

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ
по дисциплине «Введение в нереляционные базы
данных»

Тема: Информационная система для пожарных бригад

Студенты гр. 2303 и 2382

Ефремова А.В.

Ильин Е.С.

Карнаухов В.В.

Семененко А.С.

Щёголева Н.Д.

Преподаватель

Заславский М.М.

Санкт-Петербург

2025

ЗАДАНИЕ

Студенты: Ефремова А.В., Ильин Е.С., Карнаухов В.В.

Группа 2303

Студенты: Семененко А.С., Щёголева Н.Д.

Группа 2382

Тема проекта: Информационная система для пожарных бригад

Исходные данные:

Необходимо реализовать сервис, где будет храниться вся необходимая информация для работы одного пожарного депо, а также необходимые веб-страницы, чтобы с этими данными работать. Для реализации задачи предлагается использовать СУБД Neo4j.

Содержание пояснительной записки:

«Содержание»

«Введение»

«Качественные требования к решению»

«Сценарий использования»

«Модель данных»

«Разработка приложения»

«Вывод»

«Приложение»

Предполагаемый объем пояснительной записки: не менее 25 страниц.

Дата выдачи задания:

Дата сдачи реферата:

Дата защиты реферата:

Студенты гр. 2303 и 2382

Ефремова А.В.

Ильин Е.С.

Карнаухов В.В.

Семененко А.С.

Щёголева Н.Д.

Преподаватель

Заславский М.М.

АННОТАЦИЯ

В рамках курса «Введение в нереляционные базы данных» необходимо разработать в составе команды приложение «Информационная система для пожарных бригад». При реализации приложения предлагается использовать СУБД Neo4j. При выполнении работы будут учитываться особенности СУБД Neo4j, которые будут использоваться для наиболее эффективной реализации проекта. Исходный код проекта и дополнительная информация о нем находится по ссылке: <https://github.com/moevm/nosql1h25-unirepair>

ANNOTATION

As part of the course "Introduction to non-relational databases", it is necessary to develop the application "Information System for Fire brigades" as part of the team. When implementing the application, it is proposed to use the Neo4j database management system. When performing the work, the features of the Neo4j DBMS will be taken into account, which will be used for the most effective implementation of the project. The source code of the project and additional information about it can be found at the link: <https://github.com/moevm/nosql1h25-unirepair>

Оглавление

1. Введение	7
2. Качественные требования к решению	7
3. Сценарии использования.....	7
4. Модель данных.....	12
5. Разработанное приложение.....	31
6. Вывод.....	33
7. Приложения.....	34
8. Используемая литература	34

1. ВВЕДЕНИЕ

Целью работы является создание приложения, которое обеспечит возможность работы пожарного депо. Будет разработано приложение, упрощающее работу сотрудников пожарного депо (операторов, администраторов, пожарных и бригадиров). Приложение позволит пожарным получать актуальную информацию о вызовах (даже на мобильных устройствах), бригадирам создавать отчёты о вызовах, операторам автоматизировать отправку информации о вызовах, а администраторам регулировать работу приложения и получать статистику.

2. КАЧЕСТВЕННЫЕ ТРЕБОВАНИЯ К РЕШЕНИЮ

Требуется разработать приложение с использованием СУБД Neo4j и данных OpenStreetMap. Приложение должно позволять пользоваться им разным сотрудникам, распределять пожарных по бригадам, создавать вызовы, распределять автомобили, заполнять отчёты с оценкой ущерба.

3. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ

Макет UI

Все экраны приложения и переходы между ними представлены на рисунке 1, также их можно найти по ссылке <https://github.com/moevm/nosql1h25-unirepair> в разделе Wiki «Макет и сценарии использования».

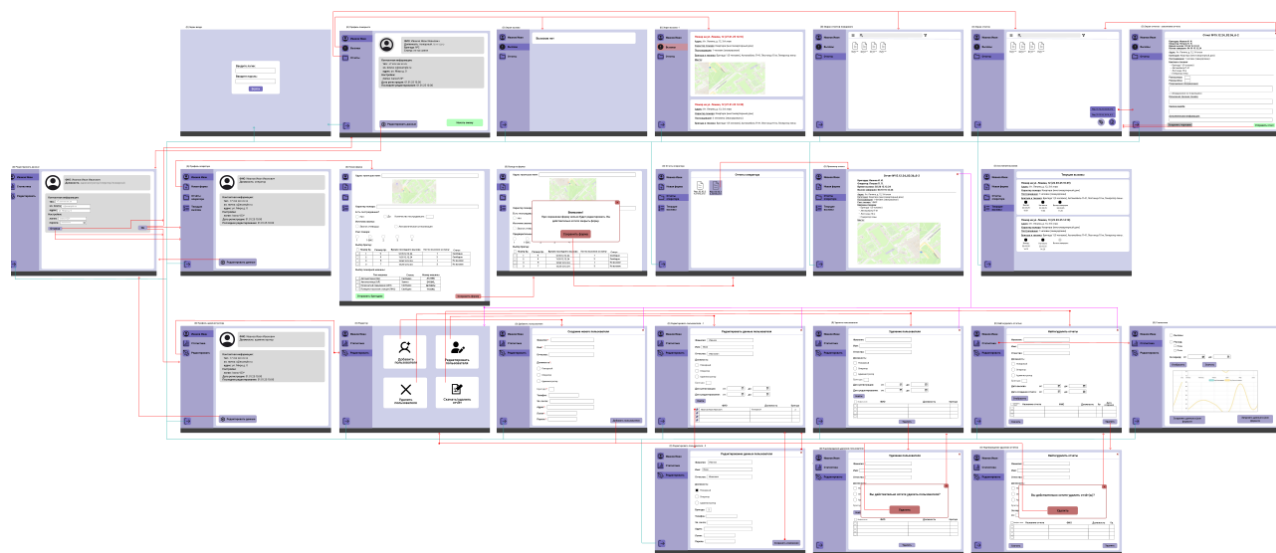


Рисунок 1 – Макет UI приложения для работы пожарного депо

Описание сценариев использования

1) Действующее лицо: Оператор

Профиль оператора

При входе в приложение оператор попадает в свой профиль. В профиле отображены ФИО и должность сотрудника, дата создания и последнего редактирования профиля. Оператор может редактировать свою контактную информацию, а также пароль с помощью кнопки "Редактировать данные" (для однозначности идентификации пользователей редакция логина недоступна): сохранить изменения можно с помощью кнопки "Ок", отменить изменения можно с помощью кнопки "Отмена".

Работа с вызовами

Создание вызова: при получении вызова оператор нажимает кнопку "Новая форма". При нажатии на кнопку автоматически фиксируется время вызова. В "Новой форме" оператор обязательно заполняет все поля.

Отправка вызова: если все поля заполнены, становится доступна кнопка "Отправить бригадам". Нажав эту кнопку, оператор отправляет сообщение о пожаре выбранным бригадам (пожарные увидят это сообщение на "Экране вызова"). Также эта форма появляется у оператора на экране "Состояние вызовов".

Сохранить форму: после отправки вызова становится доступна кнопка "Сохранить форму". После нажатия "Сохранить форму" форма сохраняется в формате отчета (в названии отчета - дата и время поступления вызова).

