Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО Ульяновский государственный технический университет

Кафедра «Вычислительная техника»

Лабораторная работа №3

«Построение онтологической модели»

Вариант №4

Выполнил студент

группы ИВТИИбд-11

Шингарева Я. В.

Преподаватель

Хайрулин И.Д.

Ульяновск, 2025

**1. Структурное описание онтологической модели**

Онтологическая модель описывает систему музыкального сервиса и формализует ключевые объекты и их взаимосвязи. Основные сущности (далее классы) модели включают:

**- Исполнители (Artist):** музыканты и группы, создающие музыкальный контент

**- Альбомы (Album):** сборники треков, выпускаемые исполнителями

**- Треки (Track):** отдельные музыкальные композиции

**- Плейлисты (Playlist):** пользовательские и системные подборки треков

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.

**2. Перечень классов, отношений и аксиом**

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

**2.1 Классы (Classes)**

Модель содержит следующие основные классы:

**- Исполнители (Artist):** музыканты и группы, создающие музыкальный контент

**- Альбомы (Album):** сборники треков, выпускаемые исполнителями

**- Треки (Track):** отдельные музыкальные композиции

**- Плейлисты (Playlist):** пользовательские и системные подборки треков

Каждый класс является фундаментальной единицей модели и служит основой для определения свойств и отношений.

**2.2 Свойства данных (Data Properties)**

Свойства данных описывают характеристики отдельных объектов (индивидуумов) классов:

| **Свойство** | **Принадлежит классу** | **Описание / Комментарий** |
| --- | --- | --- |
| hasName | Artist, Album, Track, Playlist | Имя исполнителя, альбома, трека, плейлиста |
| hasDuration | Track | Продолжительность трека |
| hasReleaseDate | Album | Дата выпуска альбома |
| hasСreationDate | Playlist | Дата создания плейлиста |

Эти свойства позволяют хранить конкретные данные для каждого объекта и использовать их при поиске и логическом выводе.

**2.3 Свойства объектов (Object Properties)**

Свойства объектов определяют связи между различными объектами модели:

| **Свойство** | **Описание** |
| --- | --- |
| hasTrack | Сервис содержит трек |
| isPartOfAlbum | Является частью альбома |
| hasCreator | Создан пользователем |

Эти свойства обеспечивают **логическую структуру модели** и позволяют выводить новые связи между объектами.

**2.4 Аксиомы (SWRL-правила) для Protégé**

Аксиомы формализуют правила логического вывода в модели OWL:

1. **Связь исполнителя с треком через альбом:**

Artist(?a) ^ Track(?t) ^ Album(?alb) ^ hasCreator(?t, ?a) ^ isPartOfAlbum(?t, ?alb) -> hasCreator(?alb, ?a)

Если исполнитель создал трек, и этот трек является частью альбома, то исполнитель также является создателем альбома

1. **Автоматическая классификация плейлистов**

hasTrack(?x, ?y) -> Playlist(?x)

Если сущность содержит треки, то она является плейлистом

# 2.5 Аналог аксиом для Neo4j (Cypher-запросы)

Для графовой модели логические выводы реализуются с помощью запросов:

1. **Создание рекомендаций плейлистов:**

MATCH (u:User)-[:LIKES]->(t:Track)-[:INCLUDED\_IN]->(p:Playlist)

MATCH (t2:Track)-[:INCLUDED\_IN]->(p)

WHERE NOT (u)-[:LIKES]->(t2)

MERGE (u)-[:RECOMMENDED\_PLAYLIST]->(p);

1. **Связь исполнителей через совместные плейлисты:**

MATCH (a1:Artist)<-[:PERFORMED\_BY]-(t1:Track)-[:INCLUDED\_IN]->(p:Playlist)

MATCH (t2:Track)-[:INCLUDED\_IN]->(p)<-[:PERFORMED\_BY]-(a2:Artist)

WHERE a1 <> a2

MERGE (a1)-[:RELATED\_ARTIST]->(a2);

Эти запросы автоматически создают новые связи в графовой базе, аналогично SWRL-правилам в Protégé, что позволяет выполнять логический вывод и анализ взаимодействий между сущностями.

1. **Реализация в Protégé**

Для построения онтологической модели использовался инструмент **Protégé**, который позволяет создавать OWL-онтологии, задавать свойства и правила, а также выполнять логический вывод.

