каждый клиент вправе запросить распечатку звонков за любой период.

**Проектирование**

Создание модели предметной области начинается с определения абстракций, существующих в реальном мире, то есть ключевых концептуальных объектов, присутствующих в системе. Концептуальная модель представляет собой описание понятий через призму предметной области. Она отражает структуру проектируемой системы на высоком уровне абстракции, не привязанном к её физической реализации. При построении концептуальной модели предпочтительнее излишняя детализация, чем недостаточная проработка, поэтому в неё включаются все понятия предметной области, даже те, которые не планируется использовать при разработке базы данных.

Создадим к нашему заданию таблицу с ролями пользователей и их функциями.

*Таблица 1. Роли пользователей и их функции*

| **Роль** | **Функции** |
| --- | --- |
| **Физическое лицо** | Просмотр собственного баланса, просмотр истории звонков, подключение/смена некорпоративного тарифа |
| **Юридическое лицо** | Просмотр баланса компании, просмотр звонков по всем номерам компании, подключение корпоративных тарифов |
| **Оператор колл-центра** | Просмотр данных клиентов, тарификации, истории звонков, смена тарифов по заявке клиента |
| **Служба биллинга** | Расчёт стоимости звонков, списание средств, ведение лога звонков, формирование отчетов |
| **Администратор БД** | Полный доступ ко всем таблицам и объектам базы данных, управление пользователями и ролями |

Теперь перейдём к созданию диаграмм.

**1) Создание диаграммы классов (рис. 1)**

В UML концептуальная модель изображается в виде статической диаграммы классов без указания операций. Такая диаграмма описывает типы объектов системы и различные виды статических связей между ними.

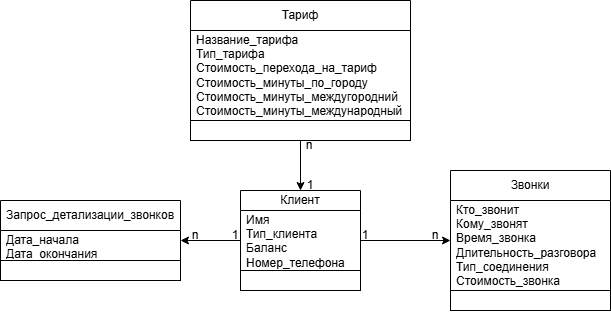


Рис. 1. Диаграмма классов.

**2) Создание концептуальной модели базы данных (рис. 2)**

Концептуальная модель базы данных описывает объекты предметной области, их атрибуты и взаимосвязи между ними в той степени, в которой они подлежат непосредственному сохранению в базе данных системы.

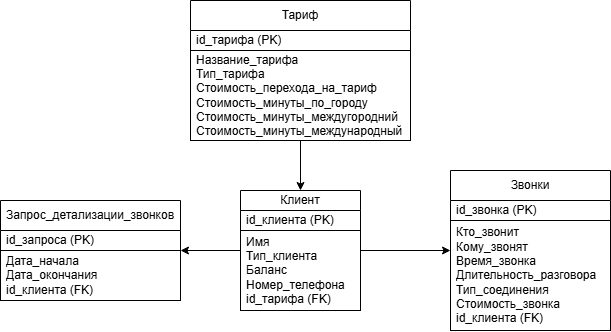


Рис. 2. Концептуальная модель базы данных

**3) Создание физической модели базы данных (рис. 3)**

Физическая модель включает в себя все необходимые детали для конкретной СУБД, обеспечивающие эффективное определение данных, такие как наименования столбцов, типы данных, описание первичных и внешних ключей и другие элементы.

Изображение выглядит как текст, снимок экрана, диаграмма, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рис.3. Физическая модель базы данных.

**Создание БД в Microsoft SQL Server**

Чтобы диаграмма была построена в Microsoft SQL Server, нужно создать БД и создать там таблицы, с необходимыми нам значениями. Будем использовать SQL-запросы.

Создаём БД с названием MTS (рис. 4).

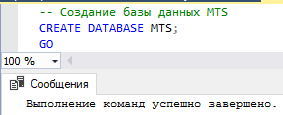


Рис. 4. Создание БД

Теперь перейдём к созданию необходимых нам таблиц с параметрами.

Таблица для Тарифов (рис. 5)

Изображение выглядит как текст, Шрифт, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 5. Структура таблицы Тарифов

Таблица для Клиентов (рис. 6)

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 6. Структура таблицы Клиентов

Таблица для Звонков (рис. 7)

Изображение выглядит как текст, Шрифт, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 7. Структура таблицы Звонков

Таблица для Запросов детализации звонков (рис. 8)

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 8. Структура таблицы Детализации звонков

Выполняем запрос, для БД MTS (рис. 9).

Изображение выглядит как текст, электроника, снимок экрана, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 9. Обозреватель объектов

Как видим, в обозревателе решений появилась БД с нашими таблицами.

Теперь можем сделать диаграмму в Microsoft SQL Server, и посмотреть, что получилось (рис. 10).

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 10. Диаграмма в Microsoft SQL Server

**Описание ограничений**

При создании таблиц мы использовали следующие ограничения:

1. **Значения по умолчанию. DEFAULT**

У нас будет начальный баланс у клиента 0, и по умолчанию стоимость совершённого звонка, также будет 0 (рис. 11).





Рис. 11 Значения по умолчанию.

1. **Ограничения на вводимые данные. CHECK**

У нас будут ограничения на вводимые данные для Типа тарифа ('Корпоративный', 'Некорпоративный'), Типа клиента ('Физическое', 'Юридическое') и Типа звонка ('По городу', 'Междугородний', 'Международный') (рис. 12.)







Рис. 12. Ограничения на вводимые данные

1. **Правила удаления. DELETE**





Рис. 13. ON DELETE CASCADE

Здесь в таблицах Calls и Call\_Detail\_Request используется ON DELETE CASCADE для связи с Client. Это значит: если удалить клиента из таблицы Client, то все его звонки (Calls) и запросы на детализацию (Call\_Detail\_Request) тоже автоматически удалятся.



Рис. 14. ON DELETE SET NULL

Здесь в таблице Client установлено правило ON DELETE SET NULL для tariff\_id. Это значит: если удалить тариф из таблицы Tariff, у всех клиентов, которые были на этом тарифе, поле tariff\_id станет NULL, но сами записи клиентов останутся в таблице.

**Запросы на модификацию данных**

*Таблица 2. Команды на модификацию данных*

| **Команда** | **Назначение** |
| --- | --- |
| INSERT | Добавить новую запись |
| UPDATE | Изменить существующую запись |
| DELETE | Удалить запись |
| SELECT | Получить записи из таблицы |

1. **INSERT**

**Добавление тарифа** (рис. 15)

Изображение выглядит как текст, Шрифт, линия, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 15. Добавление тарифа

**Добавление клиента** (рис 16.)

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 16. Добавление клиента

**Добавление звонка** (рис. 17)

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 17. Добавление звонка

Просмотрим теперь содержание таблиц.

Для этого выполним следующий запрос (рис. 18)

Изображение выглядит как текст, электроника, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 18. Просмотр данных в таблицах

Результат (рис. 19).

Изображение выглядит как текст, снимок экрана, дисплей, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 19. Данные в таблицах.

