

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

**ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ**  
**по дисциплине «Введение в нереляционные СУБД»**  
**Тема: ИТ-каталог**

Студентка гр. 2381

---

Малюсская Д.И.

Студенты гр. 2383

---

Медведский Д.Ю.

---

Мещеряков Р.В.

---

Сериков М.

---

Шмонова Н.А.

---

Заславский М.М.

Преподаватель

---

Санкт-Петербург

2025

## **ЗАДАНИЕ**

Студенты

Малюсая Д.И. (группа 2381)

Медведский Д.Ю.

Мещеряков Р.В.

Сериков М.

Шмонова Н.А.

Группа 2383

Тема работы: ИТ-каталог

Исходные данные:

Задача - сделать сервис, где будет база данных по всевозможным ИТ профессиям, навыкам, инструментам, технологиям. Интересно построить граф, в котором можно будет увидеть как взаимосвязи разных понятий, так и их востребованность на текущий момент (например, по данным github или открытых рейтингов).

Полный текст задания: [ИДЗ - состав, порядок работы \[МОЭВМ Вики \[se.moevm.info\]\]](#)

Содержание пояснительной записки:

- Содержание
- Введение
- Сценарии использования
- Модель данных
- Разработанное приложение
- Заключение
- Приложения
- Литература

Предполагаемый объем пояснительной записки:

Не менее 20 страниц.

Дата выдачи задания: 16.02.2025

Дата сдачи реферата: 22.05.2025

Дата защиты реферата: 22.05.2025

Студентка гр. 2381

Малюская Д.И.

Студенты гр. 2383

Медведский Д.Ю.

Преподаватель

Мещеряков Р.В.

Сериков М.

Шмонова Н.А.

Заславский М.М.

## **АННОТАЦИЯ**

В рамках данного курса предполагалось разработать какое-либо приложение в команде на одну из поставленных тем. Была выбрана тема создания приложения для хранения сущностей в ИТ-каталоге. Требуется реализовать хранение данных в графовой СУБД Neo4j. Найти исходный код и всю дополнительную информацию можно по ссылке:  
<https://github.com/moevm/nosql1h25-itcatalog>

## **SUMMARY**

As part of this course, students were expected to develop an application in teams based on one of the proposed topics. The chosen topic was the creation of an application for storing entities in an IT catalog. The task requires implementing data storage using the Neo4j graph database. The source code and all additional information can be found at the following link:  
<https://github.com/moevm/nosql1h25-itcatalog>

## СОДЕРЖАНИЕ

1.	Введение	6
1.1.	Актуальность решаемой проблемы	6
1.2.	Постановка задачи	6
1.3.	Предлагаемое решение	7
1.4.	Качественные требования к решению	7
2.	Сценарии использования	8
2.1.	Макет UI	8
2.2.	Сценарии использования для различных задач	9
2.3.	Вывод	11
3.	Модель данных	12
3.1.	Нереляционная модель	12
3.2.	Реляционная модель	22
3.3.	Сравнение моделей	34
4.	Разработанное приложение	36
4.1.	Краткое описание	36
4.2.	Использованные технологии	37
4.3.	Снимки экрана приложения	38
5.	Заключение	42
5.1.	Достигнутые результаты	42
5.2.	Недостатки и пути для улучшения полученного решения	42
5.3.	Будущее развитие решения	43
6.	Приложения	44
6.1.	Документация по сборке и развертыванию приложения	44
6.2.	Инструкция для пользователя	44
7.	Список использованных источников	46

# **ВВЕДЕНИЕ**

## **1. Введение**

### **1.1. Актуальность решаемой проблемы**

ИТ-индустрия развивается стремительными темпами, что приводит к постоянному появлению новых профессий, технологий и инструментов. Это вызывает сложности при ориентации в отрасли как у начинающих специалистов, так и у работодателей, студентов и аналитиков. Возникает потребность в структурированном представлении взаимосвязей между ИТ-сущностями и в оценке их актуальности.

### **1.2. Постановка задачи**

#### **Цель работы**

Разработка веб-сервиса для хранения и визуализации информации об ИТ-профессиях, навыках, инструментах и технологиях на основе графовой базы данных Neo4j.

#### **Задачи**

- Спроектировать структуру графа, отражающего взаимосвязи между ИТ-профессиями, навыками, инструментами и технологиями.
- Реализовать хранение данных с использованием графовой СУБД Neo4j.
- Разработать пользовательский интерфейс для взаимодействия с каталогом: просмотр, добавление и редактирование сущностей и связей.
- Организовать визуализацию графа, чтобы пользователь мог видеть связи между понятиями.
- Интегрировать данные о популярности или востребованности сущностей (например, на основе статистики GitHub или других открытых источников).

- Обеспечить удобную навигацию и поиск по каталогу.
- Документировать проект и подготовить инструкции по установке и использованию.

### **1.3. Предлагаемое решение**

В рамках проекта предлагается построить ИТ-каталог на основе графовой базы данных Neo4j. Данные представлены в виде узлов (сущности) и рёбер (связи), что позволяет эффективно моделировать и визуализировать сложные структуры. Реализован веб-интерфейс для просмотра, редактирования и поиска по графу, а также механизмы для интеграции внешних источников данных.

### **1.4. Качественные требования к решению**

В рамках разработки веб-приложения особое внимание уделяется качественным характеристикам системы, обеспечивающим её устойчивость, удобство использования и готовность к дальнейшему развитию. Ниже перечислены ключевые требования, которым должно соответствовать итоговое решение:

- надёжность и согласованность данных — система должна обеспечивать целостность информации при любых действиях пользователя и в условиях возможных сбоев;
- удобный и понятный пользовательский интерфейс — все функции должны быть легко доступны, логично организованы и не вызывать затруднений при использовании;
- визуализация графа в интерактивной форме — структура связей и элементов отображается в виде наглядного графа с возможностью взаимодействия (наведение, клики, масштабирование);
- гибкость и масштабируемость архитектуры — решение проектируется так, чтобы было легко дорабатывать новые функции, подключать другие источники данных и масштабировать при увеличении нагрузки.

