**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**

A logo of a triangle

Description automatically generated

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ЛИПЕЦКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

|  |  |
| --- | --- |
| Институт | компьютерных наук |
| Кафедра | автоматизированных систем управления |

ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

По дисциплине: "Базы данных"

На тему: "Разработка прототипа прикладного приложения для БД"

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ПИ-22-2 | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись, дата | Абрамов М.А. |
| Руководитель | |  |  |  |
| к.т.н., доцент  ученая степень, ученое звание | |  | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись, дата | Алексеев В.А. |

Липецк, 2024 г.

**Цель работы**

Получение первичных навыков разработки WEB для БД, освоение фреймворков для работы с БД.

**Задание**

Реализовать прикладное приложение, обеспечивающее просмотр и редактирование содержимого спроектированной в ходе лабораторного практикума БД. Прототип должен реализовывать функции для поддержки одного бизнес-процесса информационной системы и содержать следующие элементы интерфейса:

1) Меню для навигации по функциям (левое или верхнее). Для мобильных приложений могут использоваться карточки для навигации по функциям.

2) Табличное представление данных с фильтром и пагинацией (для веб-приложений) https://www.uprock.ru/articles/dizayn-tablic-ot-putanicy-k-intuitivno-ponyatnomu-opytu В таблице не должны отображаться искусственные идентификаторы, они должны быть заменены содержательными значениями полей.

3) Редактирование записей в таблице «на месте» - при небольшом числе атрибутов, если горизонтальная прокрутка отсутствует.

4) Редактирование записей в режиме «модального окна» или «боковой панели» (паттерн «контекстный оверлей»). Не должны отображаться искусственные идентификаторы, они должны быть заменены содержательными значениями полей.

5) Выпадающие списки или чек-листы для выбора связанных значений в другой таблице.

6) Календари, переключатели, чекбоксы – там, где это применимо.

7) Отчет с выбором периода и других объектов (например, отчет по определенной категории товаров за год и т.п.).

Прикладное приложение должно содержать минимум 3 табличных представления и минимум 2 формы редактирования в режиме «модального окна» или «боковой панели». Дополнительным плюсом является наличие в отчете графика/диаграммы или возможности экспорта в формат PDF.

**1. Характеристика предметной области**

**1.1. Описание предметной области**

Магазин автодеталей предоставляет большой выбор товаров для автомобилей любой марки.

В рамках деятельности сети собираются данные о клиентах, менеджерах, авто, клиентах и поставщиков. Эти данные хранятся в базе данных и используются для улучшения качества предоставляемых услуг, а также для аналитики и принятия управленческих решений.

**1.2. Автоматизируемые бизнес-процессы**

В процессе автоматизации предусмотрены следующие ключевые бизнес-процессы:

- Управление данными: изменение и удаление;

- Учет клиентов: управление контактной информацией;

- Управление персоналом: учет данных менеджеров, включая их ФИО и дату начала работы;

- Формирование отчетности: генерация отчетов по работе менеджеров и сумме заказов клиентов за указанный период.

**1.3. Характеристика пользователей ИС**

Информационная система предназначена для использования следующими категориями пользователей:

- Клиенты:

- Создание заказов;

- Администраторы:

- Управление данными;

- Формирование отчетов для внутреннего пользования.

- Владелец сети:

- Получение аналитической информации и отчетов для принятия

управленческих решений;

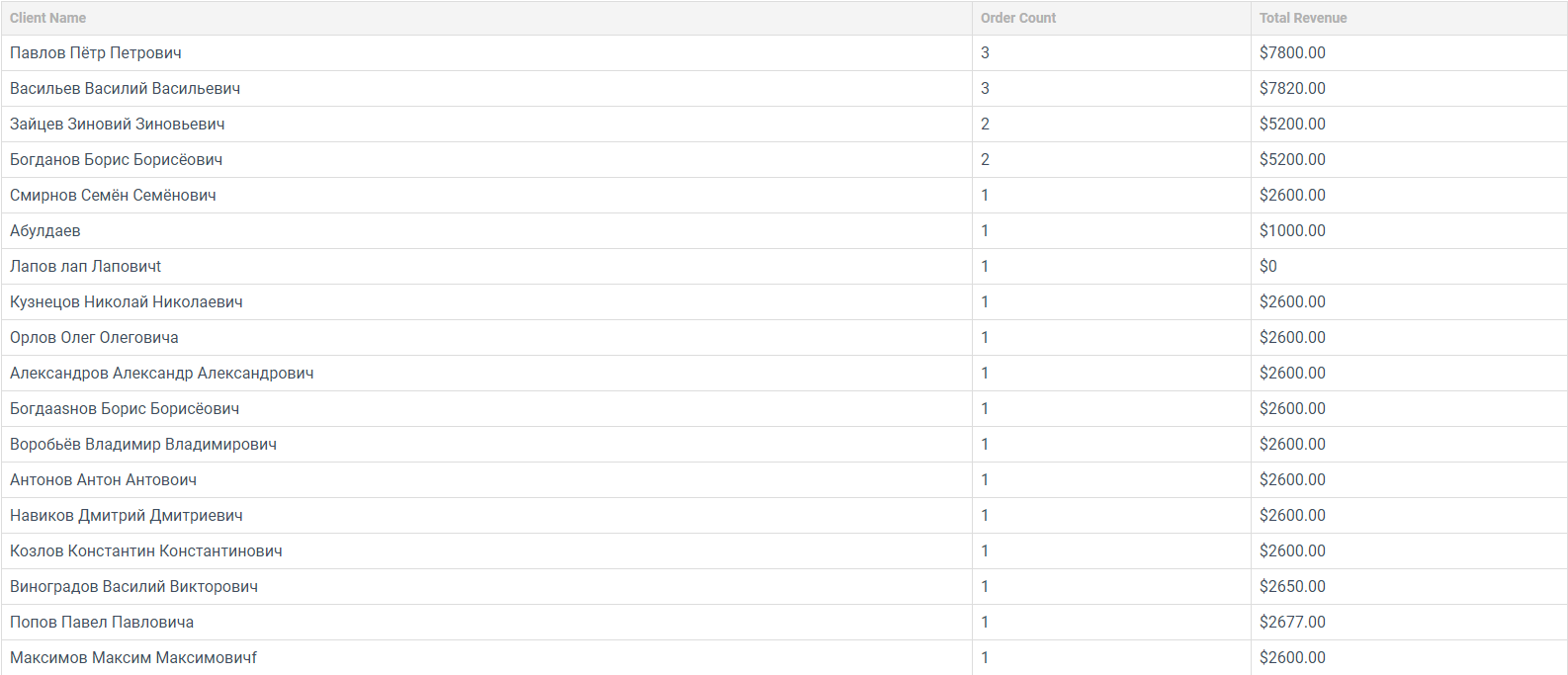
- Контроль финансовых показателей и активности.