Текущие вызовы

Нажатие на кнопку "Текущие вызовы" откроет окно "Состояние вызовов", где будут показаны все текущие вызовы оператора и их статус. Фиксировать статусы вызовов - задача оператора. После нажатия на вызов завершен фиксируется время завершения вызова, соответствующее окно с информацией о вызове у пожарного исчезнет. Нажатие на блок с конкретным вызовом (кроме полей с фиксированием статуса вызова) откроет соответствующую форму вызова (которую можно редактировать).

Работа с отчетами

Оператор может просмотреть свои отчёты, нажав кнопку "Отчеты оператора". Нажав на отчет, оператор может его просмотреть (без возможности редактирования).

2) Действующее лицо: Администратор

Профиль администратора

При входе в приложение администратор попадает в свой профиль. В профиле отображены ФИО и должность сотрудника, дата создания и последнего редактирования профиля. Администратор может редактировать свою контактную информацию через кнопку "Редактировать данные". Сохранить изменения – кнопка "Ок". Отменить изменения – кнопка "Отмена".


Управление пользователями

Кнопка "Редактировать" открывает окно "Редактор", где администратор может выбрать одно из действий:

Добавить пользователя

В окне "Добавить пользователя" необходимо ввести: ФИО, должность, номер бригады (если сотрудник – пожарный), телефон, эл. почту, адрес проживания, временный логин и пароль (пользователь может изменить их в своем профиле). Кнопка "Добавить пользователя" сохраняет данные в базе. Кнопка "Отмена" закрывает форму без сохранения.

Редактировать пользователя

В окне "Редактировать пользователя - 1" можно найти сотрудника по ФИО и/или должности и/или дате создания/редактирования профиля пользователя. После поиска отображается список сотрудников. Клик на значок  (карандаш) открывает окно "Редактировать пользователя - 2", где можно изменить данные. Кнопка "Сохранить изменения" обновляет данные в базе. Кнопка "Отмена" закрывает форму без сохранения.

Удалить пользователя

В окне "Удалить пользователя" можно найти сотрудников по ФИО и/или должности и/или дате создания/редактирования профиля пользователя. После

поиска отображается список сотрудников. Выбрав чекбокс напротив сотрудника и нажав "Удалить", пользователь удаляется из базы.

Скачать/Удалить отчет

В окне "Найти/удалить отчеты" можно найти отчеты по: ФИО сотрудника, должности, дате вызова, дате создания отчета. Отмеченные отчеты можно скачать или удалить.

Просмотр статистики

Кнопка "Статистика" открывает окно со статистикой вызовов и расхода ресурсов. Доступны функции массового импорта-экспорта данных: "Сохранить данные в JSON формате", "Загрузить данные в JSON формате".

3) Действующее лицо: Пожарный

Профиль пожарного

При входе в приложение пожарный попадает в свой профиль. В профиле отображены ФИО и должность сотрудника, дата создания и последнего редактирования профиля. Редактирование контактных данных – кнопка "Редактировать данные". Сохранение изменений – кнопка "Ок". Отмена изменений – кнопка "Отмена". Кнопка "Начать смену" / "Завершить смену" (при нажатии меняется на противоположную. Также при нажатии этой кнопки на странице пожарного меняется его статус: "на смене" / "не на смене"

Работа с вызовами

При нажатии кнопки "Вызовы" открывается "Экран вызова". Если вызова нет, отображается надпись "Вызовов нет". Если поступает вызов, приложение автоматически отображает всю информацию о пожаре. Если вызовов несколько будет показана краткая информация о всех вызовах. Нажав на соответствующий вызов, раскроется подробная информация о вызове (а именно карта, соответствующая адресу). Повторное нажатие обратно сожмет данные.

Работа с отчетами

При нажатии кнопки "Отчеты" открывается "Экран отчетов", где отображены все завершенные отчёты бригад. Доступен поиск и сортировка отчетов. Поиск отчетов осуществляется по названию отчета. При нажатии на

фильтр доступна сортировка по названию и по дате создания отчета. Так как в названии отчетов содержится дата и время поступления соответствующих вызовов, их сортировка по алфавиту представляет собой сортировку по дате вызова.

Заполнение отчета о пожаре (доступно только бригадиру)

Бригадир нажимает "📄+" и выбирает предзаполненный отчет. После выбора открывается "Экран отчетов - заполнение отчета", где доступны: "Сохранить черновик" – позволяет сохранить частично заполненный отчет (он переместится в черновые отчеты - значок ✎ (карандаш)). "Отправить отчет" – сохраняет отчет как завершённый (без возможности редактирования). Обычные пожарные не заполняют отчёты – это делает только бригадир.

4. МОДЕЛЬ ДАННЫХ

Нереляционная модель данных (Neo4j)

Сущности модели:

Сотрудник (User): Фамилия (Family Name), Имя (First Name), Отчество (Father Name), Должность (Role) (одно из четырёх: оператор / администратор / бригадир / пожарный) (метка), Номер бригады (Brigade Number) (обязательно для бригадира/пожарного, для оператора/администратора бригада 0), Адрес проживания (Address), Номер телефона (Phone), Адрес электронной почты (EMail), Логин (Login) (должен быть уникальным), Хэш пароля (Password Hash), Время регистрации (Registered At), Время последнего изменения (Modified At), Статус (status) (активен / удалён) (метка).

Форма вызова (Call Form): Статус (Status) (незавершён / завершён) (метка), Время создания (Created At), Последнее обновление (Modified At) (время завершения для завершённого вызова, так как редактировать их уже нельзя), Источник вызова (Call Source), Адрес (Address) 2-5. Axis Aligned Bounding Box (прямоугольник на карте, охватывающий пожар, либо две точки, либо одна точка и высоты/ширина прямоугольника), Характер пожара (Type), Ранг пожара (Fire Rank) (Одно из: 1, 1-БИС, 2, 3, 4), Пострадавшие (Victims) (Количество),

Прикреплённая бригада (Assigned To), Автомобиль (Auto).

Инвентарь (Inventory): Наименование предмета (Name).

Отчёт (Report): Расход воды (Water Spent), Расход пены (Foam Spent), Предполагаемая причина пожара (Alleged Fire Cause), Оценка ущерба (Damage), Дополнительная информация (Additional Notes), Статус (Status) (New/Incomplete/Complete) (метка), Время последнего изменения (Modified At).

Возможные связи между описанными сущностями:

- 1) (:CallForm)-[:INVENTORY_USED]->(:Inventory)
- 2) (:Report)-[:ON_CALL]->(:CallForm)
- 3) (:Report)-[:DAMAGED]->(:Inventory)
- 4) (:User)-[:CREATED]->(:CallForm) (Пользователь должен быть оператором)
- 5) (:User)-[:CREATED_BY]->(:CallForm)
- 6) (:User)-[:FILLED_IN]->(:Report) (Пользователь должен быть бригадиром)

Графическое представление модели данных представлено на рисунке 2.