## **2. СЦЕНАРИИ ИСПОЛЬЗОВАНИЯ**

**2.1. Макет UI (<https://clck.ru/3MAQDk> - в хорошем качестве)**

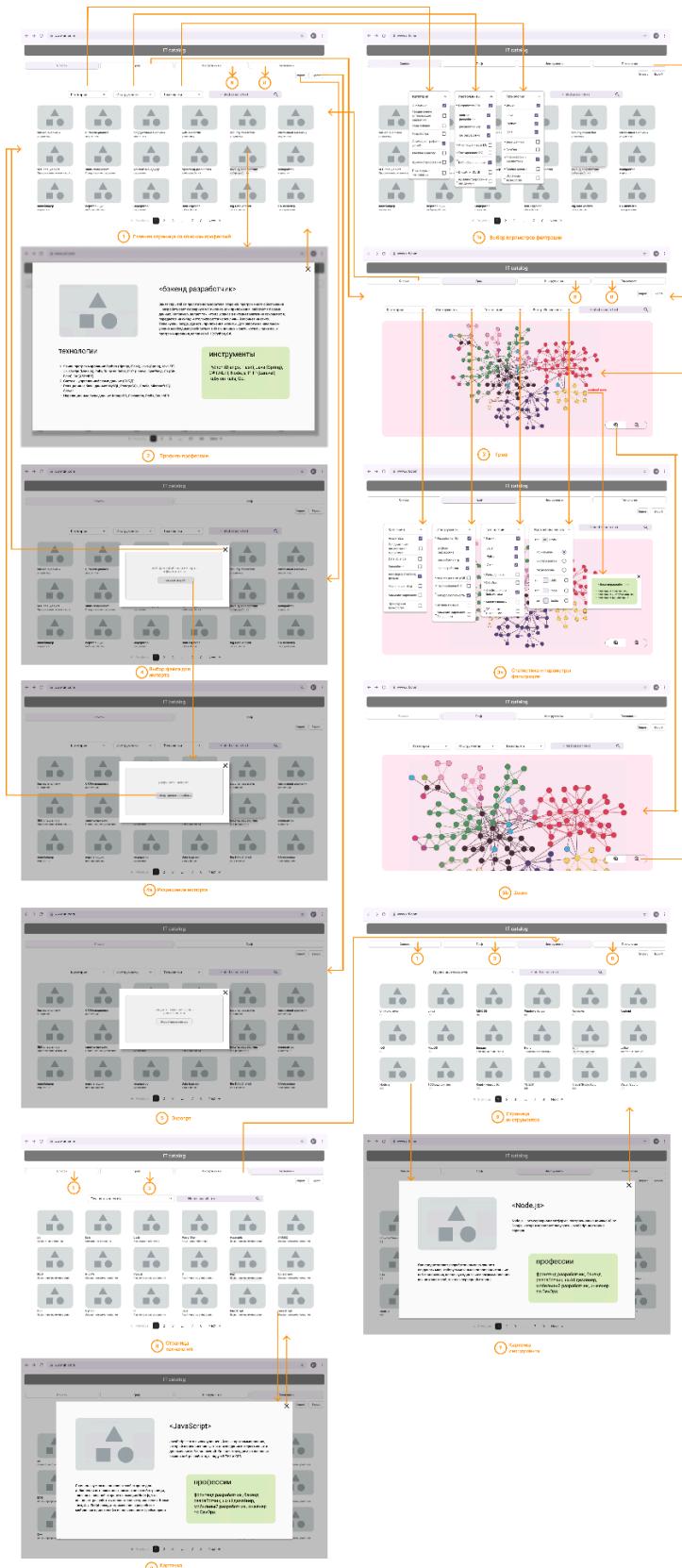


Рисунок 1 – Макет UI

## **2.2. Сценарии использования для различных задач**

Действующее лицо: Пользователь

### **1. Импорт данных**

#### **1.1. Массовый импорт (из файла или внешнего источника)**

Предусловие: Пользователь находится на главной странице.

Основной сценарий:

Пользователь нажимает кнопку «Импорт» в верхнем меню.

Выбирает опцию «Загрузить из файла»

Загружает файл с устройства

Система проверяет формат, размер и корректность данных.

При успешной проверке данные импортируются в систему.

Альтернативный сценарий:

Если формат некорректен: система выводит сообщение с требованиями к данным.

#### **1.2. Ручной ввод данных**

Основной сценарий:

Пользователь нажимает «Добавить запись» в разделе администрирования.

Заполняет форму с полями:

Название профессии.

Описание.

Связанные категории, технологии, инструменты (через выпадающие списки или теги).

Нажимает «Сохранить».

Система добавляет запись в базу данных и обновляет интерфейс.

Альтернативный сценарий:

Если обязательные поля не заполнены: система подсвечивает их и блокирует сохранение.

## **2. Представление данных**

## 2.1. Просмотр списка профессий

Пользователь видит таблицу с колонками: «Название», «Категория», «Технологии», «Инструменты».

Фильтрация:

Использует выпадающие фильтры по категориям, технологиям, инструментам.

Вводит ключевые слова в поисковую строку.

Сортировка: Кликает по заголовкам колонок для сортировки.

## 2.2. Визуализация данных

Графический режим (граф):

Пользователь переключается на вкладку «Граф».

Взаимодействует с узлами (перетаскивание, клик для деталей).

Использует кнопки «+»/«-» для масштабирования.

## 3. Анализ данных

### 3.1. Статистический анализ

Основной сценарий:

Пользователь открывает раздел «Аналитика».

Выбирает тип анализа:

Подсчет средних значений (например, среднее количество навыков на профессию).

Топ-N профессий по количеству технологий/инструментов.

Распределение профессий по категориям.

Задает параметры (например, N=10 для топа).

Нажимает «Сгенерировать отчет».

Система отображает результаты в выбранном формате.

### 3.2. Расширенный анализ графа

Пользователь применяет фильтры для выявления ключевых навыков/технологий.

Использует фильтры для изоляции подграфов (например, только backend-профессии).

#### 4. Экспорт данных

##### 4.1. Экспорт в машиночитаемом формате

Основной сценарий:

Пользователь нажимает «Экспорт» в верхнем меню.

Выбирает формат: CSV.

Указывает название файла и параметры.

Нажимает «Скачать».

Система генерирует файл и сохраняет его на устройство пользователя.

### 2.3. Вывод

В системе преобладают операции чтения данных, так как:

Пользователи чаще взаимодействуют с данными через просмотр, фильтрацию, визуализацию и анализ.

Импорт (операции записи) выполняются значительно реже в основном при первичной настройке системы или редких обновлениях.

### 3. МОДЕЛЬ ДАННЫХ

#### 3.1. Нереляционная модель

Модель данных будет представлена в виде узлов (Nodes) и связей (Relationships), что позволит эффективно хранить и обрабатывать взаимосвязи между профессиями, категориями, навыками, технологиями и инструментами. Модель представлена как в виде графической диаграммы (ER-диаграмма), так и в формате JSON-схемы, что позволяет наглядно отобразить сущности, связи, типы данных и коллекции.

##### 3.1.А. Графическое представление

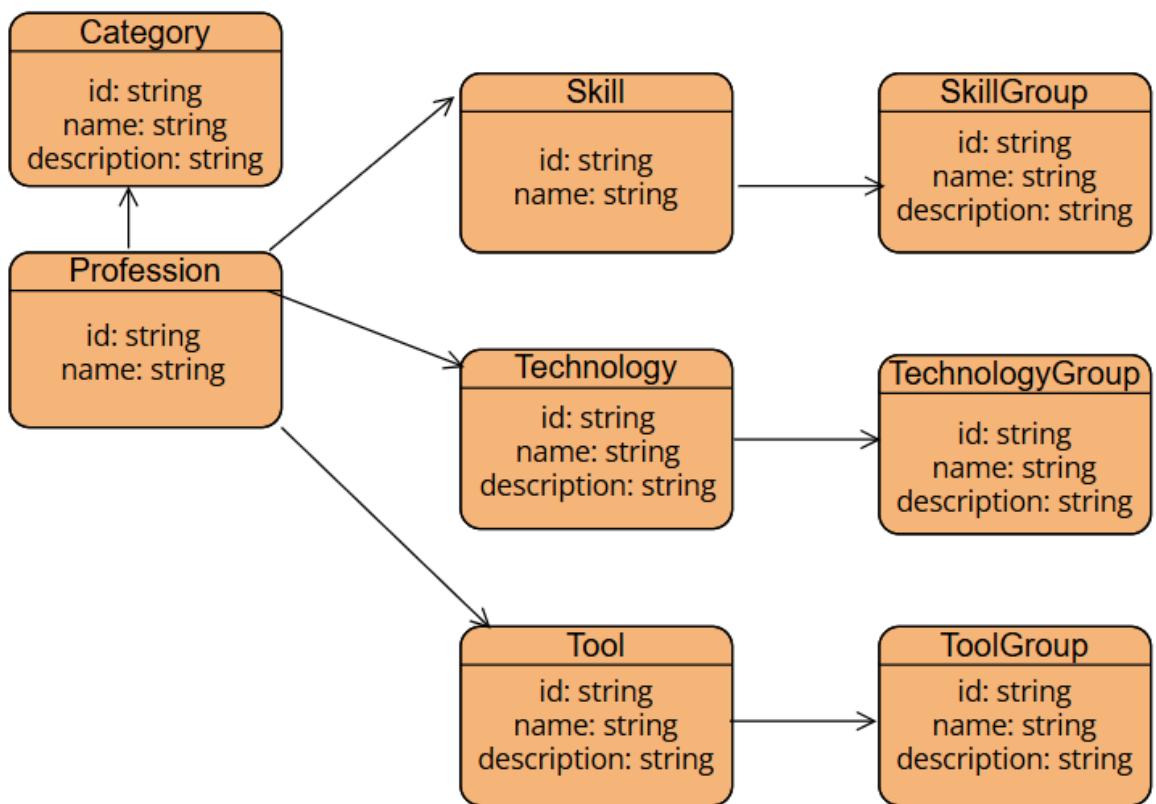


Рисунок 2 – Графическое представление нереляционной модели

##### 3.1.Б. Описание назначений коллекций, типов данных и сущностей

Сущности и их свойства:

- Profession (Профессия). Свойства: id: UUID, name: string
- Category (Категория). Свойства: id: UUID, name: string, description: string
- Skill (Навык). Свойства: id: UUID, name: string
- Technology (Технология). Свойства: id: UUID, name: string, description: string
- Tool (Инструмент). Свойства: id: UUID, name: string, description: string
- TechnologyGroup (Группа технологий). Свойства: id: UUID, name: string, description: string
- ToolGroup (Группа инструментов). Свойства: id: UUID, name: string, description: string
- SkillGroup (Группа навыков). Свойства: id: UUID, name: string, description: string

Таблица 1 – Связи в нереляционной модели

Связь	Описание	От кого → кому
BELONGS_TO	Профессия относится к категории	(:Profession) → (:Category)
REQUIRES	Профессия требует навык	(:Profession) → (:Skill)
USES_TECH	Профессия использует технологию	(:Profession) → (:Technology)
USES_TOOL	Профессия использует инструмент	(:Profession) → (:Tool)
GROUPS_TECH	Технология относится к определенной группе	(:Technology) → (:TechnologyGroup)
GROUPS_TOOL	Инструмент относится к определенной группе	(:Tool) → (:ToolGroup)
GROUPS_SKILL	Навык относится к определенной группе	(:Skill) → (:SkillGroup)

## **1. Коллекции и их назначения**

Коллекции представляют собой массивы объектов, группирующих данные по их назначению.

- **professions** (Профессии) – содержит перечень IT-профессий с категориями, навыками, технологиями и инструментами, необходимыми для работы.
- **categories** (Категории профессий) – определяет категории, к которым принадлежат профессии (например, "Аналитика", "Тестирование", "Разработка").
- **skills** (Навыки) – список ключевых навыков, необходимых для выполнения задач в различных профессиях.
- **technologies** (Технологии) – перечень языков программирования, языков разметки гипертекста, библиотек и фреймворков, сетевых протоколов и т.п., используемых в IT-профессиях.
- **tools** (Инструменты) – список сред разработки, инструментов для тестирования, инструментов дизайна, графики и анимации, программ для анализа данных и т.п., используемых в IT-профессиях.
- **skill\_groups** (Группы навыков) - объединяет навыки по их назначению (например, "Soft skills", "Hard skills").
- **technology\_groups** (Группы технологий) – объединяет технологии по смыслу (например, "Языки программирования", "Фреймворки").
- **tool\_groups** (Группы инструментов) – объединяет инструменты по их назначению (например, "Среды разработки", "Библиотеки и фреймворки для тестирования").

## **2. Типы данных**

Сущности используют следующие типы данных:

- **id: string** – уникальный идентификатор (UUID или идентификатор в виде строки).
- **name: string** – название сущности.
- **description: string** – текстовое описание.

- skills: array of strings – перечень навыков профессии.
- technologies: array of strings – список технологий, связанных с профессией.
- tools: array of strings – список инструментов, которые используются в профессии.

### **3. Сущности и их назначения:**

- Profession (Профессия)

Описывает конкретную ИТ-профессию, содержит её название, категорию, необходимые навыки, технологии и инструменты.

Пример: Бизнес-аналитик, QA-инженер, Back-end разработчик.

- Category (Категория)

Классифицирует профессии по направлениям.

Пример: "Аналитика", "Тестирование", "Разработка".

- Skill (Навык)

Отдельный ключевой навык, который нужен для выполнения работы в конкретной профессии.

Пример: "Разработка API", "Моделирование данных", "Анализ требований".

- Technology (Технология)

Язык программирования, язык разметки гипертекста, библиотека или фреймворк, сетевой протокол и т.п.

Пример: SQL, Python, Java, Docker.

- Tool (Инструмент)

Среда разработки, инструмент для тестирования, инструмент дизайна, графики и анимации, программа для анализа данных и т.п.

Пример: Jira, Postman, Visual Studio Code, Tableau.

- SkillGroup (Группа навыков)

Объединяет навыки по их использованию.

Пример: Soft skills(Навык взаимодействия с заказчиками, Коммуникация с разработчиками по исправлению багов), Hard skills(Владение стандартами

автоматизации: ERP, CRM, MRP, Навык работы с языком программирования 1C).

- TechnologyGroup (Группа технологий)

Группирует технологии по их функциональному назначению.

Пример: "Языки программирования" (Python, Java), "Библиотеки и фреймворки для разработки" (Django, Spring).

- ToolGroup (Группа инструментов)

Объединяет инструменты по их использованию.

Пример: "IDE (Среды разработки)" (IntelliJ IDEA, PyCharm), "QA IDE (Инструменты для тестирования)" (Selenium, JMeter).

### **3.1.С. Оценка объема информации, хранимой в модели**

Размеры сущностей:

Каждый узел имеет идентификатор (id), имя (name), описание (description) и может содержать массивы связей (например, навыков, технологий и инструментов). Каждая связь также будет занимать место и иметь идентификаторы и направление (например, BELONGS\_TO, REQUIRES и т.д.).

#### **1. Исходные данные**

В модели используются узлы (Nodes) и связи (Relationships):

Общее количество объектов:

- Профессии (Profession): 10
- Категории (Category): 5
- Группы навыков (SkillGroup): 2
- Группы технологий (TechnologyGroup): 14
- Группы инструментов (ToolGroup): 11
- Навыки (Skill): 41
- Технологии (Technology): 55
- Инструменты (Tool): 76

Связи между объектами:

- BELONGS\_TO (Профессия → Категория): 10

- REQUIRES (Профессия → Навык):  $10 \times X_s$ , где  $X_s$  – среднее количество навыков на профессию;
- USES\_TECH (Профессия → Технология):  $10 \times X_t$ , где  $X_t$  – среднее количество технологий на профессию;
- USES\_TOOL (Профессия → Инструмент):  $10 \times X_i$ , где  $X_i$  – среднее количество инструментов на профессию;
- GROUPS\_TECH (Технология → Группа технологий):  $14 \times Y_t$ , где  $Y_t$  – среднее количество технологий в одной группе;
- GROUPS\_TOOL (Инструмент → Группа инструментов):  $11 \times Y_i$ , где  $Y_i$  – среднее количество инструментов в одной группе.
- GROUPS\_SKILL (Навык → Группа навыков):  $2 \times Y_m$ , где  $Y_m$  – среднее количество навыков в одной группе.

## 2. Оценка занимаемого объема памяти

Каждый объект (узел) хранит id (UUID), name (строка) и (если есть) description (строка). Примем средние размеры: UUID: 16 байт, name: 150 байт, description: 1000 байт (если есть).