**1.4. Требования к отчетам**

Отчет, реализуемый в рамках информационной системы, представляет собой аналитический документ, предназначенный для использования владельцем сети. Отчет предоставляет информацию о ключевых показателях деятельности магазина за определенный временной период. Включает в себя имя менеджера, количество заказов и выручку. Отчет позволяет оценить эффективность работы отдельных менеджеров.

Пример отчета представлен в таблице 1.

Таблица 1 – Пример отчета



**2. Модель данных**

2.1. Диаграмма концептуальной модели данных

Диаграмма концептуальной модели с отображением name в нотации Craw’s Foot, разработанная в среде Sybase PowerDesigner представлена на рисунке 1.

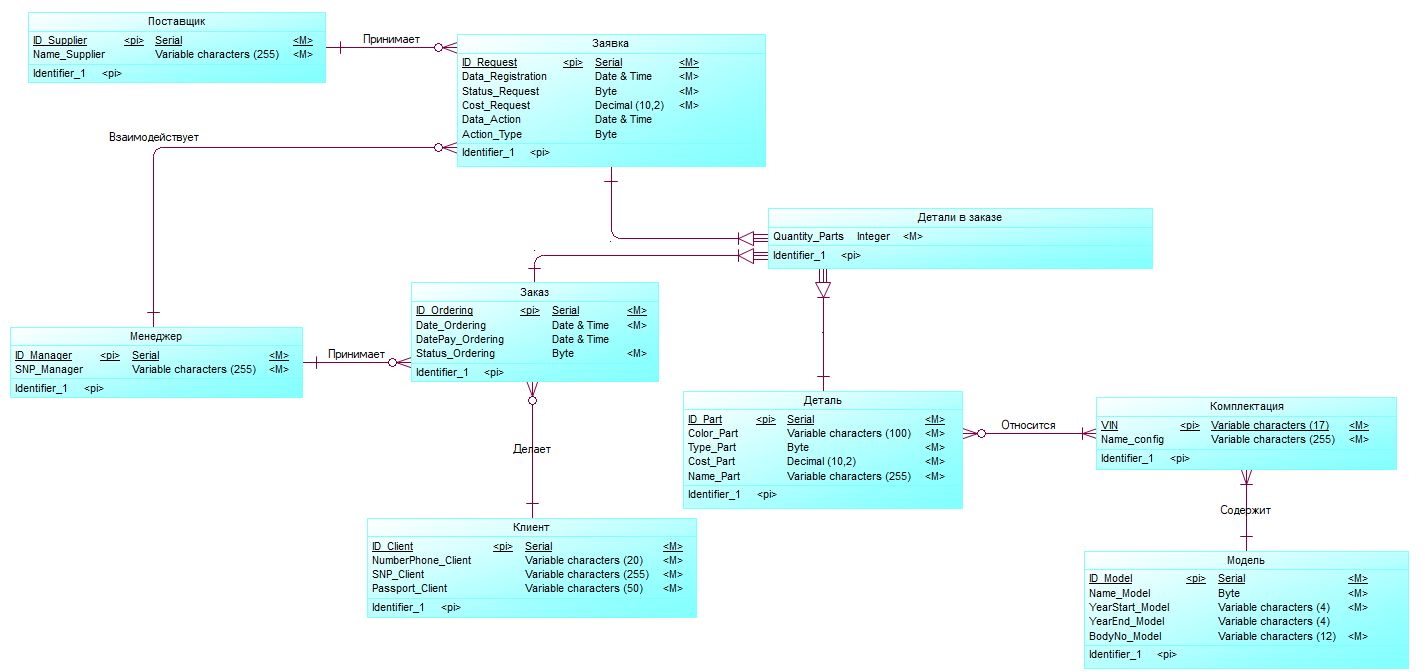


Рисунок 1 – Диаграмма концептуальной модели с отображением name в нотации Craw’s Foot

Диаграмма концептуальной модели с отображением code в нотации Craw’s Foot, разработанная в среде Sybase PowerDesigner представлена на рисунке 2.