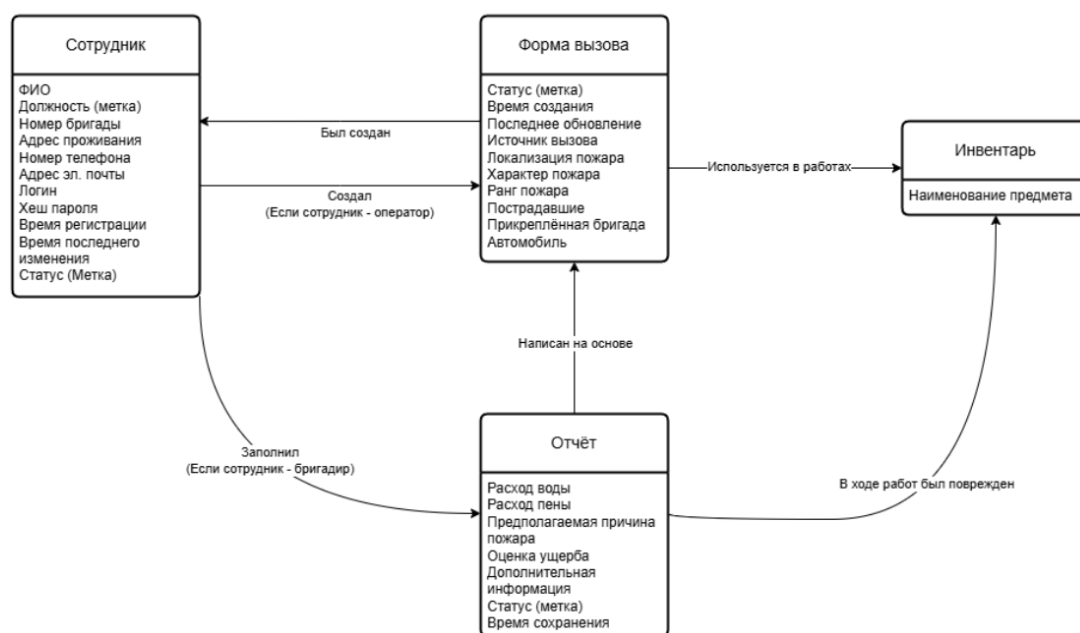


Рисунок 2 – Графическое представление нереляционной модели данных

Оценка объема информации (нереляционная модель)

Пусть S_u , S_c , S_i , S_r – средние размеры объектов типа Сотрудник, Форма вызова, Инвентарь и Отчёт. Также введём переменные для связей и количества сущностей: N_x – количество X , L_{xy} – связь $(:X) \rightarrow (:Y)$, u – Сотрудники, c – Форма вызова, i – Инвентарь, r – Отчёт. Опишем перечисленные переменные означают:

S_u = ФИО (3-х в среднем 10 букв или примерно 20 байт в кодировке UTF-8) + Должность (пусть будет строкой на 10 байт) + Номер бригады (целое число, 4 байта) + Адрес (50 символов) + Телефон (11 символов) + Email (30 символов) + Логин (24 символа) + Хеш пароля (64 символа) + Время регистрации (Временная метка, 8 байт) + Время последнего изменения (8 байт) + Статус (10 символов) = 279 байт.

S_c = Статус (10 символов) + Время создания (8 байт) + Последнее обновление (8 байт) + Источник вызова (20 символов) + Адрес (50 символов) + Локализация (2 точки на карте по 16 байт каждая, 32 байта) + Характер (20 символов) + Ранг (10 символов) + Пострадавшие (число, 4 байта) = 162 байта.

S_i = Наименование (30 символов) = 30 байт.

S_r = Расход воды (число, 4 байт) + Расход пены (4 байта) + Причина (50 символов) + Ущерб (строка, 100 байт) + Дополнительная информация (100 символов) + Статус (10 символов) + Время последнего изменения (8 байт) = 276 байт.

Средние размеры всех связей между узлами возьмём в 8 байт (поле с ID). Тогда общая формула принимает вид: $V(N_u, N_c, N_i, N_r, L_{ci}, L_{rc}, L_{ri}, L_{uc}, L_{cu}, L_{ur}, L_{ru}) = 279N_u + 162N_c + 30N_i + 276N_r + 8(L_{ci} + L_{rc} + L_{ri} + L_{uc} + L_{cu} + L_{ur} + L_{ru})$.

Рассмотрим зависимость от количества вызовов N_c . Количество отчётов предположительно прямо пропорционально количеству вызовов, допустим $N_r = N_c$ (так как для каждого вызова будет отчёт). Положим $N_u = 0.01 * N_c$ (сотрудники предположительно меняются довольно редко, предположим, что на 100 вызовов в пожарной части будет в среднем один сотрудник), $N_i = 0.1 * N_c$ (предположим, что в среднем инвентарь выходит из строя раз в 10 вызовов).

Оценим количество связей: $L_{rc} = N_c$ (по одной связи на каждый отчёт). $L_{ri} = 0.1 * N_c$ (раз в 10 вызовов инвентарь выходит из строя). $L_{uc} = L_{cu} = N_c$ – по одному исполнителю на вызов. $L_{ur} = L_{ru} = N_c$ – по одному бригадиру на отчёт. $L_{ci} = N_c$.

С учётом всего этого, формула принимает вид (обозначим N_c как просто N для простоты): $V(N) = 2.79N + 162N + 3N + 276N + 8(N + N + 0.1N + N + N + N + N) = 492.59N$ (байт).

В описанной модели присутствует избыточность – дублирование связи между оператором и его формой. Чистый объём данных (без связей) $492.59N - 8N = 484.59N$ (байт). Тогда избыточность модели: Фактический объём / Чистый объём = $492.59N / 484.59N = 1,0165088$.

Направление роста модели при увеличении количества объектов каждой сущности:

Сотрудники: Влияние отдельного сотрудника на потребляемую память значительно. Но предположительно их количество растёт очень медленно, потому их влияние на рост модели невелико.

Форма вызова: Значительное влияние отдельной формы, а за счёт большего количества связей (прямых и косвенных, так как отчёты появляются из форм вызова) имеем и наибольшее суммарное влияние на рост модели.

Инвентарь: Один предмет инвентаря весит немного и сам по себе не создаёт новых связей, а их количество растёт умеренно (раз в 10 вызовов), в итоге влияние на рост модели наименьшее.

Отчёт: Связан с формами вызовов 1 к 1, потому имеет значительное влияние на рост модели.

Примеры запросов для базы данных (нереляционная)

Примеры запросов для заполнения базы данных и создание связей представлены в листинге 1.