### Размер узлов (Nodes):

- Profession:  $10 \times (16 + 150) = 1660$  байт
- Category:  $5 \times (16 + 150 + 1000) = 5830$  байт
- SkillGroup:  $2 \times (16 + 150 + 1000) = 2332$  байт
- TechnologyGroup:  $14 \times (16 + 150 + 1000) = 16324$  байт
- ToolGroup:  $11 \times (16 + 150 + 1000) = 12826$  байт
- Skill:  $41 \times (16 + 150) = 6806$  байт
- Technology:  $55 \times (16 + 150 + 1000) = 64130$  байт
- Tool:  $76 \times (16 + 150 + 1000) = 88616$  байт

### Размер связей (Relationships):

Связь хранит идентификаторы двух узлов ( $2 \times 16$  байт) и тип связи (20 байт). Примем оценки:

- BELONGS\_TO:  $10 \times 52 = 520$  байт

- REQUIRES:  $10 \times X_s \times 52$  байт
- USES\_TECH:  $10 \times X_t \times 52$  байт
- USES\_TOOL:  $10 \times X_i \times 52$  байт
- GROUPS\_TECH:  $14 \times Y_t \times 52$  байт
- GROUPS\_TOOL:  $11 \times Y_i \times 52$  байт
- GROUPS\_SKILL:  $2 \times Y_m \times 52$  байт

### **3. Формула зависимости объема от количества профессий Р**

Пусть:

- $X_s = 15$  (среднее количество навыков на профессию),
- $X_t = 25$  (среднее количество технологий на профессию),
- $X_i = 30$  (среднее количество инструментов на профессию),
- $Y_m = 25$  (среднее количество навыков в группе),
- $Y_t = 8$  (среднее количество технологий в группе),
- $Y_i = 15$  (среднее количество инструментов в группе).

Тогда общая оценка памяти:

$$V(P) = P \times 166 + 5 \times 1166 + 2 \times 1166 + 14 \times 1166 + 11 \times 1166 + 41 \times 166 + 55 \times 1166 + 76 \times 1166 + P \times 52 + P \times (15 \times 52) + P \times (25 \times 52) + P \times (30 \times 52) + 2 \times (25 \times 52) + 14 \times (8 \times 52) + 11 \times (15 \times 52)$$

$$V(P) = 3858 \times P + 213868 \text{ байт}$$

$$V(10) = 252\,448 \text{ байт} = 246,53 \text{ Кбайт}$$

Формула показывает, что объем хранения данных линейно зависит от количества профессий Р, и основное влияние оказывают связи. Если увеличивается количество профессий, растет объем хранимых данных, особенно за счет связей с навыками, технологиями и инструментами.

#### **3.1.D. Избыточность модели**

**Шаги расчета:**

1. Чистый объем данных ( $V_{Ч}$ )
2. Фактический объем данных ( $V$ )

##### **1. Чистый объем данных ( $V_{Ч}$ )**

Чистый объем данных — это объем без учета структуры хранения, связей и метаинформации. Мы учитываем только содержимое полей, которые несут полезную информацию. Рассмотрим основные элементы модели:

- Profession: name = 150 байт  $\rightarrow 10 \times 150 = 1500$  байт
- Category: name + description = 150 + 1000  $\rightarrow 5 \times 1150 = 5750$  байт
- Skill: name = 150 байт  $\rightarrow 41 \times 150 = 6150$  байт
- Technology: name + description = 150 + 1000 = 1150 байт  $\rightarrow 55 \times 1150 = 63250$  байт
- Tool: name + description = 150 + 1000 = 1150 байт  $\rightarrow 76 \times 1150 = 87400$  байт
- SkillGroup: name + description = 150 + 1000  $\rightarrow 2 \times 1150 = 2300$  байт
- TechnologyGroup: name + description = 150 + 1000  $\rightarrow 14 \times 1150 = 16100$  байт
- ToolGroup: name + description = 150 + 1000  $\rightarrow 11 \times 1150 = 12650$  байт

Чистый объем данных можно выразить через переменную P (количество профессий):

$$V_{\text{ч}}(P) = 150 \times P + 5750 + 6150 + 63250 + 87400 + 2300 + 16100 + 12650 = 150 \times P + 193600 \text{ байт}$$

## 2. Фактический объем данных (V)

Фактический объем данных (V) с учетом всех метаданных и структуры хранения, как рассчитано выше, выражается следующей формулой:

$$V(P) = 3858 \times P + 213868 \text{ байт}$$

## 3. Формула избыточности (E)

Избыточность данных — это отношение фактического объема к чистому объему. Для расчета избыточности используем следующую формулу:

$$E(P) = \frac{V(P)}{V_{\text{ч}}(P)} = \frac{3858 \times P + 213868}{150 \times P + 193600}$$

Для P = 10 избыточность E(P) составляет примерно 1.29. Таким образом, фактический объем данных на 29% больше, чем чистый объем данных, с учетом всех метаданных и структуры хранения.

### **3.1.Е. Направление роста модели**

Была получена следующая формула, выражающая зависимость  $V$  от количества каждой сущности:

$$V(P,C,S,T,I,TG,IG) = 3868P + 1166C + 166S + 1166T + 1166I + 1166SG + 1166TG + 1166IG$$

где:

- $P$  - количество профессий;
- $C$  - количество категорий;
- $S$  - количество навыков;
- $T$  - количество технологий;
- $I$  - количество инструментов;
- $SG$  - количество групп навыков;
- $TG$  - количество групп технологий;
- $IG$  - количество групп инструментов.

Наибольшее влияние на рост объема данных оказывает увеличение количества профессий ( $P$ ). Это обусловлено тем, что каждая профессия связана с большим количеством навыков, технологий и инструментов.

Вторым по значимости фактором является увеличение количества технологий ( $T$ ) и инструментов ( $I$ ), а также категорий ( $C$ ), групп технологий ( $TG$ ) и групп инструментов ( $IG$ ). Это связано с тем, что данные сущности содержат описания, занимающие значительный объем памяти.

Наименьшее влияние на рост объема данных оказывает увеличение количества навыков ( $S$ ).

### 3.1.F. Примеры данных

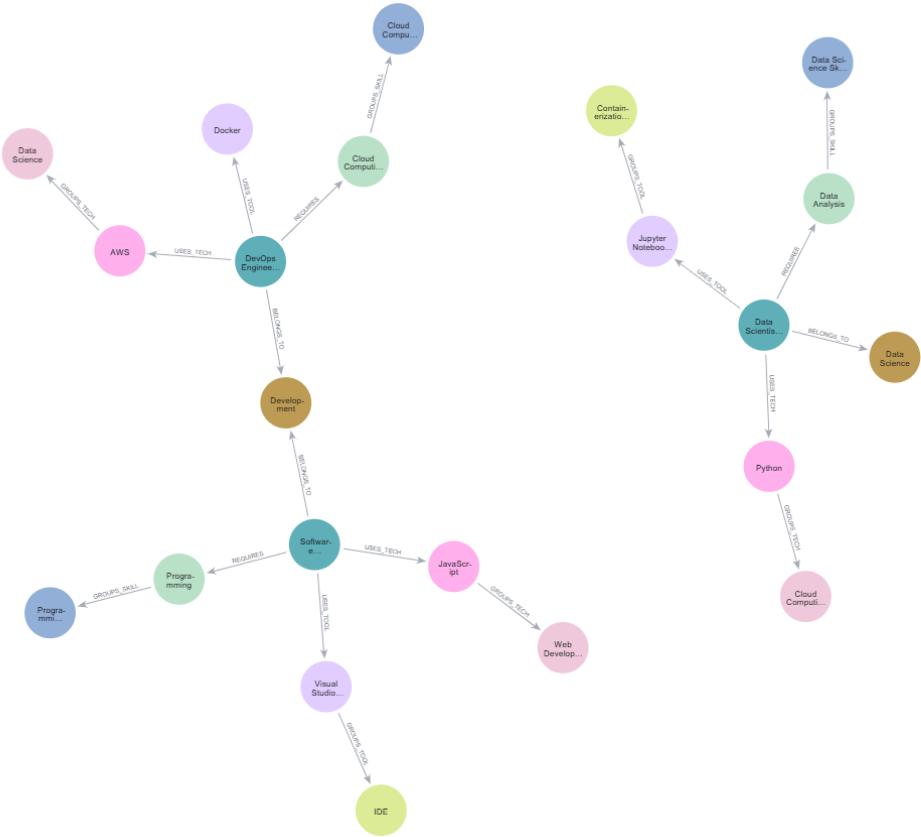


Рисунок 3 – Примеры данных для нереляционной модели

### 3.1.G. Примеры запросов

Для базы данных Neo4j используется язык запросов Cypher. Приведем несколько примеров запросов к нашей бд на нем:

1. Получить профессии с категориями:

```
MATCH (p:Profession) - [:BELONGS_TO] -> (c:Category)
RETURN p.name, c.name
```

2. Получить навыки для профессии "Бизнес-аналитик":

```
MATCH (p:Profession {name: "Бизнес-аналитик"}) - [:REQUIRES] -> (s:Skill)
RETURN s.name
```

3. Получить технологии и их группы:

```
MATCH (t:Technology) - [:GROUPS_TECH] -> (tg:TechnologyGroup)
RETURN t.name, tg.name
```

4. Получить инструменты, используемые профессией "QA-инженер":

```
MATCH (p:Profession {name: "QA-инженер"}) - [:USES_TOOL] -> (tool:Tool)
```

```
RETURN tool.name, tool.description
```

5. Получить профессии, использующие технологию "SQL":

```
MATCH (p:Profession) - [:USES_TECH] -> (t:Technology {name: "SQL"})
RETURN p.name
```

6. Поиск n навыков, которые встречаются в наибольшем количестве профессий:

```
MATCH (s:Skill) <- [:REQUIRES] - (p:Profession)
WITH s, COUNT(p) AS profession_count
ORDER BY profession_count DESC
LIMIT $n
RETURN s.name, profession_count
```

## 3.2. Реляционная модель

### 3.2.А. Графическое представление

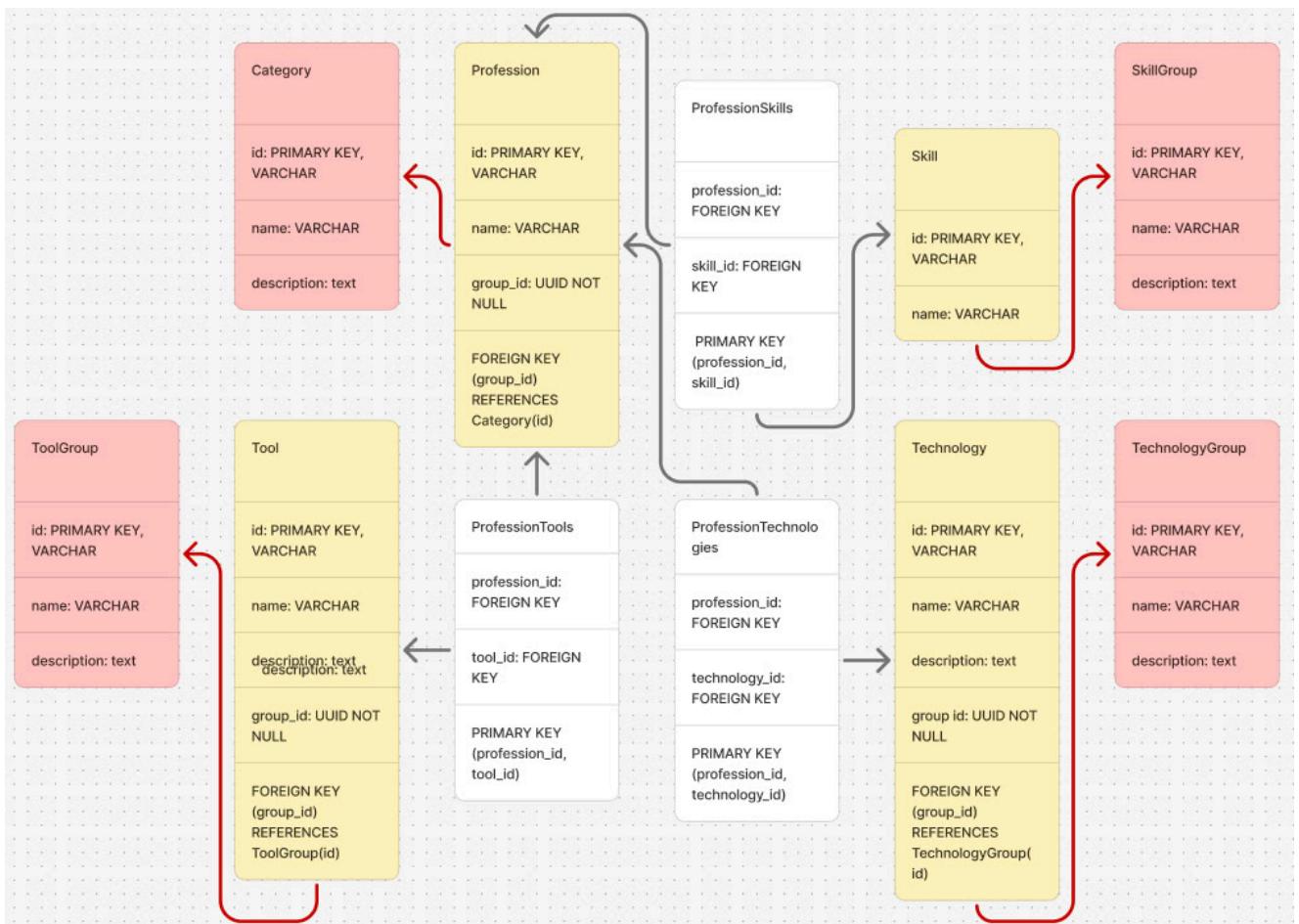


Рисунок 4 – графическое представление реляционной модели

### **3.2.В. Назначение коллекций, типы данных и сущностей**

#### **Основные сущности**

##### Profession

Назначение: содержит профессии (например, "Web-программист", "VR-разработчик").

Типы данных:

- id (VARCHAR) – уникальный идентификатор профессии;
- name (VARCHAR) – название профессии;
- group\_id (VARCHAR) – ссылка на категорию профессий (Category.id).

##### Category

Назначение: хранит категории (группы) профессий (например, "Тестирование", "Разработка, проектирование").

Типы данных:

- id (VARCHAR) – уникальный идентификатор категории (например, UUID);
- name (VARCHAR) – название категории (до 255 символов);
- description (TEXT) – описание.

##### Skill

Назначение: описывает навыки (например, "Навык взаимодействия с заказчиками", "Умение пользоваться базами данных").

Типы данных:

- id (VARCHAR) – уникальный идентификатор навыка;
- name (VARCHAR) – название навыка;
- group\_id (VARCHAR) – ссылка на группу навыков (SkillGroup.id).

##### SkillGroup

Назначение: группирует навыки на две группы: "Soft skills", "Hard skills").

Типы данных:

- id (VARCHAR) – уникальный идентификатор группы;
- name (VARCHAR) – название группы;
- description (TEXT) – описание.

## Technology

Назначение: хранит технологии (например, "Python", "Docker").

Типы данных:

- id (VARCHAR) – уникальный идентификатор технологии;
- name (VARCHAR) – название технологии;
- description (TEXT) – описание (до 1000 символов);
- group\_id (VARCHAR) – ссылка на группу технологий (TechnologyGroup.id).

## TechnologyGroup

Назначение: группирует технологии по направлениям (например, "Языки программирования", "Библиотеки и фреймворки для разработки").

Типы данных:

- id (VARCHAR) – уникальный идентификатор группы;
- name (VARCHAR) – название группы;
- description (TEXT) – описание.

## Tool

Назначение: содержит инструменты (например, "GitHub", "Figma").

Типы данных:

- id (VARCHAR) – уникальный идентификатор инструмента;
- name (VARCHAR) – название инструмента;
- description (TEXT) – описание;
- group\_id (VARCHAR) – ссылка на группу инструментов (ToolGroup.id).