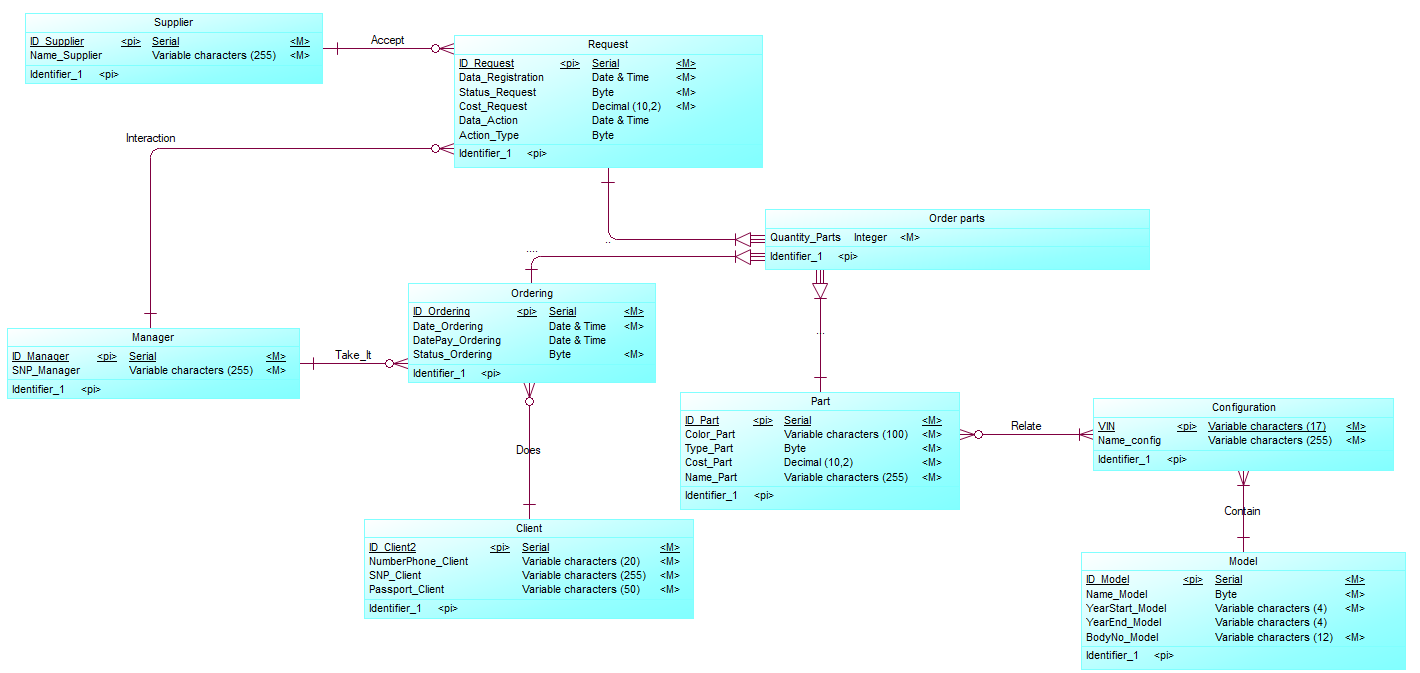


Рисунок 2 – Диаграмма концептуальной модели с отображением code в нотации Craw’s Foot

**2.2. Диаграмма физической модели данных**

Диаграмма физической модели с отображением name, разработанная в среде Sybase PowerDesigner представлена на рисунке 3.

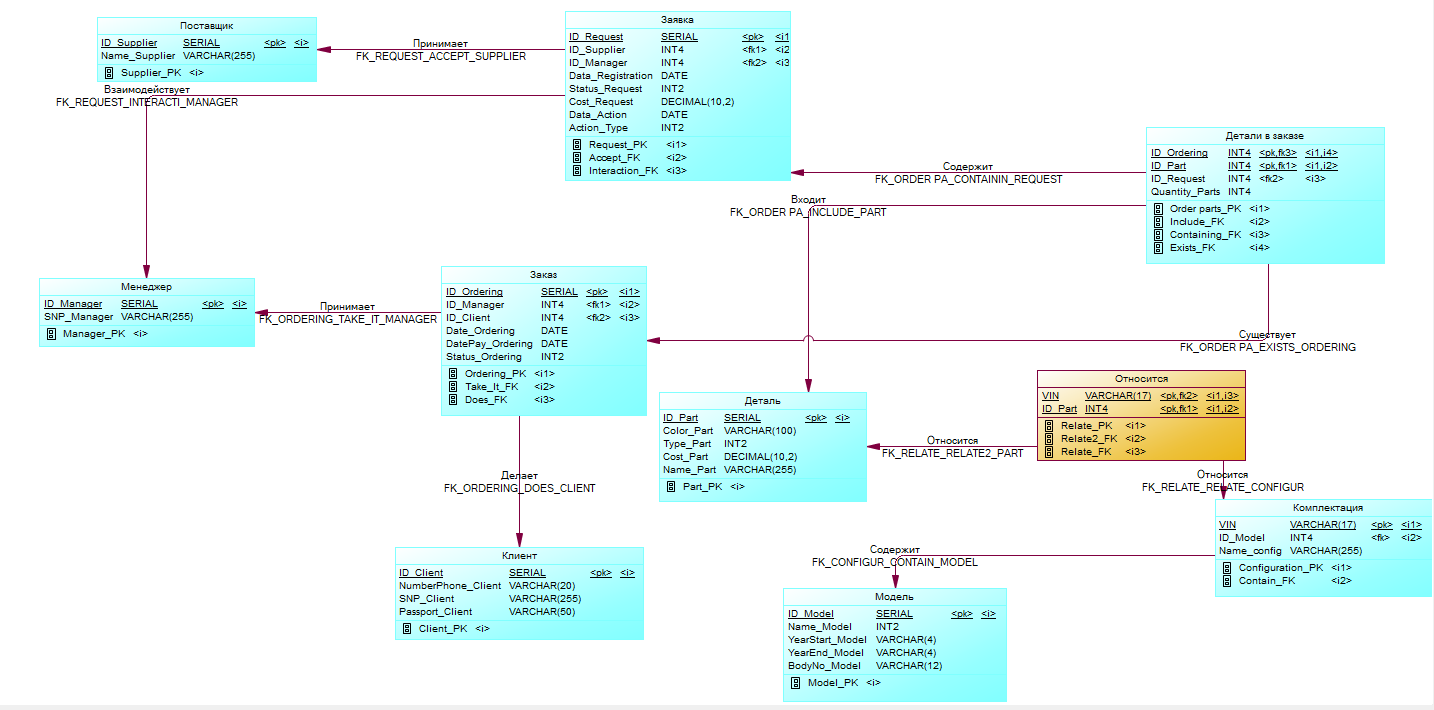


Рисунок 3 – Диаграмма физической модели с отображением name

Диаграмма физической модели с отображением code, разработанная в среде Sybase PowerDesigner представлена на рисунке 4.

