Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО Ульяновский государственный технический университет

Кафедра «Вычислительная техника»

Лабораторная работа №3

«Построение онтологической модели»

Вариант №17

Выполнила студентка

группы ИВТИИбд-11

Лой.Ю.К.

Проверил преподаватель

Хайрулин И.Д.

Ульяновск, 2025

1. **Структурное описание онтологической модели**

Онтологическая модель (онтология) — это формализованное представление знаний о предметной области. Она описывает понятия (концепты) и отношения между ними, что позволяет компьютеру и человеку использовать информацию эффективнее. В данном варианте реализации Система Отслеживания Ошибок (Bug Tracker), основные сущности (далее классы) модели включают:

* Программисты
* Проекты
* Ошибки

1. **Для реализации модели использованы два инструмента:**

1. Protégé — применяется для построения OWL-онтологии, добавления аннотаций и формулировки логических правил с использованием SWRL.

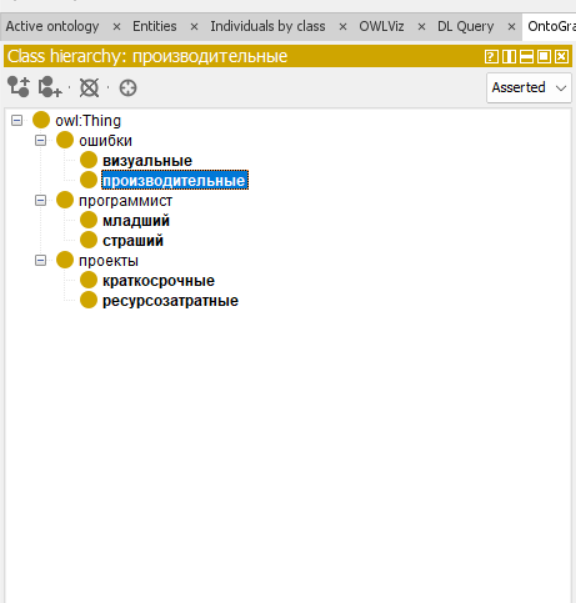
2. Neo4j — используется для создания графовой модели с узлами и связями, где логические зависимости реализуются с помощью языка запросов Cypher.

Онтологическая модель включает несколько типов элементов: классы (сущности), свойства данных и свойства объектов, а также логические правила (аксиомы), описывающие взаимосвязи между объектами.

1. **Особенности реализации в Protégé:**

Первым этапом создаются классы. Создаем несколько классов и подклассов:

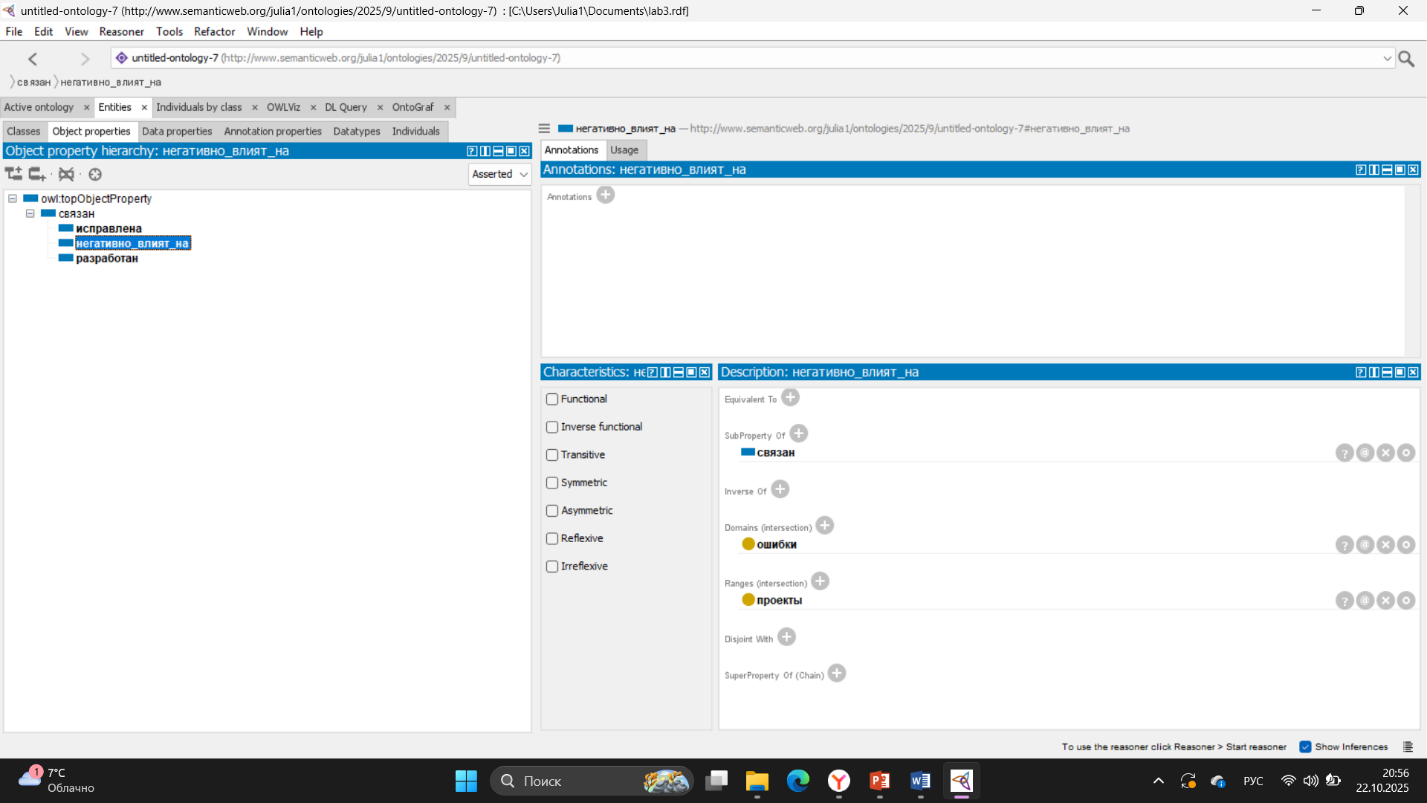
* Ошибки
* Программисты
* Проекты



Далее нужно создать для