## ToolGroup

Назначение: группирует инструменты (например, "IDE (Среды разработки)", "Системы управления базами данных (СУБД)").

Типы данных:

- id (VARCHAR) – уникальный идентификатор группы;
- name (VARCHAR) – название группы;
- description (TEXT) – описание.

## **Промежуточные таблицы (связи "многие-ко-многим")**

### ProfessionSkills

Назначение: связывает профессии с требуемыми навыками.

Типы данных:

- profession\_id (VARCHAR) – ссылка на профессию;
- skill\_id (VARCHAR) – ссылка на навык.

### ProfessionTechnologies

Назначение: связывает профессии с требуемыми технологиями.

Типы данных:

- profession\_id (VARCHAR) – ссылка на профессию;
- technology\_id (VARCHAR) – ссылка на технологию.

### ProfessionTools

Назначение: связывает профессии с требуемыми инструментами.

Типы данных:

- profession\_id (VARCHAR) – ссылка на профессию;
- tool\_id (VARCHAR) – ссылка на инструмент.

## **3.2.С. Оценка объема информации, хранимой в модели**

### **1. Исходные данные**

Основные сущности:

- Profession: 10
- Category: 5
- Skill: 41
- SkillGroup: 2
- Technology: 55
- TechnologyGroup: 14
- Tool: 76
- ToolGroup: 11

Промежуточные таблицы (связи "многие-ко-многим"):

- ProfessionSkills (Профессия → Навык):  $10 \times X_s$ , где  $X_s$  – среднее количество навыков на профессию;
- ProfessionTechnologies (Профессия → Технология):  $10 \times X_t$ , где  $X_t$  – среднее количество технологий на профессию;
- ProfessionTools (Профессия → Инструмент):  $10 \times X_i$ , где  $X_i$  – среднее количество инструментов на профессию.

## 2. Оценка занимаемого объема памяти

Таблица 2 – Оценка занимаемого объема памяти

Объект (коллекция)	Поля	Средний вес каждого поля, байт	Общий вес объекта, байт
Profession	id (VARCHAR), name (VARCHAR), group_id (VARCHAR)	16, 150, 16	182
Category	id (VARCHAR), name (VARCHAR), description (TEXT)	16, 150, 1000	1166
Skill	id (VARCHAR), name (VARCHAR), group_id (VARCHAR)	16, 150, 16	182
Technology	id (VARCHAR), name (VARCHAR), description (TEXT), group_id (VARCHAR)	16, 150, 1000, 16	1182
TechnologyGroup	id (VARCHAR), name (VARCHAR), description (TEXT)	16, 150, 1000	1166
Tool	id (VARCHAR), name (VARCHAR), description (TEXT), group_id (VARCHAR)	16, 150, 1000, 16	1182
ToolGroup	id (VARCHAR), name (VARCHAR), description (TEXT)	16, 150, 1000	1166

ProfessionSkills	profession_id (VARCHAR), skill_id (VARCHAR)	16, 16	32
ProfessionTechnologies	profession_id (VARCHAR), technology_id (VARCHAR)	16, 16	32
ProfessionTools	profession_id (VARCHAR), tool_id (VARCHAR)	16, 16	32

### 3. Формула зависимости объема от количества профессий Р

Пусть:

- $X_s = 15$  (среднее количество навыков на профессию),
- $X_t = 25$  (среднее количество технологий на профессию),
- $X_i = 30$  (среднее количество инструментов на профессию).

Тогда общая оценка памяти:

$$\begin{aligned}
 V(P) = & 182 \times P (\text{Profession}) + \\
 & 1166 \times 5 (\text{Category}) + \\
 & 182 \times 41 (\text{Skill}) + \\
 & 1166 \times 2 (\text{SkillGroup}) + \\
 & 1182 \times 55 (\text{Technology}) + \\
 & 1166 \times 14 (\text{TechnologyGroup}) + \\
 & 1182 \times 76 (\text{Tool}) + \\
 & 1166 \times 11 (\text{ToolGroup}) + \\
 & 32 \times 15 \times P (\text{ProfessionSkills}) + \\
 & 32 \times 25 \times P (\text{ProfessionTechnologies}) + \\
 & 32 \times 30 \times P (\text{ProfessionTools}) \\
 = & 2422 \times P + 199\,616 \text{ (байт)}
 \end{aligned}$$

$$V(P) = 2422 \times P + 199\,616 \text{ байт}$$

$$V(10) = 223\,836 \text{ байт} = 218,59 \text{ Кбайт}$$

Формула показывает, что объем хранения данных линейно зависит от количества профессий Р, и основное влияние оказывают связи. Если увеличивается количество профессий, растет объем хранимых данных, особенно за счет связей с навыками, технологиями и инструментами.

## D. Избыточность модели

### Шаги расчета:

1. Чистый объем данных ( $V_{\text{Ч}}$ )
2. Фактический объем данных ( $V$ )

#### 1. Чистый объем данных ( $V_{\text{Ч}}$ )

Чистый объем данных — это объем без учета структуры хранения, связей и метаинформации. Мы учитываем только содержимое полей, которые несут полезную информацию. Рассмотрим основные элементы модели:

- Profession: name = 150 байт
- Category: name + description = 150 + 1000 = 1150 байт
- Skill: name = 150 байт
- SkillGroup: name + description = 150 + 1000 = 1150 байт
- Technology: name + description = 150 + 1000 = 1150 байт
- TechnologyGroup: name + description = 150 + 1000 = 1150 байт
- Tool: name + description = 150 + 1000 = 1150 байт
- ToolGroup: name + description = 150 + 1000 = 1150 байт

Чистый объем данных можно выразить через переменную  $P$  (количество профессий):

Тогда общая оценка памяти:

$$\begin{aligned}V_{\text{Ч}}(P) &= 150 \times P (\text{Profession}) + \\&1150 \times 5 (\text{Category}) + \\&150 \times 41 (\text{Skill}) + \\&1150 \times 2 (\text{SkillGroup}) + \\&1150 \times 55 (\text{Technology}) + \\&1150 \times 14 (\text{TechnologyGroup}) + \\&1150 \times 76 (\text{Tool}) + \\&1150 \times 11 (\text{ToolGroup}) \\&= 150 \times P + 193\,600 \text{ (байт)}\end{aligned}$$

$$V_{\text{Ч}}(P) = 150 \times P + 193\,600 \text{ байт}$$

$$V_{\text{Ч}}(10) = 195\,100 \text{ байт} = 190,53 \text{ Кбайт}$$

## **2. Фактический объем данных (V)**

Фактический объем данных (V) с учетом всех метаданных и структуры хранения, как рассчитано выше, выражается следующей формулой:

$$V(P) = 2422 \times P + 199616 \text{ байт}$$

## **3. Формула избыточности (E)**

Избыточность данных — это отношение фактического объема к чистому объему. Для расчета избыточности используем следующую формулу:

$$E(P) = \frac{V(P)}{V_{\text{ч}}(P)} = \frac{2422 \times P + 199616}{150 \times P + 193600}$$

Для  $P = 10$  избыточность  $E(P)$  составляет примерно 1,15. Таким образом, фактический объем данных на 15% больше, чем чистый объем данных, с учетом всех метаданных и структуры хранения.