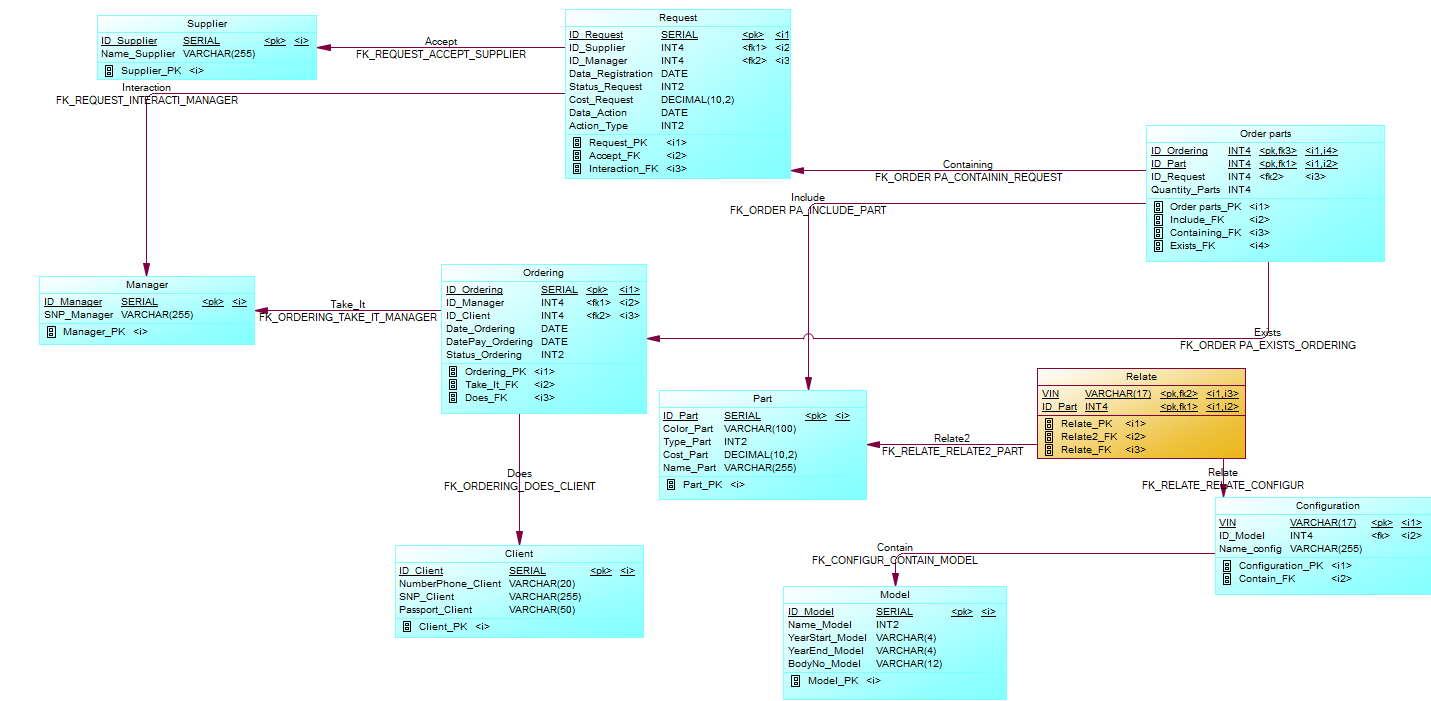


Рисунок 4 – Диаграмма физической модели с отображением code

**3. Разработка прикладного приложения**

**3.1. Платформа разработки**

Для разработки WEB приложения был выбран язык программирования Python и JavaScript. Для реализации графического пользовательского интерфейса использовался HTML и CSS. Для взаимодействия с базой данных PostgreSQL применялась библиотека psycopg2.

**3.2. Схема взаимодействия с БД**

**3.2.1. Связь приложения с БД**

Ниже приведен фрагмент кода, обеспечивающий подключения к базе данных.

def get\_db\_data():

    conn = psycopg2.connect(

        dbname="FINALAuto",

        user="postgres",

        password="Max342213",

        host="localhost",

        port="5432"

    )

    cursor = conn.cursor()

    cursor.execute("SELECT numberphone\_client, snp\_client, passport\_client, id\_client FROM client;")

    data = cursor.fetchall()

    conn.close()

    return [{"Number": row[0], "SNP": row[1], "Passport": row[2], "id":row[3]} for row in data]

В приведенном фрагменте используется метод psycopg2.connect, который устанавливает соединение с базой данных, используя параметры: адрес сервера, имя базы данных, учетные данные пользователя и порт. При успешном подключении создается курсор для выполнения запросов.

**3.2.2. Получение данных из БД**

Ниже приведен фрагмент кода, обеспечивающий загрузку данных и БД.

def get\_db\_data():

    conn = psycopg2.connect(

        dbname="FINALAuto",

        user="postgres",

        password="Max342213",

        host="localhost",

        port="5432"

    )

    cursor = conn.cursor()

    cursor.execute("SELECT numberphone\_client, snp\_client, passport\_client, id\_client FROM client;")

    data = cursor.fetchall()

    conn.close()

    return [{"Number": row[0], "SNP": row[1], "Passport": row[2], "id":row[3]} for row in data]

После выполнения запроса данные извлекаются с помощью метода fetchall и добавляются в таблицу

**3.2.3. Модификация данных в БД**

Ниже приведен фрагмент кода, отвечающий за модификацию данных в БД.

@app.route('/update-client/<int:id\_client>', methods=['PUT'])

def update\_client(id\_client):

    data = request.json

    new\_name = data.get('SNP')

    new\_num = data.get('Number')

    new\_pas = data.get('Passport')

    if new\_name is None and new\_num is None and new\_pas is None:

        return jsonify({"success": False, "message": "No valid fields to update"}), 400

    try:

        with psycopg2.connect(

            dbname="FINALAuto",

            user="postgres",

            password="Max342213",

            host="localhost",

            port="5432"

        ) as conn:

            with conn.cursor() as cursor:

                fields\_to\_update = []

                values = []

                if new\_name is not None:

                    fields\_to\_update.append("snp\_client = %s")

                    values.append(new\_name)

                if new\_num is not None:

                    fields\_to\_update.append("numberphone\_client = %s")

                    values.append(new\_num)

                if new\_pas is not None:

                    fields\_to\_update.append("passport\_client = %s")

                    values.append(new\_pas)

                values.append(id\_client)

                query = f"UPDATE client SET {', '.join(fields\_to\_update)} WHERE id\_client = %s"

                cursor.execute(query, values)

                conn.commit()

        return jsonify({"success": True, "message": "Client updated successfully"})

    except psycopg2.Error as e:

        print(f"Error updating client: {e}")

        return jsonify({"success": False, "message": "Error updating client"}), 500

**3.2.4. Вызов хранимой процедуры**

Процедура вызывается посредством добавления или изменения данных в таблице.