Листинг 1 – Заполнение базы данных и создание связей

```
// Создание пользователей
CREATE (:User:Operator:Active {
    familyName: "Иванов",
    firstName: "Иван",
    fatherName: "Иванович",
    brigadeNumber: 1,
    address: "г. Москва, ул. Ленина, д. 1",
    phone: "+7 (123) 456-78-90",
    email: "ivanov@example.com",
    login: "ivanov",
    passwordHash: "hashed_password_1",
    registeredAt: timestamp(),
    modifiedAt: timestamp()
});

CREATE (:User:Fireman:Brigadier:Active {
    familyName: "Петрова",
    firstName: "Мария",
    fatherName: "Сергеевна",
    brigadeNumber: "1",
    address: "г. Москва, ул. Пушкина, д. 2",
    phone: "+7 (123) 987-65-43",
    email: "petrova@example.com",
    login: "petrova",
    passwordHash: "hashed_password_2",
    registeredAt: timestamp(),
    modifiedAt: timestamp()
});

CREATE (:User:Admin:Active {
    familyName: "Сидоров",
    firstName: "Алексей",
    fatherName: null,
    brigadeNumber: null,
    address: "г. Москва, ул. Чехова, д. 3",
    phone: "+7 (123) 321-45-67",
    email: "sidorov@example.com",
    login: "sidorov",
    passwordHash: "hashed_password_3",
    registeredAt: timestamp(),
    modifiedAt: timestamp()
});

// Создание формы вызова
CREATE (:CallForm:Incomplete {
    createdAt: timestamp(),
    modifiedAt: timestamp(),
    callSource: "телефонный звонок",
    fireAddress: "г. Москва, ул. Ленина, д. 1",
```



```

        bottomLeft: point({latitude: 55.7558, longitude: 37.6173}),
        topRight: point({latitude: 55.7560, longitude: 37.6180}),
        fireType: "лесной пожар",
        fireRank: "3",
        victimsCount: 0,
        assignedTo: null,
        auto: null
    });

    // Создание инвентаря
    CREATE (:Inventory { name: "Пожарная машина" });
    CREATE (:Inventory { name: "Лестница" });
    CREATE (:Inventory { name: "Лопата" });

    // Создание отчёта
    CREATE (:Report:Complete {
        waterSpent: 1000,
        foamSpent: 200,
        allegedFireCause: "неосторожное обращение с огнём",
        damage: 50000,
        additionalNotes: "Пожар был локализован быстро.",
        modifiedAt: timestamp()
    });

    // Связь между формой вызова и инвентарём
    MATCH (cf:CallForm), (inv1:Inventory {name: "Пожарная машина"})
    CREATE (cf)-[:INVENTORY_USED]->(inv1);

    MATCH (cf:Callform), (inv2:Inventory {name: "Лестница"})
    CREATE (cf)-[:INVENTORY_USED]->(inv2);

    // Связь между отчётом и формой вызова
    MATCH (cf:CallForm), (rep:Report)
    CREATE (rep)-[:ON_CALL]->(cf);

    // Связь между отчётом и инвентарём
    MATCH (rep:Report), (inv3:Inventory {name: "Лестница"})
    CREATE (rep)-[:DAMAGED]->(inv3);

    // Связь между оператором и формой вызова
    MATCH (u1:User {login: "ivanov"}), (cf:CallForm)
    CREATE (u1)-[:CREATED]->(cf);

    // Обратная связь между формой вызова и оператором
    MATCH (u2:User {login: "ivanov"}), (cf:Callform)
    CREATE (cf)<-[:CREATED_BY]-(u2);

    // Связь между бригадиром и отчётом
    MATCH (u2:Brigadier), (rep:Report)
    CREATE (u2)-[:FILLED_IN]->(rep);

```

Пример данных представлен на рисунке 3.

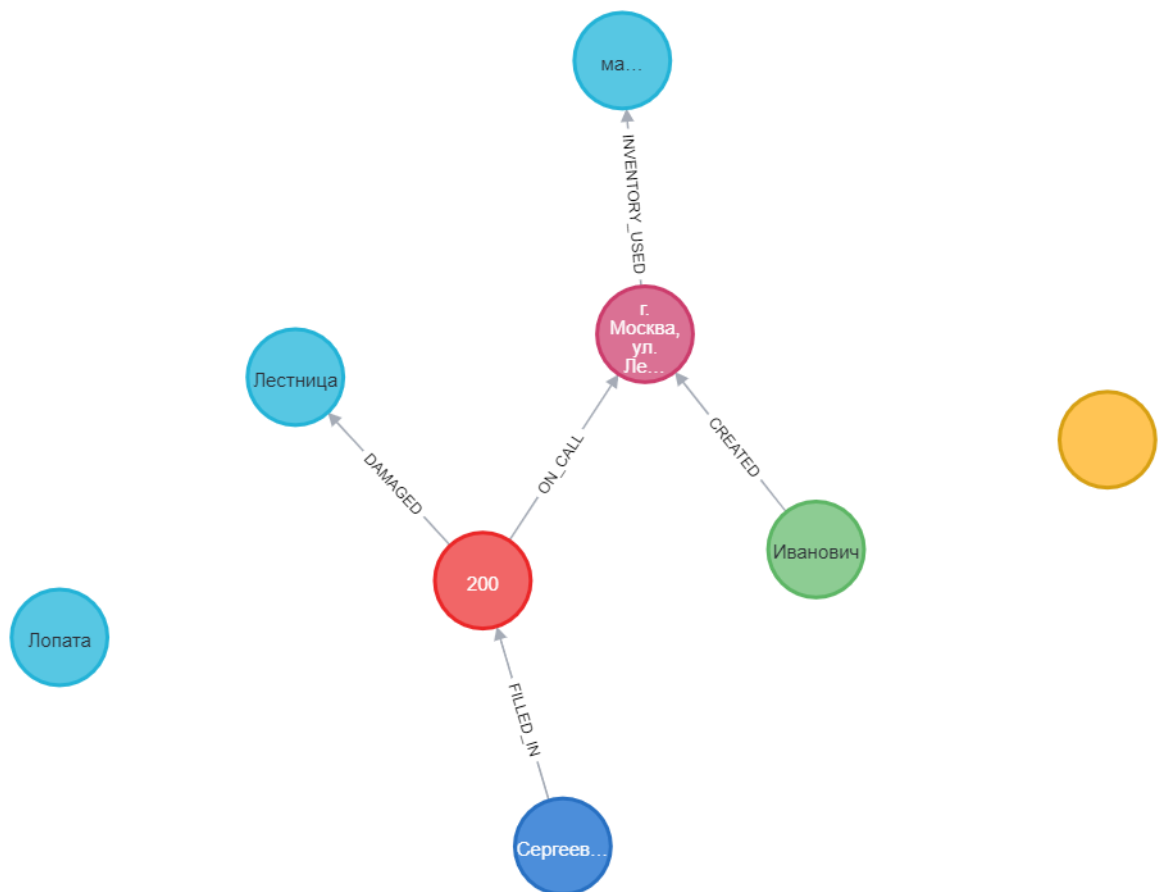


Рисунок 3 – Графическое представление данных в нереляционной СУБД (Neo4j)

Примеры запросов к базе данных представлены в листинге 2. Заметим, что для всех юзкейсов применяется ровно один запрос.

Листинг 2 – Возможные запросы для нереляционной базы данных

```

// Создание пользователей
CREATE (:User:Operator:Active {
  familyName: "Иванов",
  firstName: "Иван",
  fatherName: "Иванович",
  brigadeNumber: 1,
  address: "г. Москва, ул. Ленина, д. 1",
  phone: "+7 (123) 456-78-90",
  email: "ivanov@example.com",
  login: "ivanov",
  passwordHash: "hashed_password_1",
  registeredAt: timestamp(),
  modifiedAt: timestamp()
});

// Поиск сотрудника
MATCH (u:User)
WHERE u.login= 'ivanov' OR u.familyName='Иванов'
RETURN u;

// Редактирование сотрудника