1. UPDATE

**Обновить баланс клиента.** Этот запрос увеличит баланс клиента с номером +79001234567 на 200 рублей (рис. 20).

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 20. Обновление баланса клиента.

Проверим (рис. 21).





Рис. 21. Баланс до/после

**Изменить стоимость перехода на тариф.** Этот запрос изменяет стоимость перехода (switch\_cost) для тарифа с tariff\_id = 1 на 100 рублей (рис. 22).



Рис. 22. Изменение стоимости перехода на тариф.

Проверка (рис. 23).





Рис. 23. Стоимость перехода до/после

**Изменить тариф у клиента**. Этот запрос меняет тариф клиента **только если** тип клиента и тип тарифа совпадают (рис. 24):

Физическое лицо → Некорпоративный тариф

Юридическое лицо → Корпоративный тариф

Если не подходит — тариф не изменится.

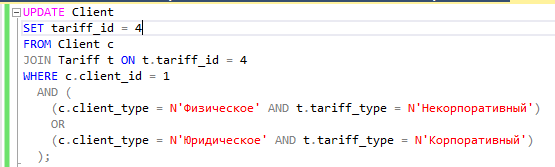


Рис. 24. Изменение тарифа у клиента.

Проверка (рис. 25).

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, дисплей, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 25. Изменение тарифа до/после

1. DELETE

**Удаление всех звонков у клиента** (рис. 26).

Изображение выглядит как текст, Шрифт, снимок экрана, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 26. Удаление звонков у клиента.

**Удаление клиента по номеру телефона** (рис. 27).



Рис. 27. Удаление клиента по номеру телефона.

**Удаление тарифа** (рис. 28).



Рис. 28. Удаление тарифа.

Таблицы до выполнения запросов (рис. 29).

Изображение выглядит как текст, снимок экрана, дисплей, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 29. Таблицы до выполнения запросов.

После выполнения этих запросов (рис. 30).

Изображение выглядит как текст, электроника, снимок экрана, дисплей

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 30. Таблицы после выполнения команд.

Всё сработало корректно.

**Запросы на выборку данных. 10 запросов SELECT**

Перед выполнением этих запросов заполним таблицы.

Заполнение Тарифов (рис. 31)

Изображение выглядит как текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 31. Заполнение тарифов

Теперь заполним Пользователей (рис. 32)

Изображение выглядит как текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 32. Заполнение пользователей

Заполнение Звонков (рис. 33).

Изображение выглядит как текст, снимок экрана

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 33. Заполнение звонков

И заполним таблицу детализации звонков (рис. 34).

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 34. Заполнение детализации звонков

Выведем результат, что получилось (рис. 35).

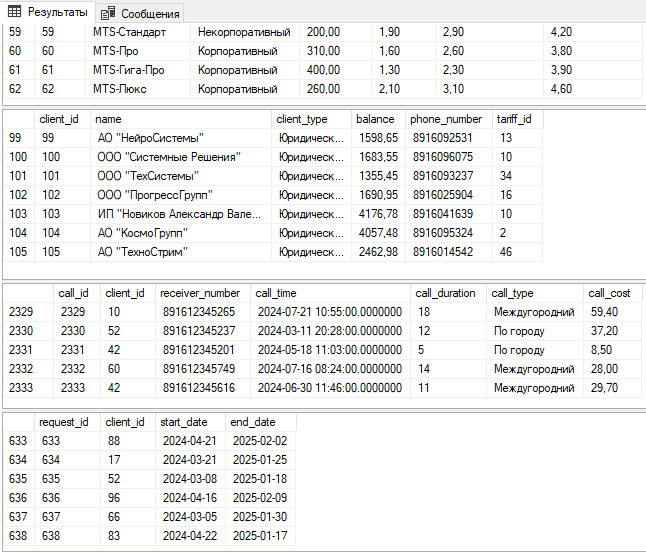


Рис. 35. Заполненные таблицы.

Теперь перейдём к 10 запросам SELECT.

1. **WHERE**

Изображение выглядит как текст, электроника, снимок экрана, дисплей

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 36. WHERE

Запрос выводит список клиентов, у которых баланс превышает 1500. Условие WHERE используется для фильтрации данных, чтобы выбрать только те записи, которые соответствуют указанному критерию.

1. **ORDER BY**

Изображение выглядит как текст, электроника, снимок экрана, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 37. ORDER BY

Запрос выводит все данные из таблицы Tariff, сортируя результаты по стоимости перехода на тариф (switch\_cost) от наименьшей к наибольшей.

1. **Функция агрегации**

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 38. Функция агрегации

Запрос подсчитывает общую сумму всех звонков (их стоимости) в таблице Calls и выводит её как total\_revenue (общий доход).

1. **GROUP BY + HAVING**

Изображение выглядит как текст, снимок экрана, дисплей, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 39. GROUP BY + HAVING

Запрос выводит количество звонков для каждого типа звонка, но только для тех типов, где количество звонков больше или равно 750.

1. **CAST**

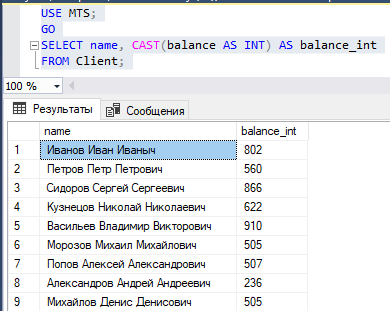


Рис. 40. CAST

Запрос выводит имена клиентов и их баланс, при этом баланс преобразуется в целое число (отбрасываются дробные значения, если они есть).

1. **CASE**

Изображение выглядит как текст, электроника, снимок экрана, дисплей

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 41. CASE

Запрос выводит идентификатор звонка, стоимость звонка и категорию звонка (Дешевый, Средний или Дорогой) в зависимости от стоимости звонка.

1. **EXISTS**

Изображение выглядит как текст, электроника, снимок экрана, дисплей

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 42. EXISTS

Запрос выводит тарифы, которые используются хотя бы одним клиентом. Если у тарифа есть хотя бы один клиент, он будет включён в результат.

1. **Подзапрос**

Изображение выглядит как текст, электроника, снимок экрана, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 43. Подзапрос

Запрос выводит всех клиентов, чьи балансы больше, чем средний баланс всех клиентов в базе данных.

1. **JOIN**

Изображение выглядит как текст, электроника, снимок экрана, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 44. JOIN

Запрос выводит имена клиентов, их номера телефонов и название их тарифов. Для этого объединяются данные из таблиц Client и Tariff по полю tariff\_id.

1. **UNION**

Изображение выглядит как текст, электроника, снимок экрана, дисплей

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 45. UNION

Запрос выводит уникальные номера телефонов, которые присутствуют либо среди клиентов (phone\_number из Client), либо среди получателей звонков (receiver\_number из Calls).

**Хранимые процедуры**

1. Хранимая процедура GetMostActiveClients позволяет вывести информацию о самых активных клиентах за определённый период. Для этого нужно передать дату начала и окончания периода, а также количество клиентов, которых нужно вывести в результатах. Процедура возвращает топ клиентов, совершивших наибольшее количество звонков, и сортирует их по убыванию активности.



Рис. 46. Запрос к процедуре 1.