**Рисунок 1 – Дерево классов**

* Отображает иерархию всех классов модели: Artist, Album, Track, Playlist.
* Видно, как классы структурированы и какие объекты могут быть связаны между собой.

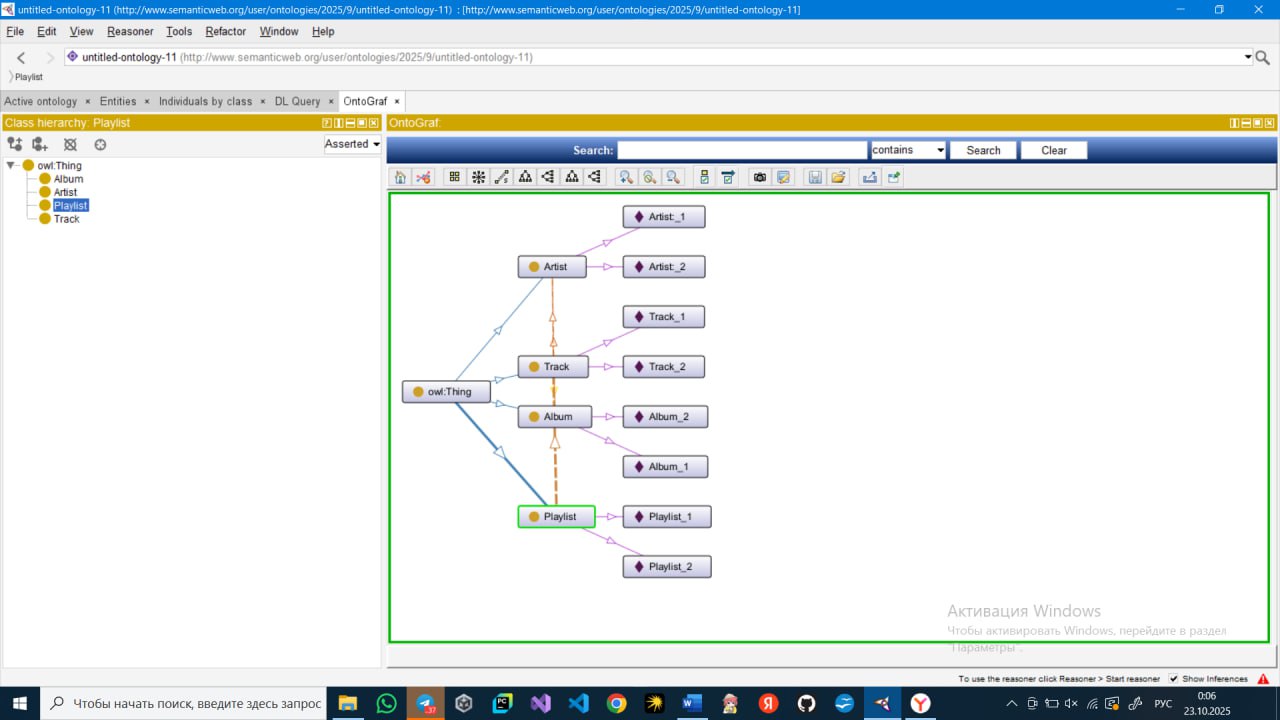


Рисунок 1 – Дерево классов

**Рисунок 2 – Object Properties**

Показывает все свойства объектов:

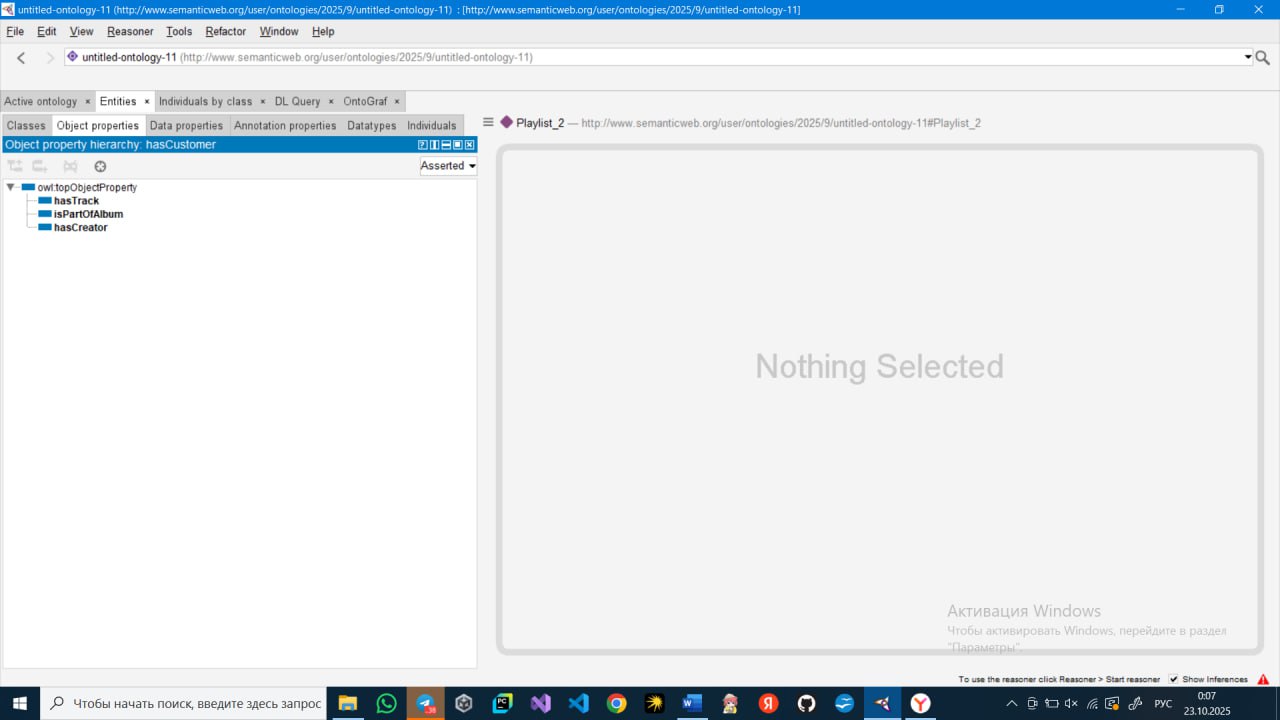


Рисунок 2 – Object Properties

**Рисунок 3 – Individuals**

Отображает конкретные экземпляры классов (индивиды):

* Исполнители: The Beatles, Taylor Swift
* Альбомы: Abbey Road, 1989
* Трек: "Come Together", "Shake It Off"
* Плейлист: "My Favorites", "Rock Classics"

Индивиды связаны между собой через объектные свойства, отражая реальные взаимодействия.