Данная хранимая процедура меняет поле type\_active в зависимости от активности со строкой.

CREATE OR REPLACE PROCEDURE handle\_request\_activity()

RETURNS TRIGGER AS $$

BEGIN

IF TG\_OP = 'INSERT' THEN

NEW.Action\_Type := 3; -- "Добавлено"

NEW.Data\_Action := NOW(); -- Текущая дата

RETURN NEW;

ELSIF TG\_OP = 'UPDATE' THEN

NEW.Action\_Type := 2; -- "Изменено"

NEW.Data\_Action := NOW(); -- Текущая дата

RETURN NEW;

END IF;

RETURN NULL;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_handle\_request\_insert

BEFORE INSERT ON request

FOR EACH ROW

EXECUTE FUNCTION handle\_request\_activity();

CREATE TRIGGER trg\_handle\_request\_update

BEFORE UPDATE ON request

FOR EACH ROW

EXECUTE FUNCTION handle\_request\_activity();

**3.3. Разработка пользовательского интерфейса**

Пользовательский интерфейс приложения разработан с HTML и CSS и включает несколько ключевых элементов. Главное окно содержит боковую панель, центральную область и верхнюю панель управления.

Боковая панель представляют собой меню, в котором перечислены все доступные таблицы базы данных. Пользователь выбирает нужную таблицу, после чего данные из нее отображаются в центральной области окна.

Центральная область предназначена для отображения таблицы с данными. В таблице можно просматривать записи, редактировать данные, а также удалять строки. Для каждой строки добавлены кнопки "Редактировать" и "Удалить", которые вызывают соответствующий диалог или выполняют удаление. Данные в таблице загружаются динамически в зависимости от выбранной в боковой панели таблицы.

Верхняя панель управления расположена над таблицей и включает несколько элементов. Поиск, сортировка, название таблицы и кол-во записей.

Для редактирования записей реализованы отдельные модальные окна. Они открываются при нажатии на кнопку редактирования строки. Диалоги содержат соответствующие поля ввода для данных, включая выпадающие списки и текстовые поля, адаптированные под структуру таблицы. После заполнения формы пользователь может сохранить изменения, которые затем обновляют данные в базе.

На рисунке 5 представлено приложения с открытой таблицей "configuration".

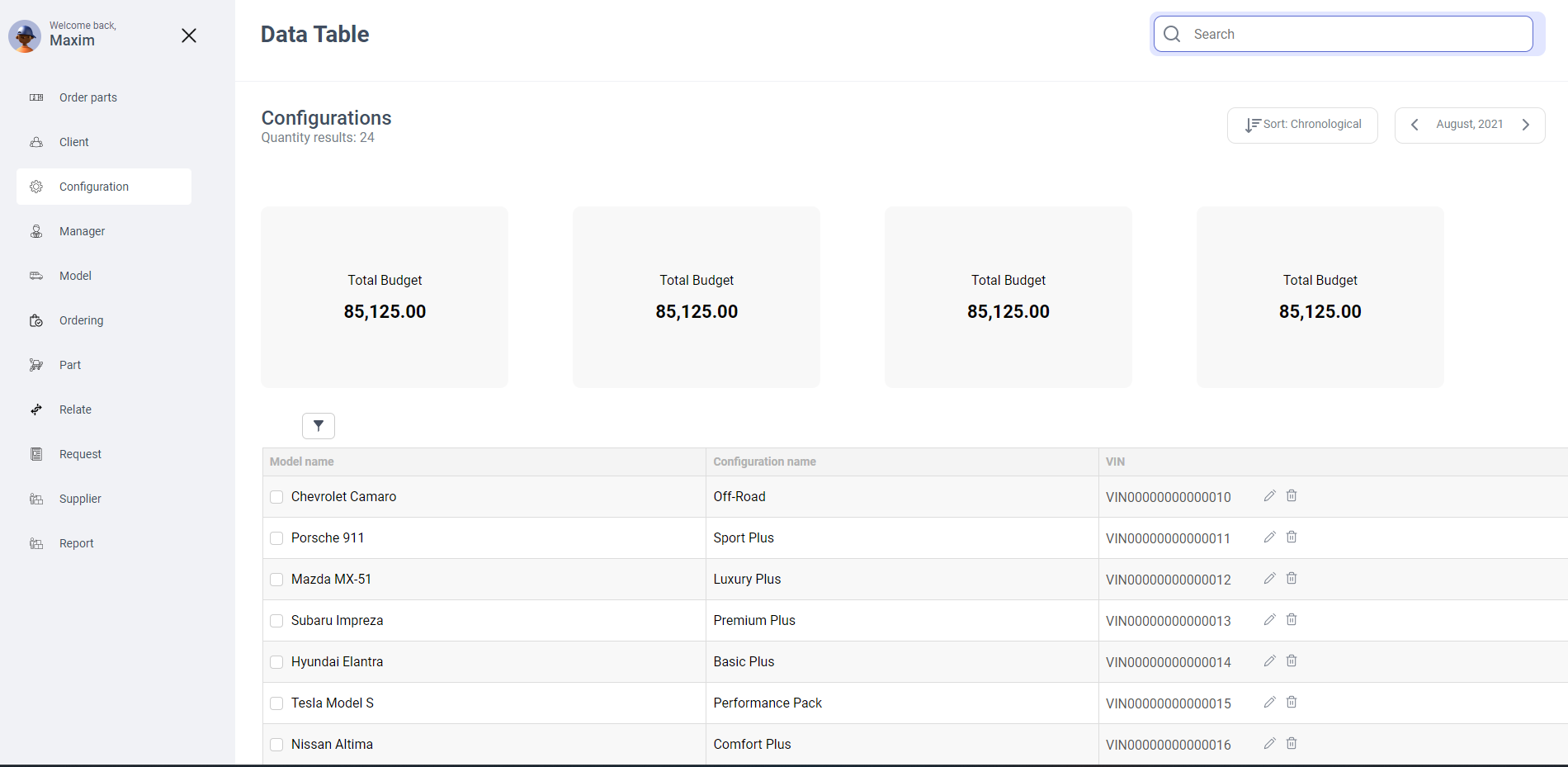


Рисунок 5 – Макет главного окна

На рисунке 6 представлено окно изменения записи для таблицы "configuration". На рисунке 7 отображены меню для сортировки и вывода отображаемых столбцов. На рисунке 8 представлено окно для формирования отчета.