Этот запрос вернёт топ-5 самых активных клиентов, которые совершили наибольшее количество звонков в период с 1 января 2024 года по 1 августа 2024 года.

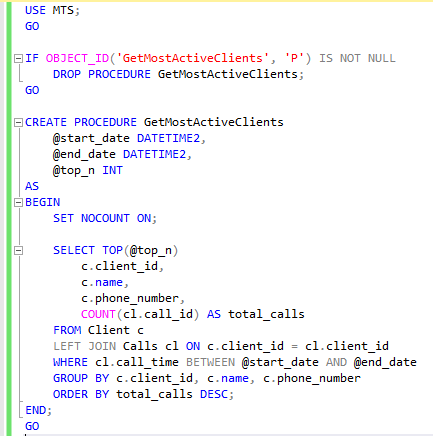


Рис. 47. Процедура 1.

Результат:

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 48. Результат выполнения 1-ой процедуры

1. Процедура GetClientCallCostAnalysis анализирует стоимость звонков клиентов за определённый период, который задаётся двумя параметрами — начальной и конечной датой. Она собирает информацию о стоимости звонков каждого клиента, разделяя их на городские, междугородние и международные. Для каждого клиента рассчитывается общая стоимость всех его звонков в указанный период. Результаты выводятся в виде списка с именем клиента, его номером телефона и детализацией стоимости звонков по категориям, с указанием общей стоимости. Данные сортируются по общей стоимости звонков в порядке убывания, то есть сначала идут клиенты с наибольшими расходами.

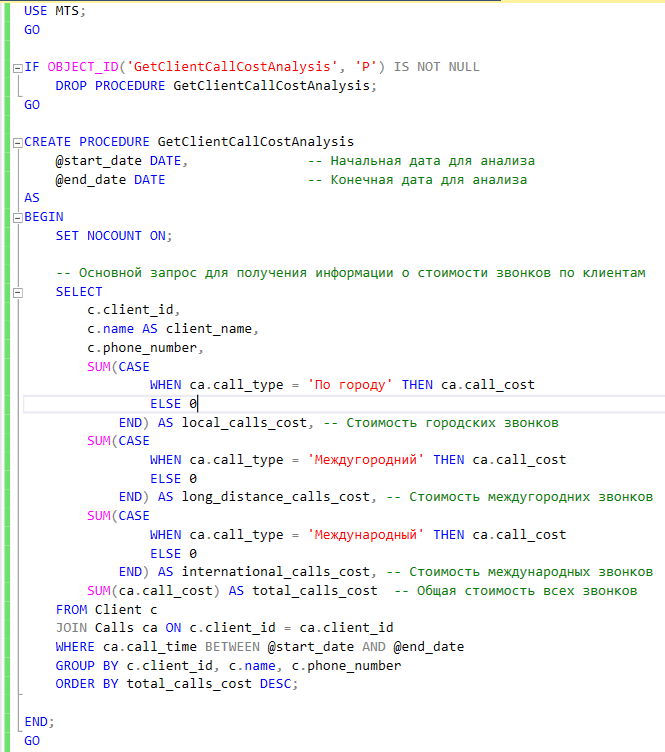


Рис. 49. Процедура 2.

Изображение выглядит как текст, снимок экрана, число, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 50. Результат запроса к процедуре 2.

**Триггеры**

Теперь создадим 2 триггера.

**Триггер 1. trg\_PreventTariffChangeWithNegativeBalance.**

Этот триггер срабатывает после обновления записи в таблице Client.  
Его задача — запретить изменение тарифа (tariff\_id), если у клиента отрицательный баланс.

Как работает: Если при обновлении обнаружено, что тариф изменён (tariff\_id до и после разные), и при этом значение поля balance меньше нуля — происходит откат транзакции и вывод сообщения об ошибке: 'Нельзя сменить тариф при отрицательном балансе.'

Таким образом, триггер обеспечивает контроль целостности логики: клиент с отрицательным балансом не может перейти на другой тариф.

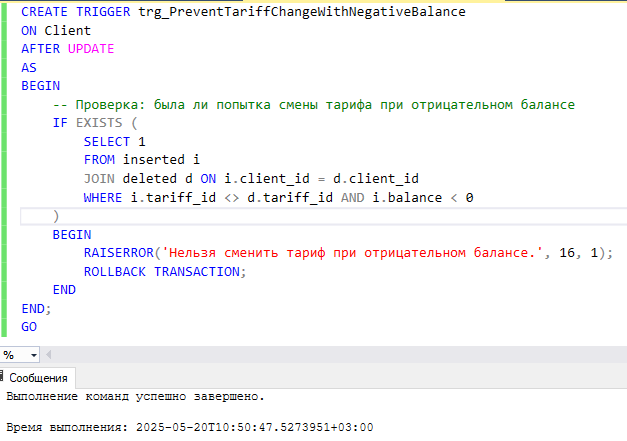


Рис. 51. Триггер 1

**Триггер 2. trg\_PreventCallWithNegativeBalance.**

Этот триггер срабатывает после вставки записи в таблицу Calls.  
Он блокирует добавление звонка, если у клиента, который совершает звонок, отрицательный баланс.

Как работает: Триггер проверяет каждую вставляемую строку. Если у соответствующего клиента (client\_id) баланс меньше нуля, выполнение вставки отменяется с помощью ROLLBACK, и выводится сообщение об ошибке: 'Нельзя совершить звонок при отрицательном балансе.'

Этот механизм предотвращает возможность совершения звонков при нехватке средств, что обеспечивает реалистичное поведение системы биллинга.

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 52. Триггер 2

Проверим триггер. У нас у клиента с id 21 – отрицательный баланс (рис. 50).

Изображение выглядит как текст, снимок экрана, число, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 53. Клиент 21

Выполним запрос на изменение тарифа (рис. 51).

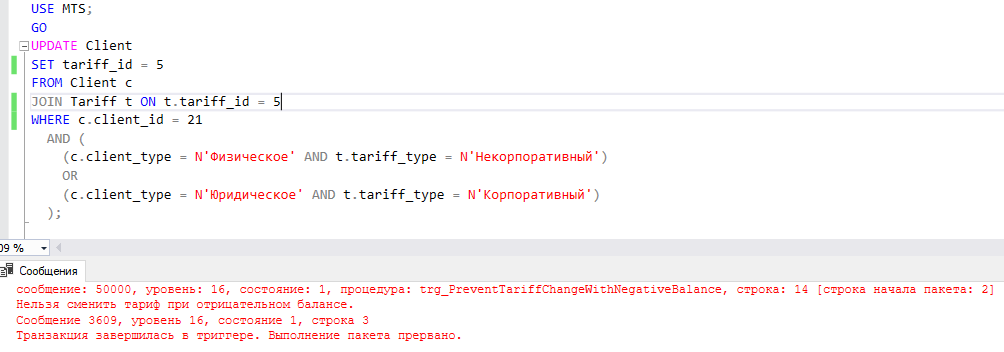


Рис. 54. Проверка триггера

Как видим, триггер сработал.

**Приложение**

В Microsoft Visual Studio создадим Приложение Windows Forms (.Net Framework). Назовём проект MTS и запустим проект (рис. 55).