них свойства:

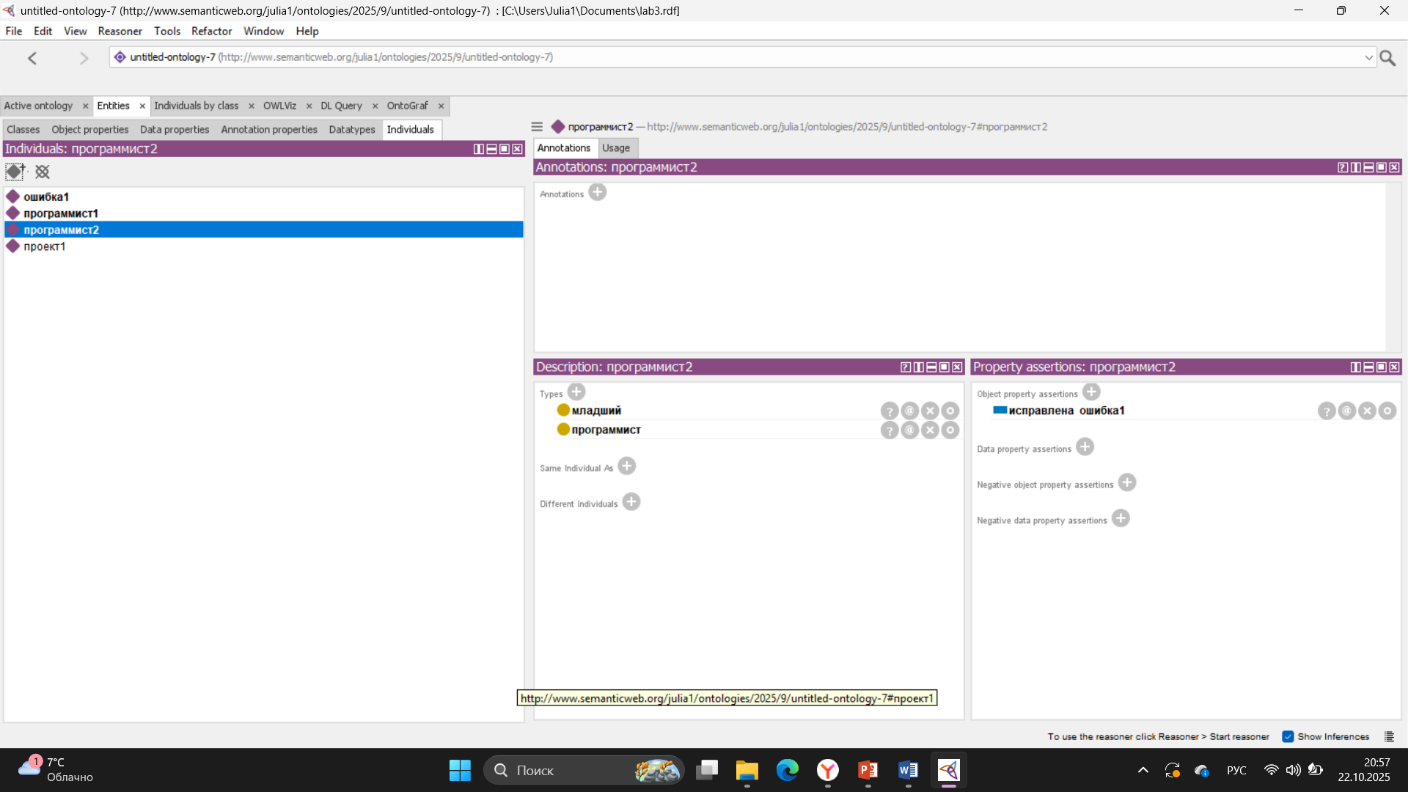
Здесь прописываются связи между классами:

* Ошибка исправлена программистом
* Проект разработан программистом
* Ошибка негативно влияет на проект

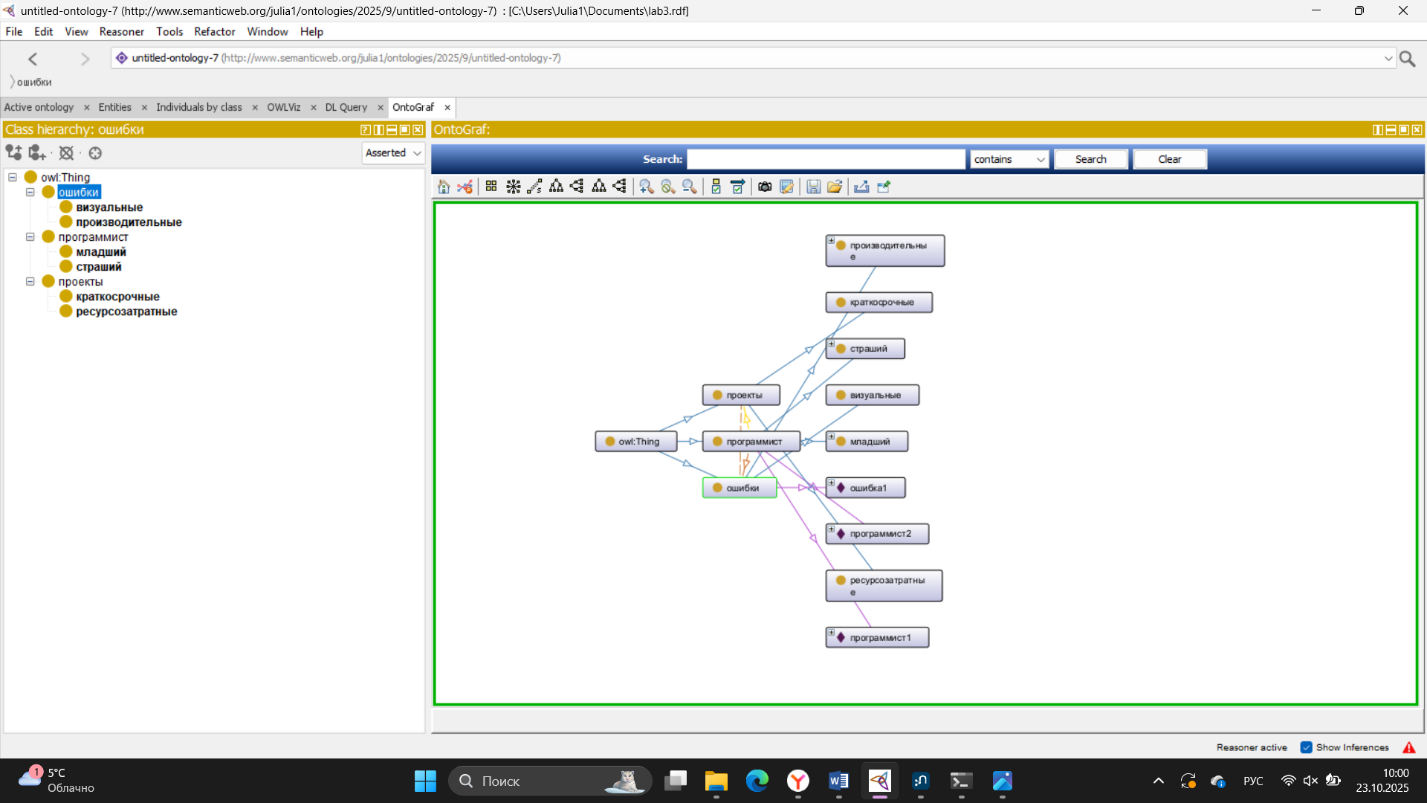


Создание конкретных экземпляров классов:

* Ошибка1 исправлена младшим программистом
* Программист2 исправлена ошибка1
* Программист один разработан проект1



В результате должно получиться дерево классов, которое реализует онтологическую модель, которая описывает различные типы программных проектов:



1. **Особенности реализации в** **Neo4j:**

Структура графа:

Узлы:

* Programmers: Иван, Елена, Алексей
* Projects: Проект А, Проект Б, Проект В
* Errors: Ошибки для каждого проекта (например, каждая ошибка имеет уникальное название).

Связи:

* WORKS\_ON: программисты работают над проектами
* HAS\_ERROR: проекты содержат ошибки

Создаем трех программистов с именами "Иван", "Елена" и "Алексей". Создаем три проекта ("Проект А", "Проект Б", "Проект В"). Определяем несколько ошибок для каждого проекта. Разделим работу следующим образом:

* Иван работает над всеми проектами.
* Елена работает над проектом А и проектом Б.
* Алексей работает над проектом Б и проектом В.

Присваиваем каждой ошибке соответствующий проект.

В итоге мы получаем:

* Три программиста: Иван, Елена, Алексей.
* Три проекта: Проект А, Проект Б, Проект В.
* Каждая ошибка связана с соответствующим ей проектом.
* Каждый программист работает над одним или несколькими проектами.

Создание узлов и связей (Cypher):

CREATE (:Programmer {name: 'Иван'}),

(:Programmer {name: 'Елена'}),

(:Programmer {name: 'Алексей'});

CREATE (:Project {title: 'Проект А', description: 'Первостепенный проект'}),

(:Project {title: 'Проект Б', description: 'Средне приоритетный проект'}),

(:Project {title: 'Проект В', description: 'Низкоприоритетный проект'});

CREATE (:Error {description: 'Ошибка #1 в проекте А'}),

(:Error {description: 'Ошибка #2 в проекте А'}),

(:Error {description: 'Ошибка #3 в проекте А'})

(:Error {description: 'Ошибка #1 в проекте Б'}),

(:Error {description: 'Ошибка #2 в проекте Б'}),

(:Error {description: 'Ошибка #3 в проекте Б'}),

(:Error {description: 'Ошибка #1 в проекте В'}),

(:Error {description: 'Ошибка #2 в проекте В'}),

(:Error {description: 'Ошибка #3 в проекте В'});

MATCH (ivan: Programmer {name: 'Иван'}), (projectA: Project {title: 'Проект А'}) MERGE (ivan)-[:WORKS\_ON]->(projectA);

MATCH (ivan: Programmer {name: 'Иван'}), (projectB: Project {title: 'Проект Б'}) MERGE (ivan)-[:WORKS\_ON]->(projectB);

MATCH (ivan: Programmer {name: 'Иван'}), (projectC: Project {title: 'Проект В'}) MERGE (ivan)-[:WORKS\_ON]->(projectC);

MATCH (elena: Programmer {name: 'Елена'}), (projectA: Project {title: 'Проект А'}) MERGE (elena)-[:WORKS\_ON]->(projectA);

MATCH (elena: Programmer {name: 'Елена'}), (projectB: Project {title: 'Проект Б'}) MERGE (elena)-[:WORKS\_ON]->(projectB);