```

```

MATCH (u:User {Login: 'ivanov'})
SET u.address= 'Чебоксары',
    u.phone = '88005553535 ',
    u.modifiedAt = timestamp()
RETURN u;

// Поиск отчётов по бригадиру
MATCH (u:User:Brigadier)-[:FILLED_IN]->(r:Report)
WHERE u.firstName= 'Терентий'
RETURN r;

// Удаление отчёта
MATCH (r:Report {identity: 20})
DETACH DELETE r;

// Сбор статистики по потреблению воды и пены за период
MATCH (r:Report:Complete)
WHERE r.modifiedAt >= date('2023-01-01') AND r.modifiedAt <= date('2023-
12-31')
RETURN SUM(r.waterSpent) AS TotalWaterSpent,
        SUM(r.foamSpent) AS TotalFoamSpent;

// Создание формы вызова
MATCH (u:User {login: "ivanov"})
CREATE (cf:CallForm:Incomplete {
    createdAt: timestamp(),
    modifiedAt: timestamp(),
    callSource: "телефонный звонок",
    fireAddress: "г. Москва, ул. Ленина, д. 1",
    bottomLeft: point({latitude: 55.7558, longitude: 37.6173}),
    topRight: point({latitude: 55.7560, longitude: 37.6180}),
    fireType: "лесной пожар",
    fireRank: "3",
    victimsCount: 0,
    assignedTo: null,
    auto: null
})
CREATE (u)-[:CREATED]->(cf)
CREATE (cf)<-[:CREATED_BY]-(u);

// Поиск незавершенных форм
MATCH (cf:CallForm:Incomplete)
RETURN cf;

// Редактирование формы
MATCH (cf:CallForm {identity: 12})
SET cf.callSource = 'новый_источник_вызова',
    cf.localization = 'новая_локализация',
    cf.modifiedAt = timestamp()
RETURN cf;

// Закрытие формы и создание нового отчета
MATCH (cf:CallForm {identity: 11})
SET cf :Complete,
    cf.modifiedAt = timestamp()
CREATE (r:Report:New {
    CreatedAt: timestamp(),
    ModifiedAt: timestamp()

```

```

}))
CREATE (r)-[:ON_CALL]->(cf)
RETURN cf, r;

// Поиск завершенных форм
MATCH (cf:CallForm:Complete)
RETURN cf;

// Получение списка незавершенных форм для бригады пожарного
MATCH (u:User:Fireman)-[:CREATED]->(cf:CallForm:Incomplete)
WHERE u.brigadeNumber = 6
RETURN cf;

// Получение списка бригад и их занятости (бригада - число активных вызовов)
MATCH (u:User)
WHERE u.brigadeNumber IS NOT NULL
WITH u.brigadeNumber AS brigadeNumber, COUNT(cf) AS activeCalls
OPTIONAL MATCH (cf:CallForm:Incomplete)
WHERE u.brigadeNumber = cf.assignedTo
RETURN brigadeNumber, activeCalls;

// Поиск завершенных отчетов бригады, отсортированных по дате вызова
MATCH (u:User:Brigadier)-[:FILLED_IN]->(r:Report:Complete)-[:ON_CALL]->(cf:CallForm:Complete)
WHERE u.brigadeNumber = 7
ORDER BY cf.createdAt DESC
RETURN r;

// Поиск завершенных отчетов бригады, отсортированных по времени сохранения
MATCH (u:User:Brigadier)-[:FILLED_IN]->(r:Report:Complete)-[:ON_CALL]->(cf:CallForm:Complete)
WHERE u.brigadeNumber = 7
ORDER BY r.modifiedAt DESC
RETURN r;

// Поиск новых отчетов после закрытия форм
MATCH (r:Report:New)-[:ON_CALL]->(cf:CallForm:Complete)
RETURN r;

// Заполнение отчета как черновика
MATCH (u:User:Brigadier)-[:FILLED_IN]->(r:Report { identity: 10 })
REMOVE r:New
SET r:Incomplete,
    r.waterSpent = 2004,
    r.modifiedAt = timestamp()
RETURN r;

// Окончательное сохранение отчета
MATCH (u:User:Brigadier)-[:FILLED_IN]->(r:Report { identity: 10 })
REMOVE r:New:Incomplete
SET r:Complete,
    r.foamSpent = 2026,
    r.modifiedAt = timestamp()
RETURN r;

```

Реляционная модель данных

Сущности модели:

Сотрудник (User): ID, Фамилия (Family Name), Имя (First Name), Отчество (Father Name) (может отсутствовать), Должность (Role) (одно из четырёх: оператор / администратор / бригадир / пожарный), Номер бригады (Brigade Number) (обязательно для бригадира/пожарного, отсутствует для оператора/администратора), Адрес проживания (Address), Номер телефона (Phone), Адрес электронной почты (EMail), Логин (Login) (должен быть уникальным), Хэш пароля (Password Hash), Время регистрации (Registered At), Время последнего изменения (Modified At), Статус (status) (активен / удалён).

Форма вызова (Call Form): ID, Статус (Status) (незавершён / завершён), Время создания (Created At), Последнее обновление (Modified At) (время завершения для завершённого вызова, так как редактировать их уже нельзя), Источник вызова (Call Source), Локализация пожара, Адрес (Address) 2-5. Axis Aligned Bounding Box (прямоугольник на карте, охватывающий пожар, либо две точки, либо одна точка и высоты/ширина прямоугольника), Характер пожара (Type), Ранг пожара (Fire Rank) (Одно из: 1, 1-БИС, 2, 3, 4), Пострадавшие (Victims) (Количество), Прикреплённая бригада (Assigned To), Автомобиль (Auto), Оператор (Operator ID, FK).

Инвентарь (Inventory): ID, Наименование предмета (Name).

Отчёт (Report): ID, Расход воды (Water Spent), Расход пены (Foam Spent), Предполагаемая причина пожара (Alleged Fire Cause), Оценка ущерба (Damage), Дополнительная информация (Additional Notes), ID Формы Вызова (CallForm ID, FK), Бригадир (Brigadier ID, FK), Статус (Status), Время сохранения (Saving Time).

Используемый Инвентарь (Used Inventory): ID Формы Вызова (CallForm ID, FK), ID Инвентаря (Inventory ID, FK).

Повреждённый Инвентарь (Damaged Inventory): ID Отчёта (Report ID, FK), ID Инвентаря (Inventory ID, FK).

Оценка объема информации (реляционная модель)

Пусть S_u , S_c , S_i , S_r , S_{ui} , S_{di} - средние размеры объектов типа Сотрудник, Форма вызова, Инвентарь, Отчет, Используемый инвентарь и Поврежденный инвентарь. Рассчитаем примерные значения объёма информации:

$S_u = \text{ID (4 байт)} + \text{Объем } S_u \text{ в NoSQL (279 байт)} = 283 \text{ байт}$. К оценке объема памяти добавляется числовое значение ID.

$S_c = \text{ID (4 байт)} + \text{Объем } S_c \text{ в NoSQL (162 байт)} + \text{ID Оператора (4 байт)} = 170 \text{ байт}$.

$S_i = \text{ID (4 байт)} + \text{Объем } S_i \text{ в NoSQL (30 байт)} = 34 \text{ байт}$.

$S_r = \text{ID (4 байт)} + \text{Объем } S_r \text{ в NoSQL (276 байт)} + \text{ID Формы вызова (4 байт)} + \text{ID Бригадира (4 байт)} = 288 \text{ байт}$.

$S_{ui} = \text{ID Формы вызова (4 байт)} + \text{ID Инвентаря (4 байт)} = 8 \text{ байт}$.

$S_{di} = \text{ID Отчета (4 байт)} + \text{ID Инвентаря (4 байт)} = 8 \text{ байт}$.

Тогда получим формулу $V(S_u, S_c, S_i, S_r, S_{ui}, S_{di}) = 283N_u + 170N_c + 34N_i + 288N_r + 8N_{ui} + 8N_{di}$, где N_x – количество x , u – Сотрудники, c – Форма вызова, i – Инвентарь, r – Отчёт, ui – Используемый инвентарь, di – Поврежденный инвентарь.

Исходя из анализа NoSQL, положим $N_r = N_c$ и $N_u = 0.01N_c$, $N_i = 0.1N_c$. Также $N_{ui} = 2N_c$ (если в среднем 2 предмета за вызов) и $N_{di} = 0.1N_c$ (раз в 10 вызовов предмет выходит из строя). Тогда получим $2.83N_c + 170N_c + 3.4N_c + 288N_c + 16N_c + 0.8N_c = 481.03N_c$ байт.