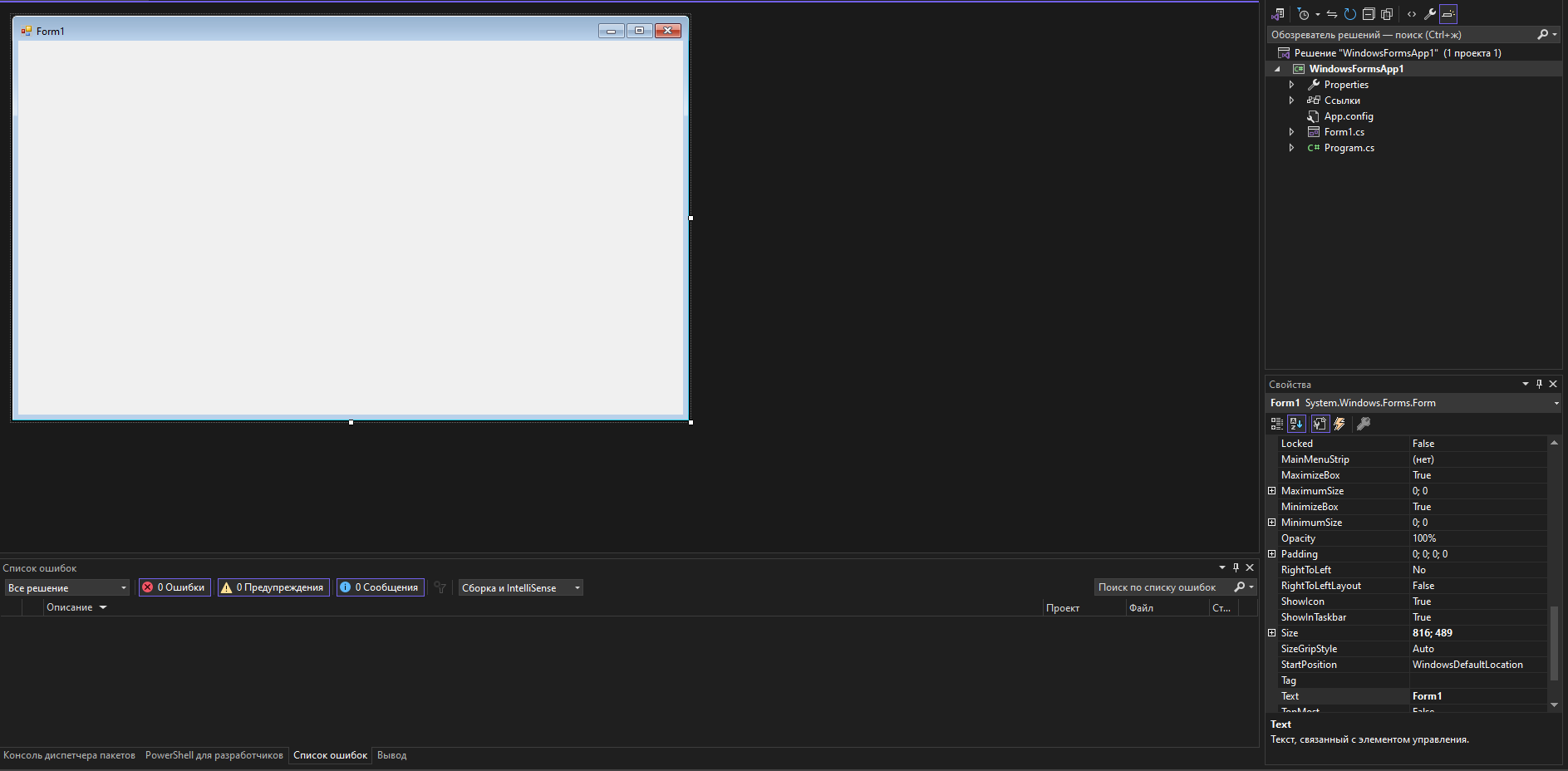


Рис. 55.

Теперь панель элементов добавим 4 DataGridView и 1 Button, при нажатии на которую будут значения в таблицах обновляться (рис. 56)

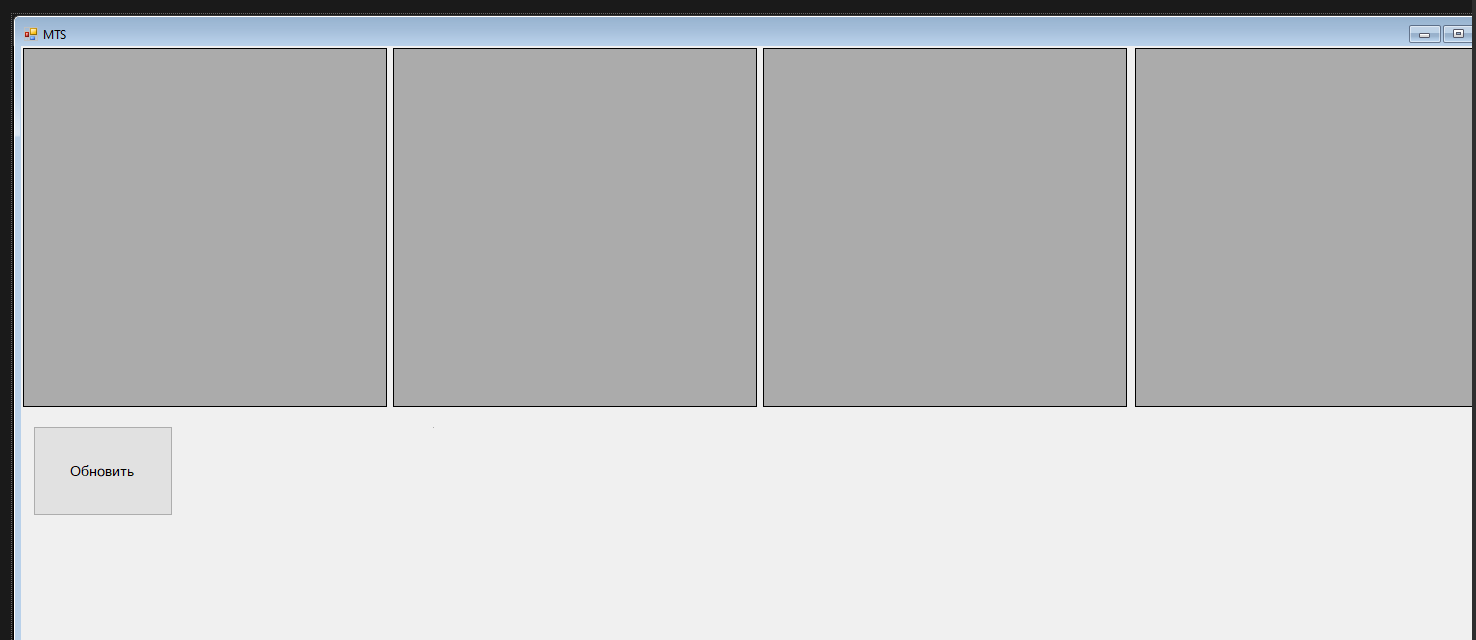
.

Рис. 56.

Добавим ещё 2 формы, в 1-ой мы будем добавлять нового пользователя, а во 2-ой делать запрос на детализацию звонков за определённый период.