MATCH (alexey: Programmer {name: 'Алексей'}), (projectB: Project {title: 'Проект Б'}) MERGE (alexey)-[:WORKS\_ON]->(projectB);

MATCH (alexey: Programmer {name: 'Алексей'}), (projectC: Project {title: 'Проект В'}) MERGE (alexey)-[:WORKS\_ON]->(projectC);

MATCH (projectA: Project {title: 'Проект А'}), (err1: Error {description: 'Ошибка #1 в проекте А'}) MERGE (projectA)-[:HAS\_ERROR]->(err1);

MATCH (projectA: Project {title: 'Проект А'}), (err2: Error {description: 'Ошибка #2 в проекте А'}) MERGE (projectA)-[:HAS\_ERROR]->(err2);

MATCH (projectA: Project {title: 'Проект А'}), (err3: Error {description: 'Ошибка #3 в проекте А'}) MERGE (projectA)-[:HAS\_ERROR]->(err3);

MATCH (projectB: Project {title: 'Проект Б'}), (err1: Error {description: 'Ошибка #1 в проекте Б'}) MERGE (projectB)-[:HAS\_ERROR]->(err1);

MATCH (projectB: Project {title: 'Проект Б'}), (err2: Error {description: 'Ошибка #2 в проекте Б'}) MERGE (projectB)-[:HAS\_ERROR]->(err2);

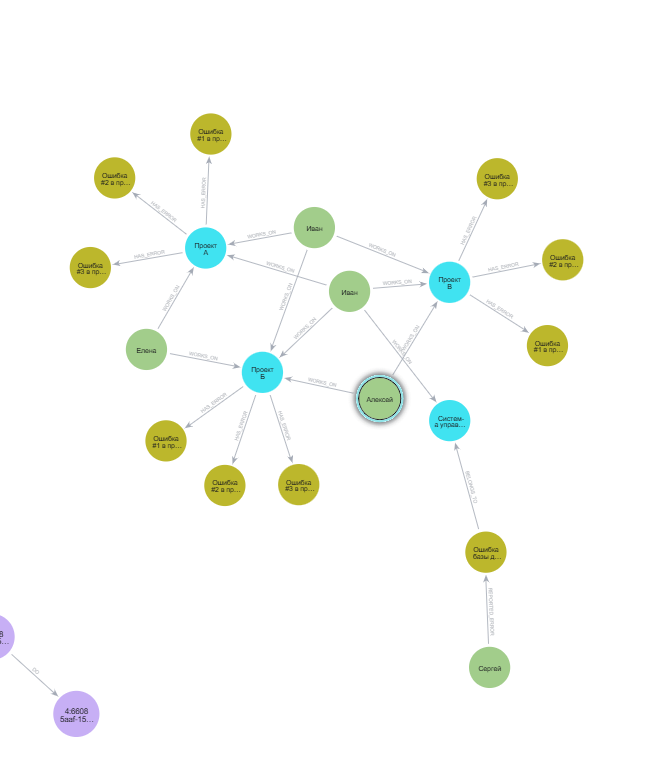
MATCH (projectB: Project {title: 'Проект Б'}), (err3: Error {description: 'Ошибка #3 в проекте Б'}) MERGE (projectB)-[:HAS\_ERROR]->(err3);