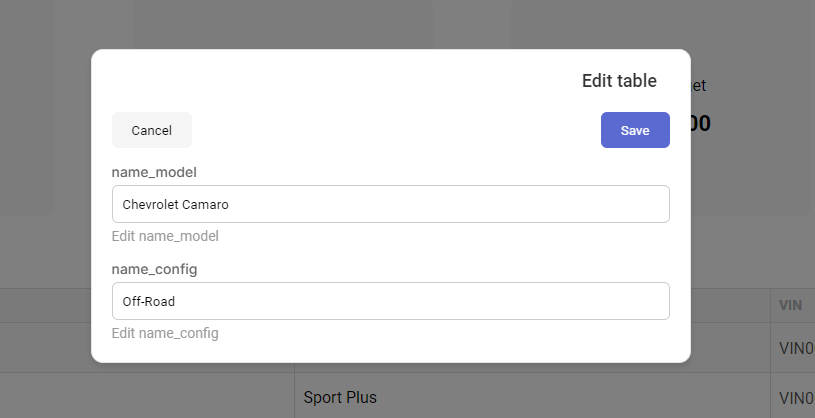


Рисунок 6 – Окно изменения записи

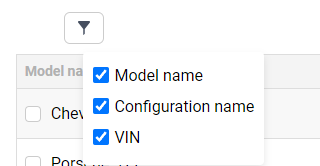
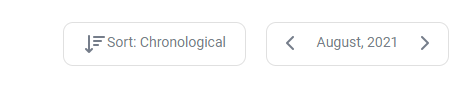


Рисунок 7 – Сортировки

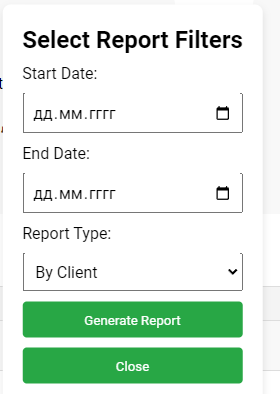


Рисунок 8 – Окно отчета

**3.4. Текст программы**

Основные функции для загрузки и обновления данных были представлены в пункте 3.3. Рассмотрим упрощенный фрагмент кода из функции showModal в котором инициализируются поля формы для редактирования записи в таблице.

    if (input.dataset.type === 'phone' || key.toLowerCase().includes('number')) {

                if (value.replace(/\D/g, '').length < 11) {

                    alert(`Поле "${key}" должно содержать 11 цифр.`);

                    valid = false;

                }

                value = value.replace(/\D/g, ''); // Удаляем все нецифровые символы

                if (value.startsWith('7')) {

                    value = '8' + value.slice(1); // Меняем +7 на 8

                }

            } else if (input.dataset.type === 'passport') {

                if (value.replace(/\D/g, '').length < 10) {

                    alert(`Поле "${key}" должно содержать 10 цифр.`);

                    valid = false;

                }

                value = value.replace(/^(\d{4})(\d{0,6})$/, '$1 $2').trim();

            } else if (input.dataset.type === 'bodyNo') {

                if (!/^[A-Z]{3}\d{6}$/.test(value)) {

                    alert(`Поле "${key}" должно содержать 3 заглавные буквы и 6 цифр.`);

                    valid = false;

                }

Рассмотрим функцию поиска

function searchRows() {

                const searchTerm = document.getElementById('searchInput').value.toLowerCase();

                let found = false;

                allRows.forEach(({ row, rowData }) => {

                    const searchInColumns = `${rowData.model\_name} ${rowData.name\_con} ${rowData.VIN}`;

                    if (searchInColumns.toLowerCase().includes(searchTerm)) {

                        row.style.display = '';

                        found = true;

                    } else {

                        row.style.display = 'none';

                    }

                });

                updateRecordCount();

            }

Функция удаления:  
document.querySelectorAll('.butt\_delete').forEach(button => {

                button.addEventListener('click', function () {

                    const supplierId = this.getAttribute('data-id');

                    fetch(`/delete2/${supplierId}`, {

                        method: 'DELETE',

                    })

                    .then(response => response.json())

                    .then(data => {

                        if (data.success) {

                            alert('Configuration deleted successfully!');

                            document.getElementById('CONFIG').click();

                        } else {

                            alert(`Error deleting Configuration: ${data.message}`);

                        }

                    })

                    .catch(error => console.error('Error:', error));

                });

            });

Функция изменения данных в реальном времени (в таблице)

document.querySelectorAll('.editable-input').forEach(input => {

                input.addEventListener('change', function () {

                    const field = this.getAttribute('data-key');

                    const id = this.getAttribute('data-id');

                    const idS = this.getAttribute('data-idC')

                    const newValue = this.value.trim();

                    if (!newValue) {

                        alert('Поле не может быть пустым.');

                        return;

                    }

                    const updateData = {};

                    updateData[field] = newValue;

                    console.log('Sending update:', updateData);

                    fetch(`/update-config/${idS}/${id}`, {

                        method: 'PUT',

                        headers: { 'Content-Type': 'application/json' },

                        body: JSON.stringify(updateData),

                    })

                    .then(response => response.json())

                    .then(data => {

                        if (data.success) {

                            console.log(`Field ${field} updated successfully for ID ${id}`);

                        } else {

                            console.error(`Error: ${data.message}`);

                        }

                    })

                    .catch(error => console.error('Error:', error));

                });

            });

Изменение через модальное окно:  
document.querySelectorAll('.butt\_up').forEach(button => {

                button.addEventListener('click', function () {

                    const modelId = this.getAttribute('data-id');

                    const vin = this.getAttribute('data-idC');

                    const row = this.closest('tr');

                    // Извлекаем данные из ячеек строки

                    const currentData = {

                        name\_model: row.querySelector('td:nth-child(1) input[type="text"]').value.trim(),

                        name\_config: row.querySelector('td:nth-child(2) input').value.trim(),

                    };

                    showModal(currentData, (updatedData) => {

                        fetch(`/update-config/${vin}/${modelId}`, {

                            method: 'PUT',

                            headers: { 'Content-Type': 'application/json' },

                            body: JSON.stringify(updatedData),

                        })

                            .then((response) => response.json())

                            .then((data) => {

                                if (data.success) {

                                    alert('Configuration updated successfully!');

                                    document.getElementById('CONFIG').click(); // Перезагружаем таблицу

                                } else {

                                    alert(`Error updating Configuration: ${data.message}`);

                                }

                            })

                            .catch((error) => console.error('Error:', error));

                    });

                });

            });

**4. Тестирование прикладного приложения**

На рисунке 9 пределен пример работы поиска (найдены люди с фамилией Богданов).