Итоговая 1-я форма (рис. 57).

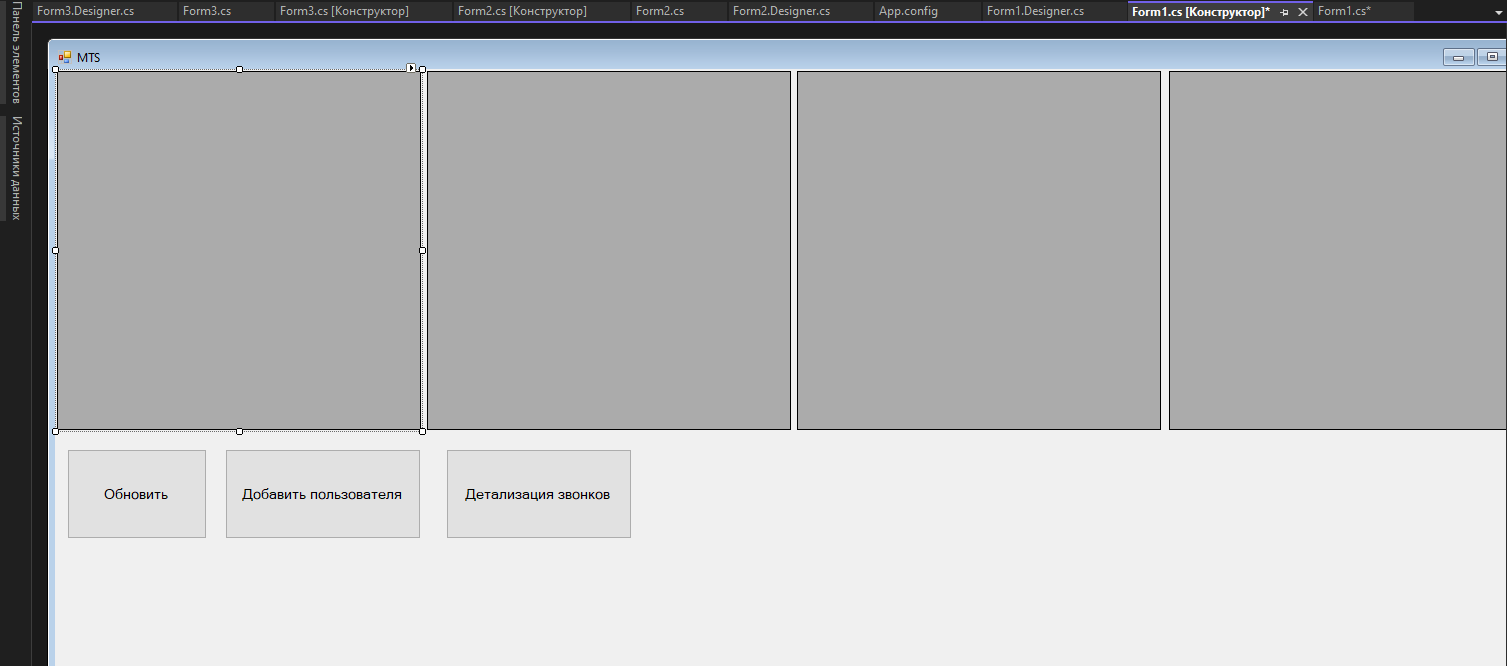


Рис. 57.

Вторая форма (рис. 58)

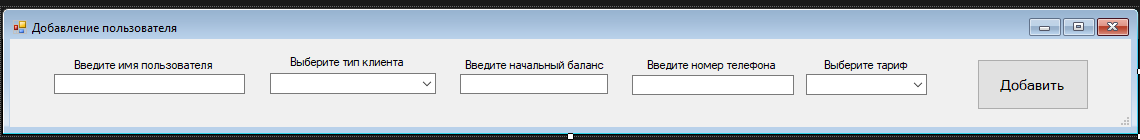


Рис. 58.

Третья форма (рис. 59).

Изображение выглядит как текст, снимок экрана, программное обеспечение, компьютер

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 59.

У всех объектов есть свои имена, которые уже используются в коде.

Код для начальной формы.

*Таблица 3. Код Form1.cs*

|  |
| --- |
| using System;  using System.Data;  using System.Data.SqlClient;  using System.Windows.Forms;  namespace MTS  {  public partial class MTS : Form  {  // Строка подключения с аутентификацией Windows  private string connectionString = @"Server=localhost;Database=MTS;Trusted\_Connection=True;";  public MTS()  {  InitializeComponent();  this.StartPosition = FormStartPosition.CenterScreen;  // Привязка обработчика к кнопке  buttonRefresh.Click += ButtonRefresh\_Click;  buttonAddClient.Click += buttonAddClient\_Click;  buttonCallDetails.Click += buttonCallDetails\_Click;  }  private void MTS\_Load(object sender, EventArgs e)  {  LoadAllData();  }  private void ButtonRefresh\_Click(object sender, EventArgs e)  {  LoadAllData();  }  private void LoadAllData()  {  LoadData("SELECT \* FROM Tariff", dataGridViewTariff);  LoadData("SELECT \* FROM Client", dataGridViewClient);  LoadData("SELECT \* FROM Calls", dataGridViewCalls);  LoadData("SELECT \* FROM Call\_Detail\_Request", dataGridViewCallDetailRequest);  }  private void LoadData(string query, DataGridView dgv)  {  try  {  using (SqlConnection conn = new SqlConnection(connectionString))  {  SqlDataAdapter adapter = new SqlDataAdapter(query, conn);  DataTable dt = new DataTable();  adapter.Fill(dt);  dgv.DataSource = dt;  }  }  catch (Exception ex)  {  MessageBox.Show($"Ошибка загрузки данных: {ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);  }  }  private void buttonAddClient\_Click(object sender, EventArgs e)  {  using (AddClientForm addClientForm = new AddClientForm())  {  if (addClientForm.ShowDialog() == DialogResult.OK)  {  // После добавления клиента обновляем данные на главной форме  LoadData("SELECT \* FROM Client", dataGridViewClient);  }  }  }  private void buttonCallDetails\_Click(object sender, EventArgs e)  {  using (CallDetailsForm form = new CallDetailsForm())  {  form.StartPosition = FormStartPosition.CenterScreen;  form.ShowDialog();  // Обновляем таблицу после возможного добавления запроса  LoadData("SELECT \* FROM Call\_Detail\_Request", dataGridViewCallDetailRequest);  }  }  }  } |

Код для второй формы с добавлением пользователя.