В описанной модели присутствует избыточность: если ID инвентаря дублируется в таблицах Используемого и Поврежденного инвентарей, то получим излишний объем памяти в 8 байт. Тогда чистый объем памяти составит $481.03N - 8N = 473.03N$. Получим избыточность модели: $481.03N / 473.03N = 1,01691$ В случае, если один ID Инвентаря находится только в одной из таблиц, избыточность составит 1.

Направление роста модели при увеличении количества объектов каждой сущности:

Сотрудники: сотрудник занимает большой объем памяти, но их кол-во

растет очень медленно, поэтому влияние на рост модели маленькое.

Форма вызова: одна такая запись занимает большой объем, причем их кол-во будет быстро расти. Влияние на рост модели большое.

Отчет: запись составляет большой объем памяти и имеет отношение 1 к 1 с Формой вызова. Влияние на рост наибольшее.

Инвентарь: объем записи невелик, а их кол-во обычно небольшое. Влияние очень маленькое.

Используемый инвентарь: кол-во используемого инвентаря еще меньше, чем сам инвентарь, его влияние еще меньше.

Поврежденный инвентарь: кол-во поврежденных инструментов растет медленно, поэтому влияние очень маленькое.

Примеры запросов для базы данных (реляционная)

Примеры запросов для заполнения базы данных и создание связей представлены в листинге 3.

Листинг 3 - Заполнение базы данных (реляционная модель)

```
CREATE TABLE Сотрудник (  
    ID INT AUTO_INCREMENT PRIMARY KEY,  
    ФИО VARCHAR(60) NOT NULL,  
    Должность VARCHAR(10) NOT NULL,  
    Номер_бригады INT,  
    Телефон VARCHAR(11),  
    Email VARCHAR(30),  
    Логин VARCHAR(24),  
    Время_регистрации DATETIME DEFAULT CURRENT_TIMESTAMP,  
    Время_последнего_изменения DATETIME,  
    Статус VARCHAR(10)  
);  
  
CREATE TABLE Форма_вызова (  
    ID INT AUTO_INCREMENT PRIMARY KEY,  
    Статус VARCHAR(10),  
    Последнее_обновление DATETIME,  
    Источник_вызова VARCHAR(20),  
    Локализация_пожара VARCHAR(32),  
    Характер_пожара VARCHAR(20),  
    Ранг_пожара VARCHAR(10),  
    Пострадавшие VARCHAR(),  
    Прикрепленная_бригада INT,  
    Автомобиль VARCHAR(10),  
    Оператор INT  
);  
  
CREATE TABLE Отчет (  
    ID INT AUTO_INCREMENT PRIMARY KEY,  
    Расход воды INT,  
    Расход пены INT,  
    Предполагаемая_причина_пожара VARCHAR(50),  
    Оценка_ущерба VARCHAR(100),  
    Дополнительная информация VARCHAR(100),  
    Форма_вызова INT,  
    Бригадир INT,  
    Статус VARCHAR(10),  
    Время_заполнения DATETIME DEFAULT CURRENT_TIMESTAMP  
);  
  
CREATE TABLE Инвентарь (  
    ID INT AUTO_INCREMENT PRIMARY KEY,  
    Имя_предмета VARCHAR(30)  
);  
  
CREATE TABLE Используемый_инвентарь (  
    Форма_вызова INT,  
    Предмет INT  
);
```



```
CREATE TABLE Поврежденный_инвентарь (
    Отчет INT,
    Предмет INT
);
```

Пример данных представлен на рисунке 4.

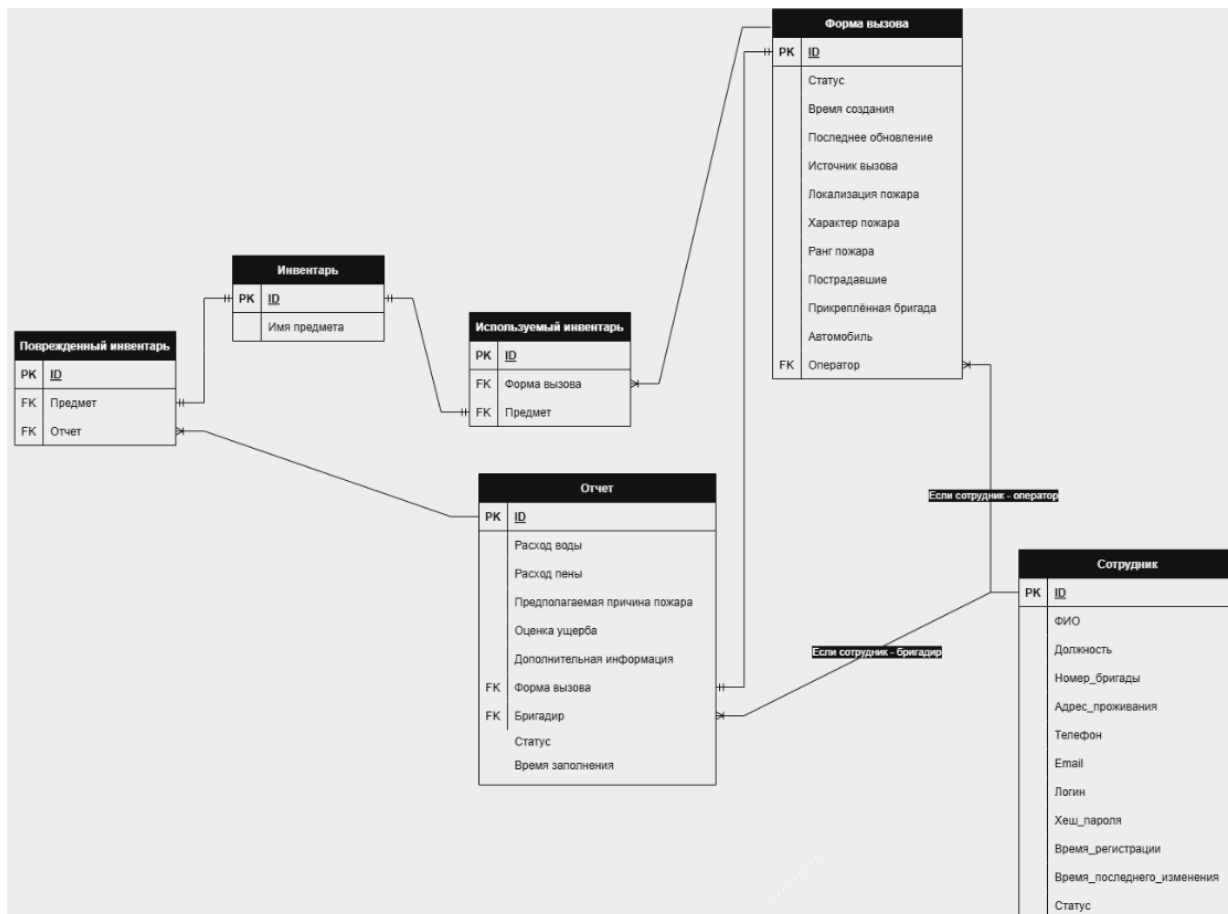


Рисунок 4 – Графическое представление данных в не реляционной СУБД (Neo4j)

Примеры запросов к базе данных представлены в листинге 4.