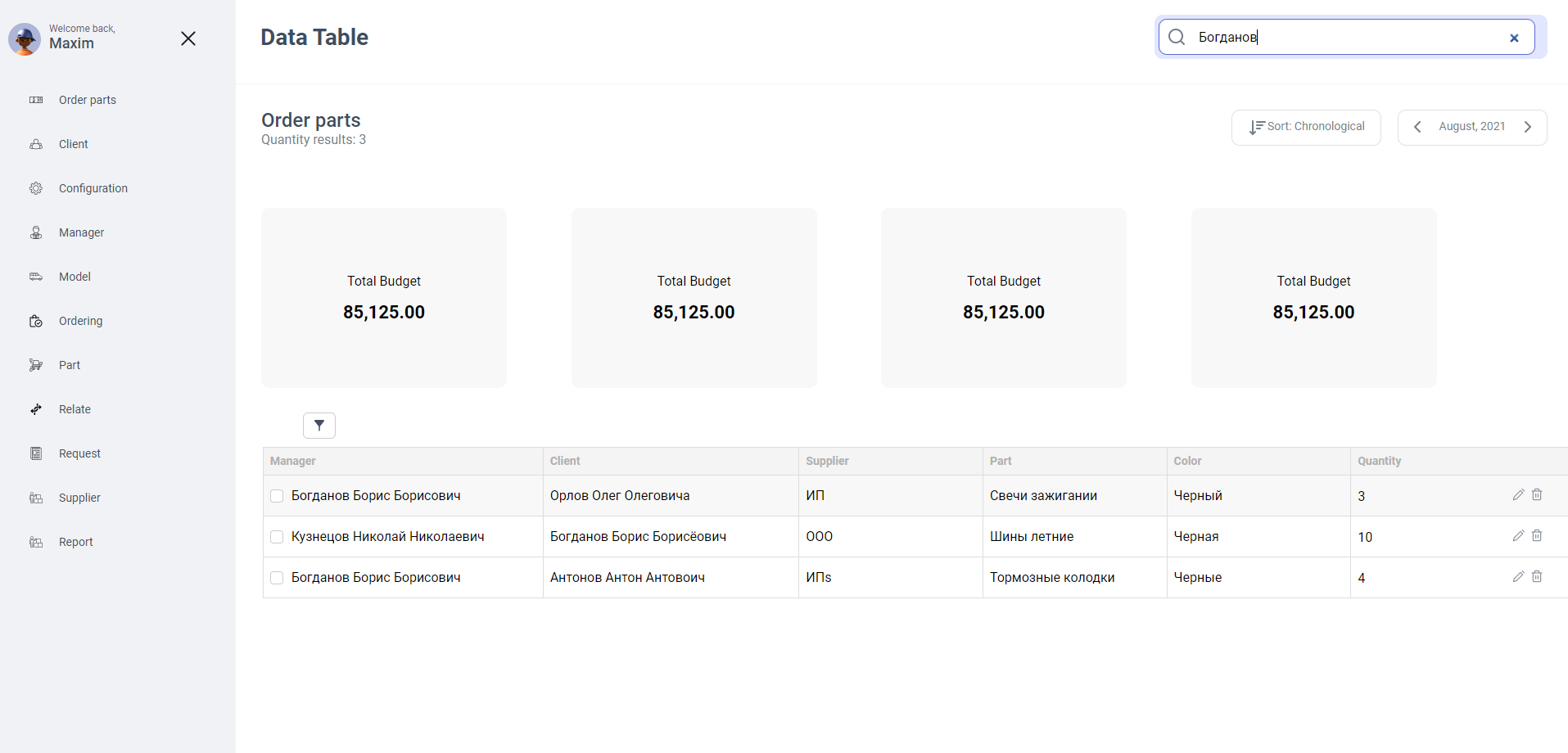


Рисунок 9 – Тестирование поиска

На рисунке 10 продемонстрирован пример работы сортировки (По времени).