*Таблица 4. Код Form2.cs*

|  |  |
| --- | --- |
| using System;  using System.Data.SqlClient;  using System.Linq;  using System.Windows.Forms;  namespace MTS  {  public partial class AddClientForm : Form  {  private string connectionString = @"Server=localhost;Database=MTS;Trusted\_Connection=True;";  public AddClientForm()  {  InitializeComponent();  this.StartPosition = FormStartPosition.CenterScreen;  comboBoxClientType.SelectedIndexChanged += ComboBoxClientType\_SelectedIndexChanged;  Load += AddClientForm\_Load;  buttonSave.Click += ButtonSave\_Click;  // Настройка маски для maskedTextBoxPhone  maskedTextBoxPhone.Mask = "8 (999) 000-00-00";  maskedTextBoxPhone.SkipLiterals = true; // курсор пропускает скобки и пробелы  maskedTextBoxPhone.Text = "8 "; // +7 по умолчанию  }  private void AddClientForm\_Load(object sender, EventArgs e)  {  comboBoxClientType.Items.Clear();  comboBoxClientType.Items.AddRange(new string[] { "Физическое", "Юридическое" });  comboBoxClientType.SelectedIndex = 0;  }  private void ComboBoxClientType\_SelectedIndexChanged(object sender, EventArgs e)  {  string selectedType = comboBoxClientType.SelectedItem.ToString();  LoadTariffsForClientType(selectedType);  }  private void LoadTariffsForClientType(string clientType)  {  string tariffType = (clientType == "Юридическое") ? "Корпоративный" : "Некорпоративный";  try  {  using (SqlConnection conn = new SqlConnection(connectionString))  {  conn.Open();  string query = "SELECT tariff\_id, tariff\_name FROM Tariff WHERE tariff\_type = @tariffType";  using (SqlCommand cmd = new SqlCommand(query, conn))  {  cmd.Parameters.AddWithValue("@tariffType", tariffType);  using (SqlDataReader reader = cmd.ExecuteReader())  {  comboBoxTariff.Items.Clear();  while (reader.Read())  {  comboBoxTariff.Items.Add(new ComboboxItem()  {  Text = reader["tariff\_name"].ToString(),  Value = reader["tariff\_id"]  });  }  }  }  }  if (comboBoxTariff.Items.Count > 0)  comboBoxTariff.SelectedIndex = 0;  }  catch (Exception ex)  {  MessageBox.Show("Ошибка загрузки тарифов: " + ex.Message);  }  }  private void ButtonSave\_Click(object sender, EventArgs e)  {  // Проверяем имя  if (string.IsNullOrWhiteSpace(textBoxName.Text))  {  MessageBox.Show("Введите имя клиента");  return;  }  // Проверяем выбран тариф  if (comboBoxTariff.SelectedItem == null)  {  MessageBox.Show("Выберите тариф");  return;  }  // Проверяем баланс  if (!decimal.TryParse(textBoxBalance.Text, out decimal balance))  {  MessageBox.Show("Введите корректный баланс");  return;  }  // Получаем номер телефона из maskedTextBox и очищаем от лишних символов  string phoneRaw = maskedTextBoxPhone.Text;  string phone = "+" + new string(phoneRaw.Where(char.IsDigit).ToArray());  if (!IsValidPhoneNumber(phone))  {  MessageBox.Show("Номер телефона должен быть в формате +7 (XXX) XXX-XX-XX");  return;  } | // Проверяем уникальность номера  if (PhoneExists(phone))  {  MessageBox.Show("Клиент с таким номером телефона уже существует");  return;  }  string name = textBoxName.Text.Trim();  string clientType = comboBoxClientType.SelectedItem.ToString();  int tariffId = (int)((ComboboxItem)comboBoxTariff.SelectedItem).Value;  try  {  using (SqlConnection conn = new SqlConnection(connectionString))  {  conn.Open();  string insertQuery = @"INSERT INTO Client (name, client\_type, balance, phone\_number, tariff\_id)  VALUES (@name, @client\_type, @balance, @phone\_number, @tariff\_id)";  using (SqlCommand cmd = new SqlCommand(insertQuery, conn))  {  cmd.Parameters.AddWithValue("@name", name);  cmd.Parameters.AddWithValue("@client\_type", clientType);  cmd.Parameters.AddWithValue("@balance", balance);  cmd.Parameters.AddWithValue("@phone\_number", phone);  cmd.Parameters.AddWithValue("@tariff\_id", tariffId);  cmd.ExecuteNonQuery();  }  }  MessageBox.Show("Клиент успешно добавлен");  this.DialogResult = DialogResult.OK;  this.Close();  }  catch (Exception ex)  {  MessageBox.Show("Ошибка при добавлении клиента: " + ex.Message);  }  }  private bool IsValidPhoneNumber(string phone)  {  // Проверяем, что строка в формате 8 и 11 цифр (8 + 10)  if (phone.Length != 11) return false;  if (!phone.StartsWith("8")) return false;  for (int i = 2; i < phone.Length; i++)  {  if (!char.IsDigit(phone[i]))  return false;  }  return true;  }  private bool PhoneExists(string phone)  {  try  {  using (SqlConnection conn = new SqlConnection(connectionString))  {  conn.Open();  string query = "SELECT COUNT(\*) FROM Client WHERE phone\_number = @phone";  using (SqlCommand cmd = new SqlCommand(query, conn))  {  cmd.Parameters.AddWithValue("@phone", phone);  int count = (int)cmd.ExecuteScalar();  return count > 0;  }  }  }  catch  {  // В случае ошибки считаем, что номер существует,  // чтобы избежать дублирования  return true;  }  }  }  // Класс для удобного хранения пары текст/значение в ComboBox  public class ComboboxItem  {  public string Text { get; set; }  public object Value { get; set; }  public override string ToString()  {  return Text;  }  }  } |

Код для третьей формы с запросом звонков от пользователя за определённый период.