### **3.2.Е. Направление роста модели**

Из [оценки занимаемого объема памяти](#) и расчёта, что в среднем

- ProfessionSkills: 15 связи/профессию,
- ProfessionTechnologies: 25 связи/профессию,
- ProfessionTool: 30 связи/профессию,

$$V(P, C, S, T, I, TG, IG) = (182 + 15 \times 32 + 25 \times 32 + 30 \times 32)P + 1166C + (182 + 32)S + (182 + 32)T + (182 + 32)I + 1166SG + 1166TG + 1166IG \text{ байт}$$

была получена следующая формула, выражающая зависимость V от количества каждой сущности:

$$V(P,C,S,T,I,TG,IG) = 2422P + 1182C + 214S + 1214T + 1214I + 1166SG + 1166TG + 1166IG \text{ байт}$$

где:

- $P$  - количество профессий;
- $C$  - количество категорий;
- $S$  - количество навыков;
- $T$  - количество технологий;
- $I$  - количество инструментов;

- $SG$  - количество групп навыков;
- $TG$  - количество групп технологий;
- $IG$  - количество групп инструментов.

Наибольшее влияние на рост объема данных оказывает увеличение количества профессий (P). Это обусловлено тем, что каждая профессия связана с большим количеством навыков, технологий и инструментов.

Вторым по значимости фактором является увеличение количества технологий (T) и инструментов (I). Это связано с тем, что данные сущности содержат описания, занимающие значительный объем памяти.

Увеличение количества категорий (C), групп навыков (SG), групп технологий (TG) и групп инструментов (IG) также приводит к росту объема данных, но в меньшей степени.

Наименьшее влияние на рост объема данных оказывает увеличение количества навыков (S).

### **3.2. F. Примеры данных**

#### **1. Таблица для профессий (Profession)**

```
CREATE TABLE Profession (
    id UUID PRIMARY KEY,
    name VARCHAR(50) NOT NULL
    group_id UUID NOT NULL, -- Внешний ключ на Category
    FOREIGN KEY (group_id) REFERENCES Category(id)
);
```

#### **2. Таблица для групп профессий - категорий (Category)**

```
CREATE TABLE Category (
    id UUID PRIMARY KEY,
    name VARCHAR(50) NOT NULL
);
```

#### **3. Таблица для навыков (Skill)**

```
CREATE TABLE Skill (
    id UUID PRIMARY KEY,
    name VARCHAR(50) NOT NULL
);
```

#### 4. Промежуточная таблица для связи профессий и навыков (*ProfessionSkills*)

```
CREATE TABLE ProfessionSkills (
    profession_id UUID REFERENCES Profession(id),
    skill_id UUID REFERENCES Skill(id),
    PRIMARY KEY (profession_id, skill_id)
);
```

#### 5. Таблица для технологий (*Technology*)

```
CREATE TABLE Technology (
    id UUID PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    group_id UUID NOT NULL, -- Внешний ключ на TechnologyGroup
    FOREIGN KEY (group_id) REFERENCES TechnologyGroup(id)
);
```

#### 6. Таблица для групп технологий (*TechnologyGroup*)

```
CREATE TABLE TechnologyGroup (
    id UUID PRIMARY KEY,
    name VARCHAR(50) NOT NULL
);
```

#### 7. Промежуточная таблица для связи профессий и технологий (*ProfessionTechnologies*)

```
CREATE TABLE ProfessionTechnologies (
    profession_id UUID REFERENCES Profession(id),
    technology_id UUID REFERENCES Technology(id),
    PRIMARY KEY (profession_id, technology_id)
);
```

#### 8. Таблица для инструментов (*Tool*)

```
CREATE TABLE Tool (
    id UUID PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    group_id UUID NOT NULL, -- Внешний ключ на ToolGroup
    FOREIGN KEY (group_id) REFERENCES ToolGroup(id)
);
```

#### 9. Таблица для групп инструментов (*ToolGroup*)

```
CREATE TABLE ToolGroup (
    id UUID PRIMARY KEY,
```

```
    name VARCHAR(50) NOT NULL  
);
```

## 10. Промежуточная таблица для связи профессий и инструментов (ProfessionTools)

```
CREATE TABLE ProfessionTools (  
    profession_id UUID REFERENCES Profession(id),  
    tool_id UUID REFERENCES Tool(id),  
    PRIMARY KEY (profession_id, tool_id)  
);
```

### 3.2.G. Примеры запросов

#### 1. Получение всех профессий с их категориями

Текст запроса:

```
SELECT p.id, p.name, c.name AS category_name  
FROM Profession p  
JOIN Category c ON p.group_id = c.id;
```

Количество запросов: 1

Количество задействованных коллекций: 2 (Profession, Category)

#### 2. Получение всех навыков для конкретной профессии

Текст запроса:

```
SELECT s.id, s.name  
FROM Skill s  
JOIN ProfessionsSkills ps ON s.id = ps.skill_id  
WHERE ps.profession_id = 'SPECIFIC_UUID'; -- Заменить на нужный ID  
профессии
```

Количество запросов: 1

Количество задействованных коллекций: 2 (Skill, ProfessionsSkills)

#### 3. Получение всех технологий для конкретной профессии

Текст запроса:

```
SELECT t.id, t.name  
FROM Technology t  
JOIN ProfessionTechnologies pt ON t.id = pt.technology_id  
WHERE pt.profession_id = 'SPECIFIC_UUID'; -- Заменить на нужный ID  
профессии
```

Количество запросов: 1  
Количество задействованных коллекций: 2 (Technology, ProfessionTechnologies)

#### 4. Получение всех инструментов для конкретной профессии

Текст запроса:

```
SELECT to.id, to.name
FROM Tool to
JOIN ProfessionTools pt ON to.id = pt.tool_id
WHERE pt.profession_id = 'SPECIFIC_UUID'; -- Заменить на нужный ID
профессии
```

Количество запросов: 1  
Количество задействованных коллекций: 2 (Tool, ProfessionTools)  
5. Получение всех профессий, которые используют определённую технологию

Текст запроса:

```
SELECT p.id, p.name
FROM Profession p
JOIN ProfessionTechnologies pt ON p.id = pt.profession_id <br>
WHERE pt.technology_id = 'SPECIFIC_UUID'; -- Заменить на нужный ID
технологии
```

Количество запросов: 1  
Количество задействованных коллекций: 2 (Profession, ProfessionTechnologies)

#### 6. Поиск n навыков, которые встречаются в наибольшем количестве профессий

Текст запроса:

```
SELECT s.name, COUNT(ps.profession_id) AS profession_count
FROM Skill s
JOIN ProfessionSkills ps ON s.id = ps.skill_id
GROUP BY s.id, s.name
ORDER BY profession_count DESC
LIMIT n;
```

Количество запросов: 1

Количество задействованных коллекций: 2 (Profession, ProfessionSkills)

### Общая информация

- Общее количество запросов: 6 (для выполнения всех представленных юзкейсов)
- Общее количество задействованных коллекций: 6 (Profession, Category, Skill, ProfessionSkills, Technology, ProfessionTechnologies, Tool, ProfessionTools)

### 3.3. Сравнение моделей

#### 3.3.А. Удельный объём информации

Таблица 3 – Удельный объём информации

Параметр	NoSQL	SQL
Формула объема(байт)	$V_1(P) = 3858 \times P + 213868$	$V_2(P) = 2422 \times P + 199616$
Объем при P=10	246,53 КБ	218,59 КБ
Зависимость от P	Быстрый рост (множитель 3858)	Медленный рост (множитель 2422)
Избыточность	1,29 (29% метаданных)	1,15 (15% метаданных)

SQL экономичнее по объему при любом P, включая большие данные. В то время как NoSQL имеет более высокую избыточность и больший объем, из-за явного хранения связей.

#### 3.3.В. Запросы по отдельным юзкейсам

Таблица 4 – Запросы по отдельным юзкейсам

Параметр	NoSQL	SQL
Пример запроса	MATCH (p:Profession)-[:REQUIRES]->(s:Skill) RETURN s.name	SELECT s.name FROM Skill s JOIN ProfessionSkills ps ON ...
Количество запросов	1 на юзкейс (связи обрабатываются напрямую)	1 на юзкейс (требует JOIN таблиц)
Сложность запросов	Низкая (оптимизирована для связей)	Высокая (JOIN для связей "многие-ко-многим")

NoSQL проще для выполнения связанных запросов. А SQL требует более сложных операций JOIN, что замедляет выполнение.