Листинг 4 – Возможные запросы для реляционной базы данных

```
-- Создание пользователей
INSERT INTO Сотрудник (
    ФИО, Должность, Номер_бригады, Адрес_проживания, Телефон, Email,
    Login, Хеш_пароля, Время_регистрации, Время_последнего_изменения, Статус
) VALUES (
    'Иванов Иван Иванович', 'Пожарный', 3, 'ул. Пожарная, д. 15',
    '+79123456789', 'ivanov@example.com', 'ivanov', 'hashed_password_123',
    NOW(), NOW(), 'Активный'
);

-- Поиск сотрудника
SELECT *
FROM Сотрудник
WHERE
    (ФИО = 'Иванов Иван Иванович' OR ФИО IS NULL) AND
    (Должность = 'Пожарный' OR Должность IS NULL) AND
```

```

        (Номер_бригады = 3 OR Номер_бригады IS NULL) AND
        (Время_регистрации = '2023-10-01 10:00:00' OR Время_регистрации IS
NULL) AND
        (Время_последнего_изменения = '2023-10-01 10:00:00' OR
Время_последнего_изменения IS NULL);
-- Редактирование сотрудника
UPDATE Сотрудник
SET
    ФИО = 'Петров Петр Петрович',
    Должность = 'Командир',
    Номер_бригады = 5,
    Адрес_проживания = 'ул. Огнеборцев, д. 20',
    Телефон = '+79123456788',
    Email = 'petrov@example.com',
    Логин = 'petrov',
    Хеш_пароля = 'hashed_password_456',
    Время_регистрации = '2023-10-02 11:00:00',
    Время_последнего_изменения = NOW(),
    Статус = 'Неактивен'
WHERE ID = 1;
-- Поиск отчетов
SELECT *
FROM Отчет
WHERE Бригадир = 1;
-- Удаление отчета
DELETE FROM Отчет
WHERE ID = 1;
-- Сбор записей о потребленном объеме воды и пены для сбора статистики за
определенный период времени
SELECT
    Расход_воды,
    Расход_пены,
    Время_создания
FROM Отчет
WHERE
    Время_создания BETWEEN '2023-10-01' AND '2023-10-02';
-- Создание формы
INSERT INTO Форма_вызова (
    Статус,    Время_создания,    Последнее_обновление,    Источник_вызова,
    Локализация_пожара,    Характер_пожара,    Ранг_пожара,    Пострадавшие,
    Прикреплённая_бригада,    Автомобиль,    Оператор
) VALUES (
    'В процессе', NOW(), NOW(), 'Телефонный вызов', 'ул. Ленина, д. 10',
    'Жилой дом', '2', 0, '3', 'Пожарная машина №1', 1
);
-- Поиск незавершенных форм
SELECT *
FROM Форма_вызова
WHERE Статус = "Не завершено"
-- Редактирование формы
UPDATE Форма_вызова
SET
    Статус = "Открыта",
    Последнее_обновление = NOW(),
    Источник_вызова = 'Телефонный вызов',
    Локализация_пожара = 'ул. Ленина, д.10',
    Характер_пожара = 'Жилой дом',
    Ранг_пожара = 2,

```

```

        Пострадавшие = 0,
        Прикреплённая_бригада = 3,
        Автомобиль = 'Пожарная машина №1",
        Оператор = 1
WHERE ID = 1;
-- Закрытие формы и создание нового отчета
UPDATE Форма_вызова
SET
    Статус = 'Закрыта',
    Последнее_обновление = NOW(),
    Источник_вызова = 'Телефонный вызов',
    Локализация_пожара = 'ул. Ленина, д.10',
    Характер_пожара = 'Жилой дом',
    Ранг_пожара = 2,
    Пострадавшие = 0,
    Прикреплённая_бригада = 3,
    Автомобиль = 'Пожарная машина №1',
    Оператор = 1
WHERE ID = 1;

INSERT INTO Отчет (
    Расход_воды,          Расход_пены,          Предполагаемая_причина_пожара,
    Оценка_ущерба,        Дополнительная_информация,  Форма_вызова,  Бригадир,  Статус,
    Время_заполнения
) VALUES (
    NULL, NULL, NULL, NULL, NULL, 1, (SELECT ID FROM Сотрудник WHERE
    (Должность = 'Бригадир' AND Номер_бригады = (SELECT Прикреплённая_бригада
    FROM Форма_вызова WHERE ID = 1))), 'Новый', NOW()
);
-- Поиск завершённых форм
SELECT *
FROM Форма_вызова
WHERE Статус = "Завершено"
-- Получение списка незавершённых форм для бригады пожарного
SELECT *
FROM Форма_вызова
WHERE (
    Статус = "Не завершено" AND Бригада = "3"
);
-- Получение списка бригад и их состояния
SELECT
    С.Бригада AS Номер_бригады,
    COUNT(*) AS Размер_бригады,
    MAX(Ф.Время_создания) AS Время_последнего_вызова,
    COUNT(Ф_сегодня.ID) AS Кол-во_вызовов_за_смену,
    CASE
        WHEN MAX(CASE WHEN Ф.Статус = 'Не завершено' THEN 1 ELSE 0 END) =
1
        THEN 'Занята'
        ELSE 'Свободна'
    END AS Статус_бригады
FROM Сотрудник С
JOIN Форма_вызова Ф ON С.Бригада= Ф.Прикреплённая_бригада
LEFT JOIN (
    SELECT ID, Прикреплённая_бригада
    FROM Форма_вызова
    WHERE DATE(Время_создания) = CURDATE()
) Ф_сегодня ON С.Номер_бригады = Ф_сегодня.Прикреплённая_бригада

```

```

GROUP BY С.Номер_бригады;
ORDER BY
    Статус_бригады ASC,
    Время_последнего_вызова ASC,
    Кол-во_вызовов_за_смену ASC
-- Поиск завершенных отчетов по дате вызова
SELECT *
FROM Отчет AS O
JOIN Форма_вызова ON O.Форма_вызова = Форма_вызова.ID
WHERE Статус = "Заполнен" AND Бригадир = (SELECT ID FROM Сотрудник WHERE
Должность = 'Бригадир' AND Номер_бригады = 3)
ORDER BY
    Форма_вызова.Время_создания ASC
-- Поиск завершенных отчетов по времени сохранения
SELECT *
FROM Отчет AS O
WHERE Статус = "Заполнен" AND Бригадир = (SELECT ID FROM Сотрудник WHERE
Должность = 'Бригадир' AND Номер_бригады = 3)
ORDER BY
    Время_заполнения ASC
-- Поиск новых отчетов после закрытия форм
SELECT *
FROM Отчет
WHERE Статус = "Новый"
ORDER BY
    Время_заполнения ASC
-- Поиск отчетов-черновиков
SELECT *
FROM Отчет
WHERE Статус = "Черновик"
ORDER BY
    Время_заполнения ASC
-- Сохранение отчета как черновика
UPDATE Отчет
SET
    Расход_воды = 600.0,
    Расход_пены = 60.0,
    Причина_пожара = 'Неисправность электропроводки',
    Оценка_ущерба = 120000.0,
    Дополнительная_информация = 'Пожар локализован в течение 45 минут',
    Время_заполнения = NOW(),
    Статус = "Черновик"
WHERE ID = 1;
-- Окончательное сохранение отчета
UPDATE Отчет
SET
    Расход_воды = 600.0,
    Расход_пены = 60.0,
    Причина_пожара = 'Неисправность электропроводки',
    Оценка_ущерба = 120000.0,
    Дополнительная_информация = 'Пожар локализован в течение 45 минут',
    Время_заполнения = NOW(),
    Статус = "Заполнен"
WHERE ID = 1;