*Таблица 5. Код Form3.cs*

|  |  |
| --- | --- |
| using System;  using System.Data.SqlClient;  using System.Linq;  using System.Windows.Forms;  namespace MTS  {  public partial class CallDetailsForm : Form  {  private string connectionString = @"Server=localhost;Database=MTS;Trusted\_Connection=True;";  public CallDetailsForm()  {  InitializeComponent();  this.StartPosition = FormStartPosition.CenterScreen;  // Настройка маски  maskedTextBoxPhone.Mask = "8 (000) 000-00-00";  maskedTextBoxPhone.Text = "8 ";  maskedTextBoxPhone.TextChanged += MaskedTextBoxPhone\_TextChanged;  listBoxSuggestions.Visible = false;  listBoxSuggestions.Click += ListBoxSuggestions\_Click;  buttonSubmitRequest.Click += buttonSubmitRequest\_Click;  }  private void MaskedTextBoxPhone\_TextChanged(object sender, EventArgs e)  {  string rawInput = new string(maskedTextBoxPhone.Text.Where(char.IsDigit).ToArray());  if (rawInput.Length < 1)  {  listBoxSuggestions.Visible = false;  return;  }  try  {  using (SqlConnection conn = new SqlConnection(connectionString))  {  conn.Open();  string query = "SELECT phone\_number FROM Client WHERE phone\_number LIKE @pattern";  using (SqlCommand cmd = new SqlCommand(query, conn))  {  cmd.Parameters.AddWithValue("@pattern", rawInput + "%");  using (SqlDataReader reader = cmd.ExecuteReader())  {  listBoxSuggestions.Items.Clear();  while (reader.Read())  {  listBoxSuggestions.Items.Add(reader.GetString(0));  }  }  }  }  listBoxSuggestions.Visible = listBoxSuggestions.Items.Count > 0;  }  catch  {  listBoxSuggestions.Visible = false;  }  }  private void ListBoxSuggestions\_Click(object sender, EventArgs e)  {  if (listBoxSuggestions.SelectedItem != null)  {  string selectedPhone = listBoxSuggestions.SelectedItem.ToString();  maskedTextBoxPhone.Text = FormatPhone(selectedPhone);  listBoxSuggestions.Visible = false;  }  }  private string FormatPhone(string phone)  {  if (phone.Length != 11) return phone;  return $"8 ({phone.Substring(1, 3)}) {phone.Substring(4, 3)}-{phone.Substring(7, 2)}-{phone.Substring(9, 2)}";  }  private void buttonSubmitRequest\_Click(object sender, EventArgs e)  {  string phoneRaw = maskedTextBoxPhone.Text;  string phone = "8" + new string(phoneRaw.Where(char.IsDigit).Skip(1).ToArray());  if (phone.Length != 11 || !phone.StartsWith("8"))  {  MessageBox.Show("Введите корректный номер телефона в формате 8 (XXX) XXX-XX-XX");  return;  }  DateTime startDate = dateTimePickerStart.Value.Date;  DateTime endDate = dateTimePickerEnd.Value.Date;  if (endDate < startDate)  {  MessageBox.Show("Конечная дата не может быть раньше начальной");  return;  } | int? clientId = GetClientIdByPhone(phone);  if (clientId == null)  {  MessageBox.Show("Клиент с таким номером не найден");  return;  }  try  {  using (SqlConnection conn = new SqlConnection(connectionString))  {  conn.Open();  // Вставка запроса на детализацию  string insertQuery = @"INSERT INTO Call\_Detail\_Request (client\_id, start\_date, end\_date)  VALUES (@client\_id, @start\_date, @end\_date)";  using (SqlCommand insertCmd = new SqlCommand(insertQuery, conn))  {  insertCmd.Parameters.AddWithValue("@client\_id", clientId.Value);  insertCmd.Parameters.AddWithValue("@start\_date", startDate);  insertCmd.Parameters.AddWithValue("@end\_date", endDate);  insertCmd.ExecuteNonQuery();  }  // Получение статистики по звонкам клиента за указанный период  string statsQuery = @"  SELECT  COUNT(\*) AS CallCount,  ISNULL(SUM(call\_cost), 0) AS TotalCost  FROM Calls  WHERE client\_id = @client\_id  AND call\_time >= @start\_date  AND call\_time <= @end\_date";  using (SqlCommand statsCmd = new SqlCommand(statsQuery, conn))  {  statsCmd.Parameters.AddWithValue("@client\_id", clientId.Value);  statsCmd.Parameters.AddWithValue("@start\_date", startDate);  statsCmd.Parameters.AddWithValue("@end\_date", endDate);  using (SqlDataReader reader = statsCmd.ExecuteReader())  {  if (reader.Read())  {  int callCount = reader.GetInt32(0);  decimal totalCost = reader.GetDecimal(1);  MessageBox.Show(  $"Запрос на детализацию успешно добавлен.\n\n" +  $"Период: {startDate:dd.MM.yyyy} – {endDate:dd.MM.yyyy}\n" +  $"Количество звонков: {callCount}\n" +  $"Общая стоимость звонков: {totalCost:C}",  "Успешно",  MessageBoxButtons.OK,  MessageBoxIcon.Information);  }  }  }  this.DialogResult = DialogResult.OK;  this.Close();  }  }  catch (Exception ex)  {  MessageBox.Show("Ошибка при добавлении запроса: " + ex.Message);  }  }  private int? GetClientIdByPhone(string phone)  {  try  {  using (SqlConnection conn = new SqlConnection(connectionString))  {  conn.Open();  string query = "SELECT client\_id FROM Client WHERE phone\_number = @phone";  using (SqlCommand cmd = new SqlCommand(query, conn))  {  cmd.Parameters.AddWithValue("@phone", phone);  var result = cmd.ExecuteScalar();  return result != null ? Convert.ToInt32(result) : (int?)null;  }  }  }  catch  {  return null;  }  }  }  } |

Теперь пройдёмся по работе приложения. Запустим приложение (рис. 60).

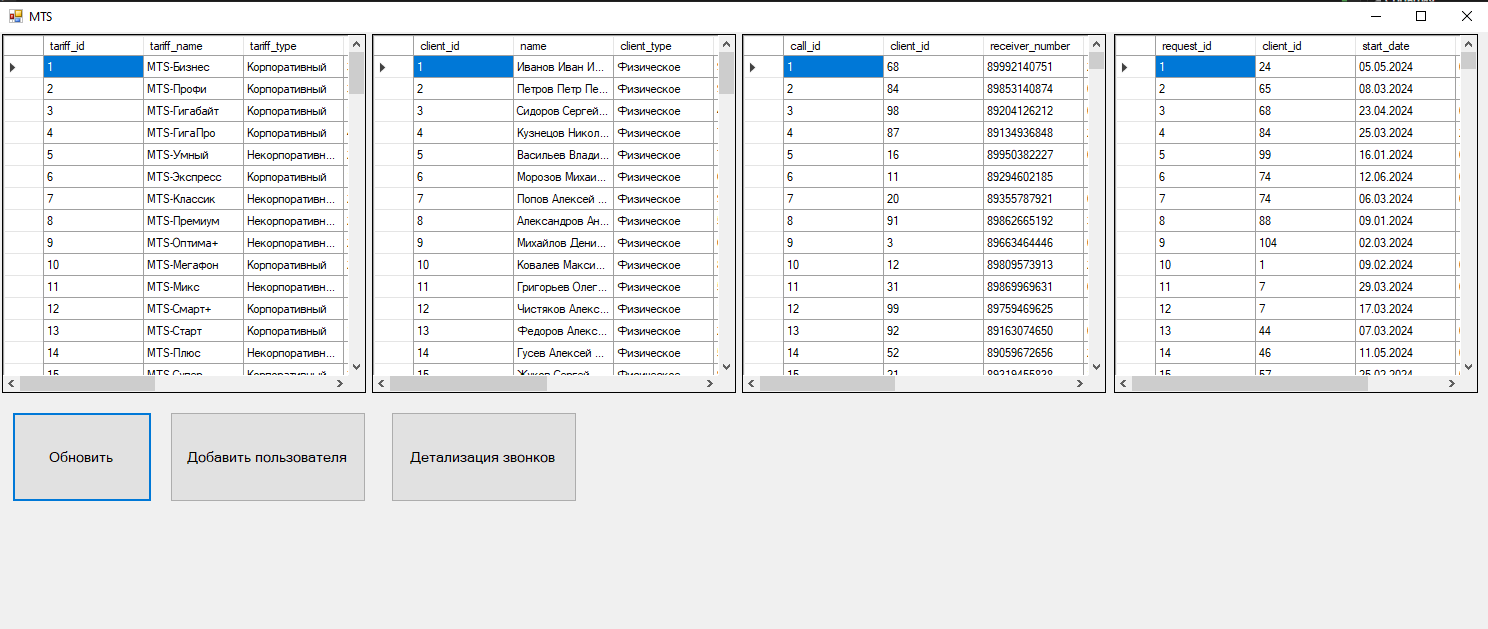


Рис. 60. Вход в приложение

Теперь добавим пользователя (рис. 61).