### **3.3.C. Вывод**

SQL лучше тем что занимает меньше памяти, но из-за JOIN медленнее и сложнее. Однако NoSQL лучше для задач, где важна скорость обработки сложных связей и легкость масштабируемости, но из-за хранения типов связей занимает больше места.

## 4. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

### 4.1. Краткое описание

Разработанное приложение представляет собой веб-систему, развертываемую с помощью Docker и состоящую из трёх основных контейнеров: базы данных, серверной (backend) и клиентской (frontend) частей.

Контейнер базы данных обеспечивает хранение всей информации каталога. Используется нереляционная графовая СУБД Neo4j. При запуске контейнера происходит инициализация структуры базы данных, что позволяет сразу приступить к работе с приложением.

Backend реализован на Python с использованием фреймворка FastAPI и запускается в контейнере на базе образа Python 3.12-slim. Все необходимые зависимости устанавливаются из файла requirements.txt, а исходный код приложения копируется в рабочую директорию контейнера. Запуск осуществляется через скрипт entrypoint.sh, который также отвечает за инициализацию пустой базы данных при первом старте. Backend предоставляет REST API для взаимодействия с данными каталога: поддерживаются операции добавления, редактирования, удаления и поиска информации. Для массового обновления или импорта данных реализована возможность приёма файлов в формате .json, что облегчает интеграцию с внешними источниками и автоматизацию работы.

Frontend — это одностраничное web-приложение, разработанное на React. Оно разворачивается в отдельном контейнере на базе Node.js 14. При запуске устанавливаются все зависимости из package.json, выполняется сборка проекта, после чего приложение стартует командой npm start. Клиентская часть взаимодействует с backend через API, отображая данные каталога в удобном и наглядном виде. Пользователь может просматривать, искать, фильтровать и редактировать записи, а также загружать новые данные.

Вся система полностью контейнеризирована, что обеспечивает простоту развертывания, масштабируемость и независимость от среды пользователя.

## **4.2. Использованные технологии**

БД: Neo4j:5.26.5

Back-end: Python:3.12-slim

Front-end: Node:14, HTML, CSS, JavaScript, React.

## **4.3. Снимки экрана приложения**

Экраны приложения и переходы между ними отображены на рисунках 5-8.



Рисунок 5. Схема экранов приложения.



Рисунок 6. Схема экранов приложения.



Рисунок 7. Схема экранов приложения.



Рисунок 8. Схема экранов приложения.

## Ссылка на приложение

Ссылка на github-репозиторий: <https://github.com/moevm/nosql11h25-itcatalog>

## **ЗАКЛЮЧЕНИЕ**

### **5.1. Достигнутые результаты**

В результате выполнения курсовой работы была разработана и реализована структура ИТ-каталога, основанная на нереляционной модели данных. В процессе работы были созданы диаграммы моделей, подготовлены примеры данных. Для удобства развертывания и тестирования решения использовались современные инструменты контейнеризации, такие как Docker и docker-compose. Фронтенд-часть приложения обеспечивает удобный и интуитивно понятный пользовательский интерфейс для работы с ИТ-каталогом: пользователи могут просматривать, искать и фильтровать записи, а также добавлять и редактировать информацию в каталоге. Бэкенд реализует логику приложения, обрабатывает запросы от клиента, выполняет валидацию данных и обеспечивает безопасное взаимодействие с базой данных. Для хранения и обработки информации используется нереляционная СУБД Neo4j. Взаимодействие между фронтеном и бэкеном осуществляется через REST API. Приложение поддерживает основные CRUD-операции (создание, чтение, обновление, удаление данных), а также реализует функции поиска и фильтрации по различным параметрам.

### **5.2. Недостатки и пути для улучшения полученного решения**

В процессе выполнения проекта были выявлены определённые недостатки, которые могут быть устранены в дальнейшем. В частности, текущий пользовательский интерфейс реализован на базовом уровне, что ограничивает удобство работы с приложением. Для повышения комфорта пользователей и расширения функциональности целесообразно доработать интерфейс, добавить адаптивную верстку и улучшить визуальное оформление. Кроме того, стоит уделить внимание более тщательной обработке ошибок и валидации данных как на стороне сервера, так и на стороне клиента, что повысит надёжность работы приложения. Дополнительные возможности поиска

и фильтрации информации также позволяют сделать работу с каталогом более эффективной. Помимо этого, перспективным направлением развития является расширение базы данных за счёт добавления новых типов объектов и связей между ними. Увеличение объёма и разнообразия данных сделает каталог более информативным и полезным для различных сценариев использования.

### **5.3. Будущее развитие решения**

В будущем можно расширить функциональность приложения за счёт внедрения системы регистрации и авторизации пользователей, а также интеграции с внешними сервисами. Также можно доработать пользовательский интерфейс, добавить новые возможности поиска и фильтрации данных. Кроме того, расширить базу данных с помощью добавления новых типов объектов и связей, что сделает каталог более информативным и удобным для пользователей.

## ПРИЛОЖЕНИЯ

### **5.1. Документация по сборке и развертыванию приложения**

Для сборки и развертывания приложения используется система контейнеризации Docker и инструмент docker-compose, что обеспечивает простоту процесса запуска. В корне проекта расположен файл docker-compose.yml, в котором описаны необходимые сервисы: база данных и серверная часть приложения, а также клиентская. Для запуска приложения необходимо установить Docker и docker-compose, после чего выполнить команду: docker-compose up --build. После успешного запуска все необходимые сервисы будут автоматически подняты в отдельных контейнерах, а приложение станет доступно по указанному в конфигурации адресу http://localhost:3000. Все параметры подключения и переменные окружения настраиваются в файле .env.

### **5.2. Инструкция для пользователя**

#### **1) Установка необходимых программ**

Перед началом работы убедитесь, что на вашем компьютере установлены: Docker Desktop (для Windows), Git (для клонирования репозитория).

#### **2) Клонирование репозитория**

Откройте терминал (или командную строку) и выполните команду: git clone <https://github.com/moevm/nosql1h25-itcatalog.git>

Перейдите в папку проекта: cd nosql1h25-itcatalog-main

#### **3) Сборка и запуск приложения**

В терминале выполните команду: docker-compose up --build.

Дождитесь завершения сборки и запуска всех сервисов.

#### **4) Доступ к приложению**

После успешного запуска приложение будет доступно по адресу, указанному в настройках http://localhost:3000. Откройте этот адрес в браузере.

#### **5) Работа с приложением**

В пользовательском интерфейсе вы сможете:

- Просматривать и искать объекты каталога
- Добавлять, редактировать и удалять объекты
- Использовать фильтры и поиск по различным параметрам
- Осуществлять импорт и экспорт объектов

## 6) Остановка приложения

Для остановки работы приложения нажмите Ctrl+C в терминале, где был запущен Docker, либо выполните команду: docker-compose down. Если вы хотите полностью удалить все контейнеры, сети и связанные с ними тома (volumes), используйте команду: docker-compose down -v. Эта команда остановит все сервисы и удалит все созданные для проекта тома, что позволит начать работу с чистого состояния при следующем запуске.

Если приложение не запускается, проверьте, что все сервисы корректно описаны в docker-compose.yml и что все необходимые файлы присутствуют. Убедитесь, что порты, указанные в .env и docker-compose.yml, свободны и не заняты другими программами.

## **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Статья "NEO4J – графовые базы данных": <https://habr.com/ru/articles/650623/>
2. Документация к Neo4j: <https://neo4j.com/docs/>
3. Современный учебник JavaScript: <https://learn.javascript.ru/>
4. Руководство по Node.js: <https://metanit.com/web/nodejs/>
5. Документация и учебник React: <https://ru.react.js.org/docs/getting-started.html>