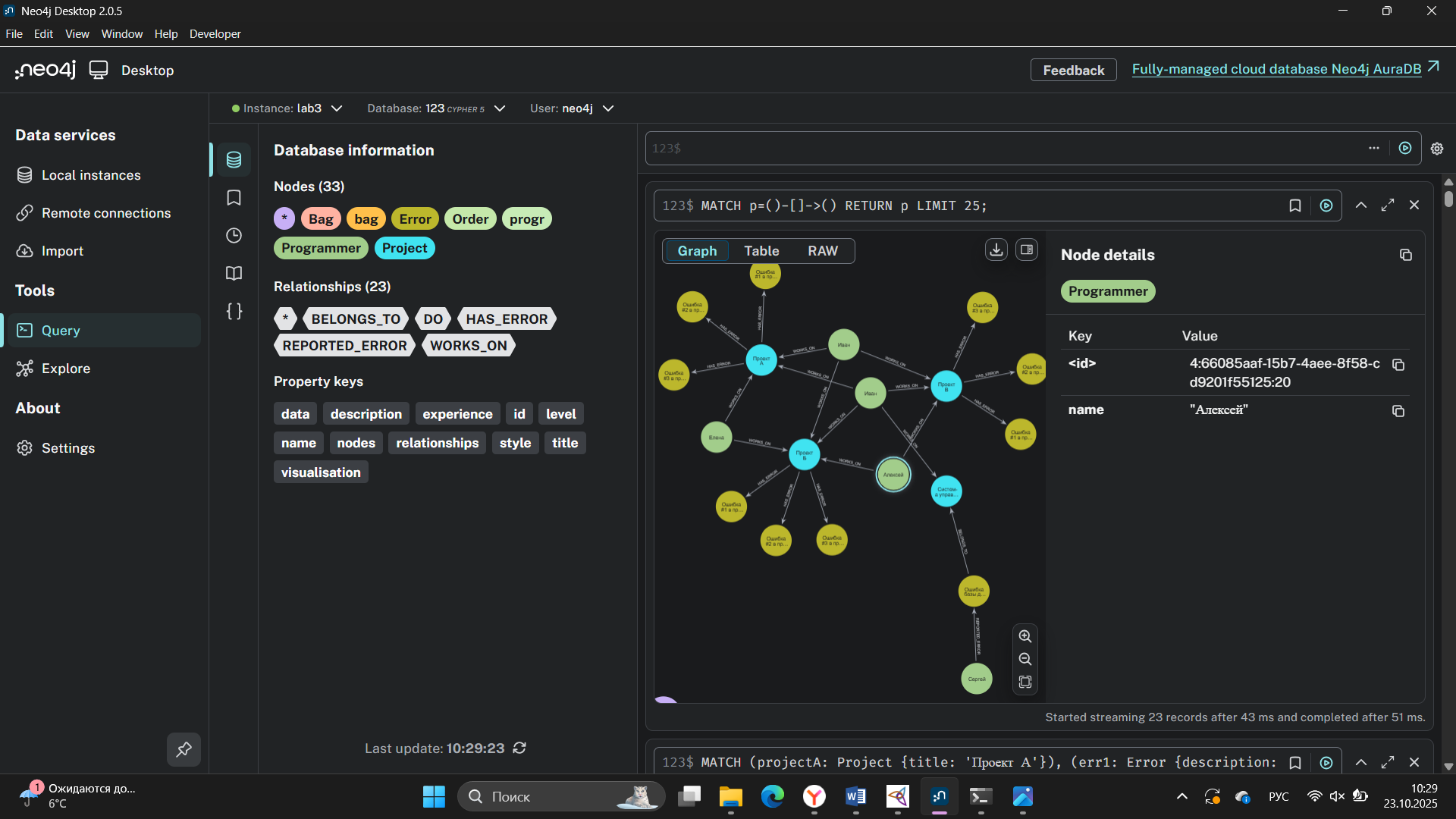
MATCH (projectC: Project {title: 'Проект В'}), (err1: Error {description: 'Ошибка #1 в проекте В'}) MERGE (projectC)-[:HAS\_ERROR]->(err1);

MATCH (projectC: Project {title: 'Проект В'}), (err2: Error {description: 'Ошибка #2 в проекте В'}) MERGE (projectC)-[:HAS\_ERROR]->(err2);

MATCH (projectC: Project {title: 'Проект В'}), (err3: Error {description: 'Ошибка #3 в проекте В'}) MERGE (projectC)-[:HAS\_ERROR]->(err3);

Получаем графовую модель такого вида:





1. **Вывод:**

Основные различия в построении этих:

Первая модель является онтологической, что означает, что она описывает структуру и взаимосвязи между различными типами сущностей (в данном случае проектов). Вторая модель является графовой, что означает, что она представляет данные в виде сети узлов и рёбер, где узлы могут иметь различные свойства, а рёбра могут представлять различные типы отношений. Первая модель использует иерархическую структуру классов для классификации проектов. Вторая модель использует сеть узлов и рёбер для представления взаимосвязанных сущностей.

Онтологическая модель лучше подходит для классификации и управления структурированными данными, в то время как графовая модель лучше подходит для анализа и визуализации сложных взаимосвязанных данных.