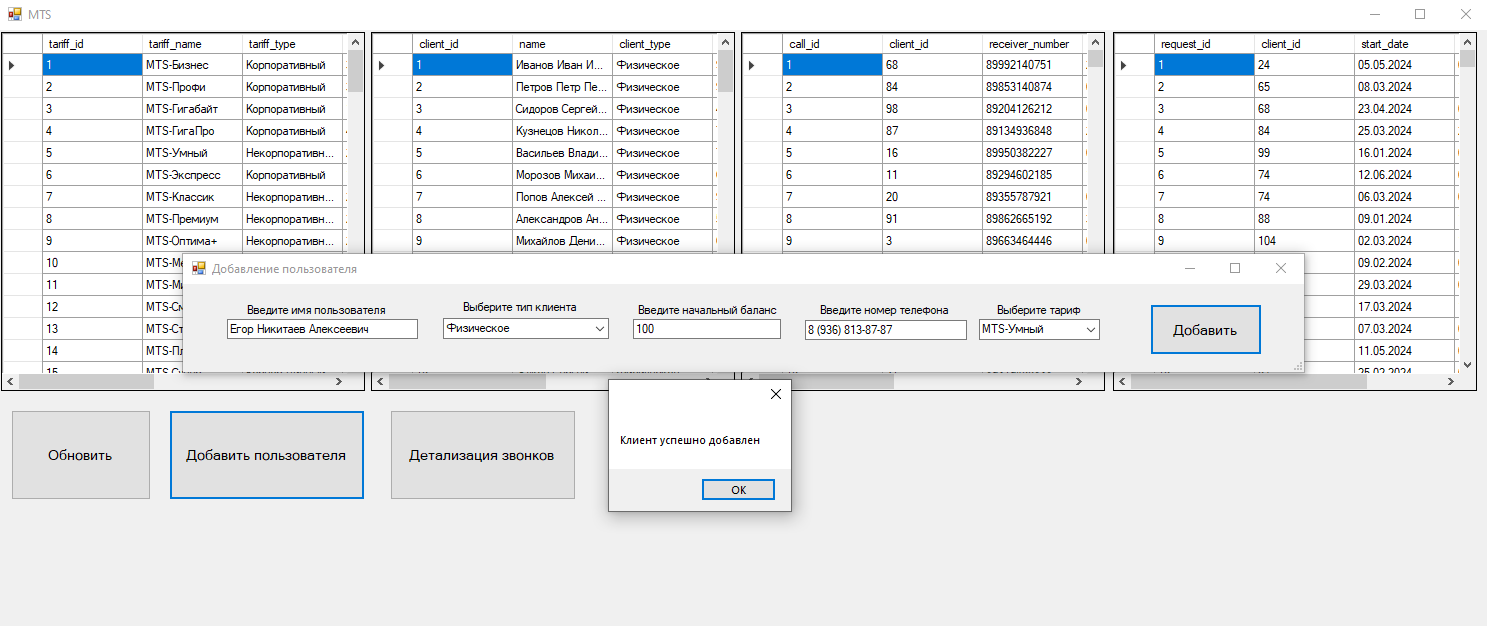


Рис. 61. Добавление пользователя

Просмотрим в таблице (рис. 62).

Изображение выглядит как текст, снимок экрана, число, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 62. Проверка добавленного пользователя

Добавился успешно. Теперь закажем детализацию звонков (рис. 63).

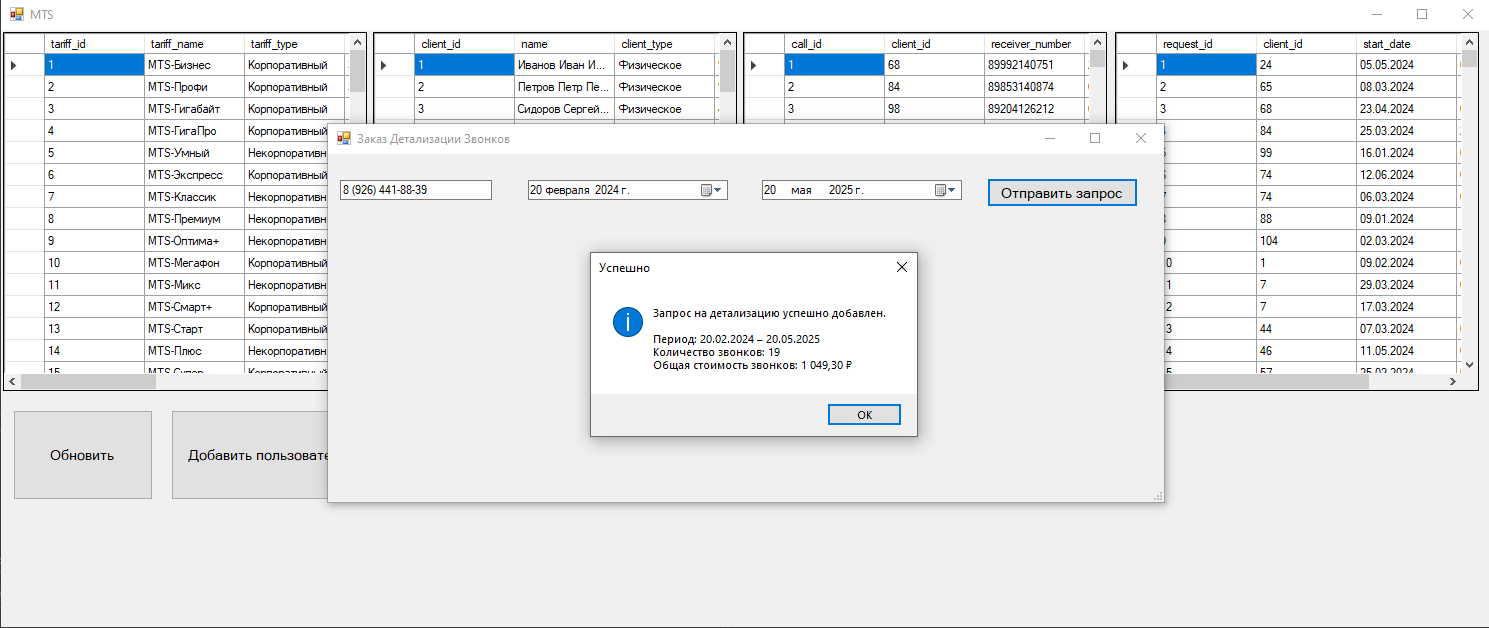


Рис. 63. Заказ детализации

Проверим данный запрос в таблице (рис. 64).

Изображение выглядит как текст, снимок экрана, число, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рис. 64. Проверка детализации запроса

Всё корректно заполнилось.

**Вывод**

В ходе курсового проекта была разработана система для учета клиентов и звонков телекоммуникационной компании. На основе анализа предметной области создана база данных в MS SQL Server с проработанными связями, ограничениями целостности и бизнес-логикой, реализованной через хранимые процедуры и триггеры. Для работы с базой данных создано Windows Forms-приложение на языке C#, позволяющее регистрировать клиентов, выбирать тарифы, оформлять запросы на детализацию звонков и просматривать статистику по ним. Интерфейс предусматривает маску ввода номера телефона, автоподсказки и валидацию данных. Разработка охватывает все основные аспекты построения и использования реляционных баз данных и демонстрирует интеграцию клиентского приложения с серверной частью.