```

Обратим внимание, что для совершения почти всех юзкейсов достаточно одного запроса. Исключения: редактирование сотрудника требует 2 запроса (Поиск ID сотрудника, обновление данных), поиск отчётов требует 2 запроса (Поиск ID бригадира, Поиск отчетов), Редактирование формы требует 2 запроса (Поиск незакрытых форм, Редактирование), Заккрытие формы и создание нового отчета требует 4 запросов (Поиск незакрытых форм, Заккрытие формы, Поиск номера прикрепленной бригады, Поиск бригадира, Создание нового отчета), Получение списка бригад и их состояния требует 2 запроса (Подсчет кол-ва выездов за сегодняшнюю смену, Составление таблицы с существующими бригадами), Поиск завершенных отчетов по дате вызова требует 2 запроса (Поиск ID бригадира, Поиск отчетов по вызовам бригады), Поиск завершенных отчетов по времени сохранения требует 2 запроса (Поиск ID бригадира, Поиск отчетов по вызовам бригады).

Сравнение моделей

Сравним удельные объёмы информации: для реляционной 463N (избыточность 1.0175), для нереляционной 492.59N (избыточность 1.0165). SQL-модель немного опережает NoSQL-модель в использовании меньшего объема памяти, избыточность вышла примерно одинаковой. Также сравним количество запросов по юзкейсам: количество запросов для всех юзкейсов NoSQL-модели равно 1, тогда как в SQL-модели для некоторых юзкейсов требуется более одного запроса. Поэтому очевидно, что NoSQL выигрывает по количеству запросов.

Вывод

Для описанного случая преимущества нереляционной модели могут быть не столь значимыми, так что вполне можно использовать и SQL-модель, занимающую меньше места (если считать исключительно данные). Нереляционная модель же демонстрирует возможность более лаконичных запросов и позволяет выполнять одним запросом любую из поставленных ранее

задач. Использование меток Neo4j хорошо ложится на задачи, поэтому в итоговом зачёте графовая база данных лучше подходит для решения поставленных задач.

5. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

Краткое описание

Backend-часть системы реализована как серверное приложение на Node.js с использованием фреймворка Express. В качестве базы данных выбрана графовая СУБД Neo4j, которая идеально подходит для работы со сложными взаимосвязями между сущностями. Сервер работает на порту 3000 и настроен для кросс-доменных запросов через CORS, разрешая взаимодействие с фронтендом, работающим на <http://localhost:5173>.

API сервера построено по REST-архитектуре и предоставляет полный набор эндпоинтов для выполнения CRUD-операций со всеми основными сущностями системы. Особенностью реализации является использование подписок на изменения данных, что позволяет реализовать реактивное обновление информации в реальном времени. Для работы с Neo4j используется официальный драйвер, а запросы к базе данных формируются с помощью удобной системы-обертки, упрощающей построение сложных графовых запросов.

Frontend-часть представляет собой одностраничное приложение (SPA), разработанное на Vue 3 с использованием современной экосистемы инструментов. Маршрутизация реализована через Vue Router с поддержкой истории браузера. Состояние приложения управляется с помощью Pinia - официального решения для state-менеджмента во Vue, с дополнительным плагином для сохранения состояния между сессиями пользователя.

Интерфейс разделен на несколько функциональных зон в зависимости от роли пользователя (пожарный, диспетчер, администратор), с соответствующей защитой маршрутов. Для картографического функционала используется библиотека Leaflet. Приложение взаимодействует с бэкендом через REST API, отправляя запросы и обрабатывая ответы в соответствии с бизнес-логикой системы.

Приложение разворачивается через Docker Compose и состоит из двух

сервисов: js, объединяющий backend на Node.js/Express и frontend на Vue в одном контейнере с портами 3000 и 5173 соответственно, и db – контейнер с Neo4j для хранения данных.

Инструкция по запуску проекта представлена в приложениях.

Схема экранов приложения

Схема экранов приложения представлена в разделе «Сценарии использования» или на GitHub по ссылке: <https://github.com/moevm/nosql1h25-unirepair> в разделе Wiki «Макет и сценарии использования».

Использованные технологии

Backend: JavaScript (Express), Frontend: JavaScript (Vue), база данных: neo4j.

Ссылка на приложение

Ссылка на github: <https://github.com/moevm/nosql1h25-unirepair>.

6. ВЫВОД

Результаты

Разработано приложение для управления оперативной деятельностью пожарных бригад, представляющее собой полноценное fullstack-решение с backend на Node.js/Express, работающим с графовой СУБД Neo4j, и frontend на Vue. Система обеспечивает полный цикл управления вызовами - от создания заявки оператором до формирования отчётов бригадами, с чётким ролевым разделением (пожарные, бригадиры, диспетчеры, администраторы) и специализированными интерфейсами для каждой группы пользователей. Предусмотрено контейнеризированное развёртывание через Docker Compose (два сервиса).

Недостатки и пути для улучшения полученного решения

К недостаткам приложения можно отнести:

- 1) Слабо развитую мобильную версию приложения (и ее отсутствие у операторов и администраторов).
- 2) Отсутствие автоматических бэкапов (предусмотрен лишь бекап силами пользователя-администратора через frontend).
- 3) Отсутствие проверки авторизации пользователя на стороне сервера.

Пути улучшения приложения – исправление описанных выше недочётов.

Будущее развитие решения

Для повышения доступности использования приложения возможно добавление версий для слабовидящих/ дальтоникиков у операторов и администраторов.

7. ПРИЛОЖЕНИЯ

Для сборки и развёртывания приложения приложены следующие инструкции:

На Windows:

- 1) Откройте терминал в корневой директории проекта (все последующие команды проводить в этом же терминале).
- 2) Введите команду: `.\clean_run.cmd` (если необходим запуск без сохранения БД для последующих запусков) или команду `.\prod_run.cmd`.
- 3) Дождитесь выполнения всех тестов и сообщения: "A total of 36 tests passed, 0 tests failed!" (в случае запуска без сохранения БД).
- 4) Перейдите по ссылке: <http://localhost:5173/>.
- 5) Для завершения работы используйте сочетание клавиш: Ctrl + C. Затем ввести `.\stop.cmd` для выключения БД.

На Linux:

- 1) Откройте терминал в корневой директории проекта (все последующие команды проводить в этом же терминале).
- 2) Введите команду: `./clean_run.sh` (если необходим запуск без сохранения БД) или команду `./prod_run.sh`.
- 3) Дождитесь выполнения всех тестов и сообщения: "A total of 36 tests passed, 0 tests failed!" (в случае запуска без сохранения БД).
- 4) Перейдите по ссылке: <http://localhost:5173/>.
- 5) Для завершения работы используйте сочетание клавиш: Ctrl + C, затем введите `docker-compose down`.

Примечания:

Если вам необходимо сохранение данных между сессиями, то для запуска используйте команды `.\prod_run.cmd` или `./prod_run.sh` описанные выше.

8. ИСПОЛЬЗУЕМАЯ ЛИТЕРАТУРА

1. Документация к Neo4j: <https://neo4j.com/docs/>

2. Документация к Vue.js: <https://ru.vuejs.org/guide/introduction.html>
3. Документация к Express: <https://expressjs.com/en/api.html>