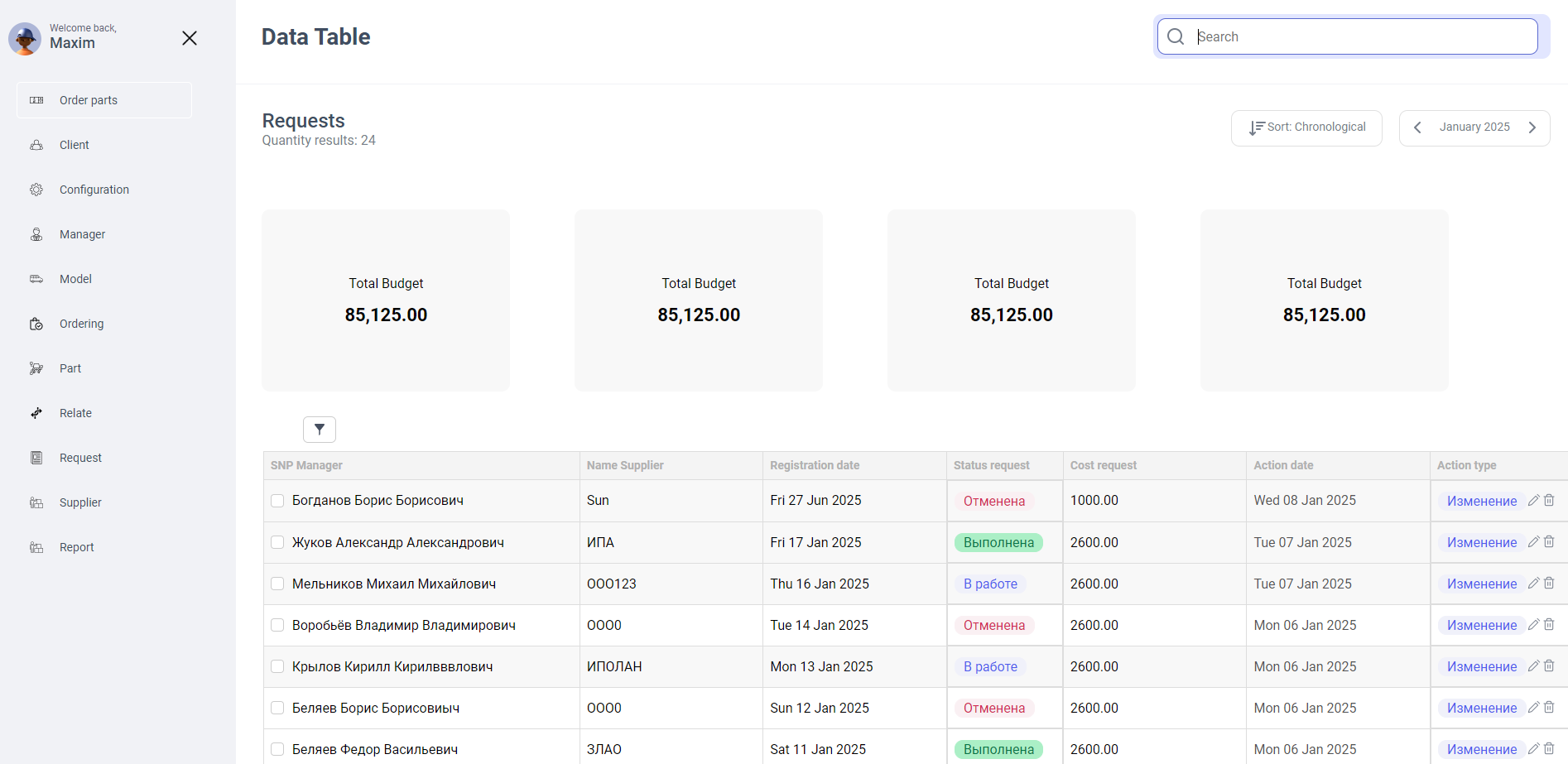


Рисунок 10 – Тестирование сортировки

На рисунках 11-12 показан пример формирования отчета. На рисунке 10 выбрана таблица, которая будет включена в отчет, также указан временной промежуток. На рисунке 12 представлен пример сформатированного отчета.

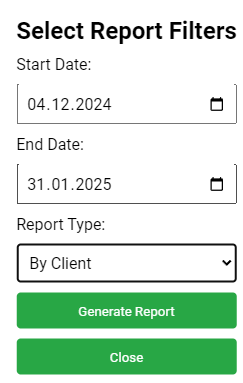


Рисунок 11 – Критерии составления отчета

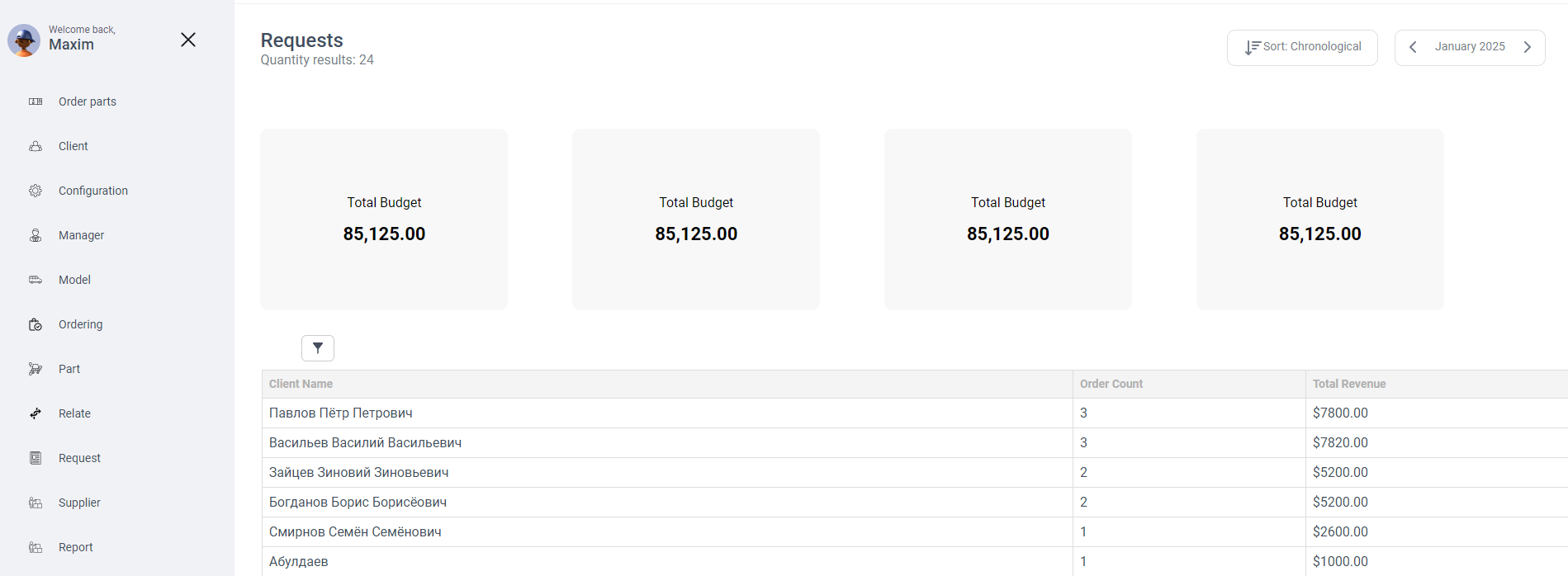


Рисунок 12 – Отчет