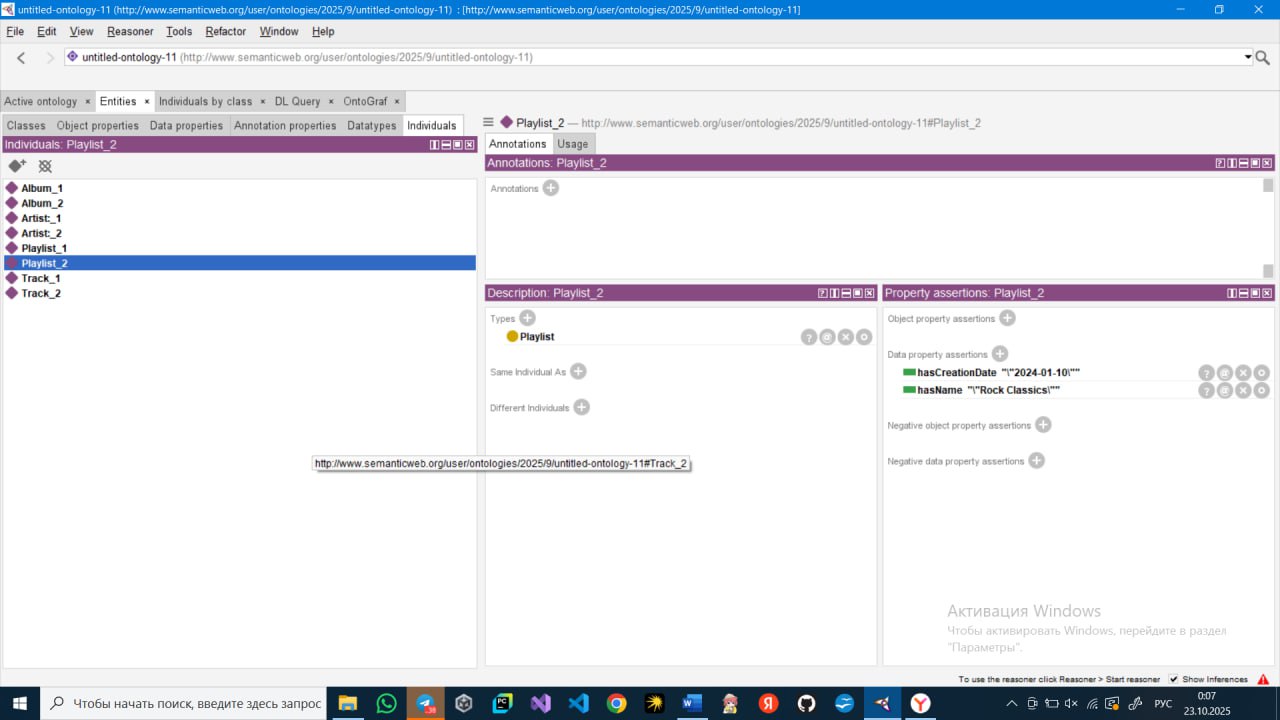


Рисунок 3 – Individuals

**Рисунок 4 – SWRLTab с правилами**

Показывает правила логического вывода, позволяющие автоматически создавать новые связи:

* Рекомендации плейлистов
* Связи между исполнителями
* Автоматическая классификация по жанрам

Благодаря этим правилам Protégé автоматически выводит логические отношения, которые не были явно заданы.

**Правило 1:** "Если Исполнитель создал Трек, и этот Трек является частью Альбома, то этот Исполнитель также является создателем этого Альбома".

hasCreator(?track, ?artist) ^ isPartOfAlbum(?track, ?album) -> hasCreator(?album, ?artist)

Объяснение: Если существует трек ?track, у которого создатель ?artist, и этот трек ?track является частью альбома ?album, то тогда сделай так, чтобы у альбома ?album тоже был создатель ?artist.

**Правило 2:** "Все, что имеет свойство hasTrack, является Плейлистом". (Это поможет автоматически классифицировать сущности).

hasTrack(?x, ?y) -> Playlist(?x)

Объяснение: Если некая сущность ?x имеет хотя бы один трек ?y, то эта сущность ?x является Плейлистом.

# 4. Реализация в Neo4j

Для построения графовой модели использовалась **Neo4j**, позволяющая хранить данные в виде узлов и связей, а также выполнять логические запросы через Cypher.

Создание узлов и связей (Cypher):

* CREATE

(a1:Artist {hasName: "The Beatles"}),

(a2:Artist {hasName: "Taylor Swift"}),

(alb1:Album {hasName: "Abbey Road", hasReleaseDate = "1969-09-26"}),

(alb2:Album {hasName: "1989", hasReleaseDate = "2014-10-27"}),

(t1:Track {hasName: "Come Together", hasDuration = 259}),

(t2:Track {hasName: "Shake It Off", hasDuration = 219}),

(p1:Playlist {hasName: "My Favorites", hasCreationDate = "2024-01-15"}),

(p2:Playlist {hasName: "Rock Classics", hasCreationDate = "2024-01-10"}),

(u1:User {username: "user123", email: "user@example.com"}),

(t1)-[:PERFORMED\_BY]->(a1),

(t2)-[:PERFORMED\_BY]->(a2),

(alb1)-[:CONTAINS\_TRACK]->(t1),

(alb2)-[:CONTAINS\_TRACK]->(t2),

(p1)-[:INCLUDES]->(t1),

(p1)-[:INCLUDES]->(t2),

(u1)-[:CREATED\_BY]->(p1),

(u1)-[:LIKES]->(t1),

(u1)-[:FOLLOWS]->(a1),

(service)-[:HAS\_ARTIST]->(a1),

(service)-[:HAS\_ARTIST]->(a2),

(service)-[:HAS\_ARTIST]->(a3),

(service)-[:HAS\_ALBUM]->(alb1),

(service)-[:HAS\_ALBUM]->(alb2),

(service)-[:HAS\_ALBUM]->(alb3),

(service)-[:HAS\_TRACK]->(t1),

(service)-[:HAS\_TRACK]->(t2),

(service)-[:HAS\_TRACK]->(t3),

(service)-[:HAS\_PLAYLIST]->(p1),

(service)-[:HAS\_PLAYLIST]->(p2),

(service)-[:HAS\_USER]->(u1);

**Рисунок 5 – Демонстрация связей**

* Визуализация графа показывает все узлы и их связи.
* Легко видеть, какие треки принадлежит альбому, кто их автор.

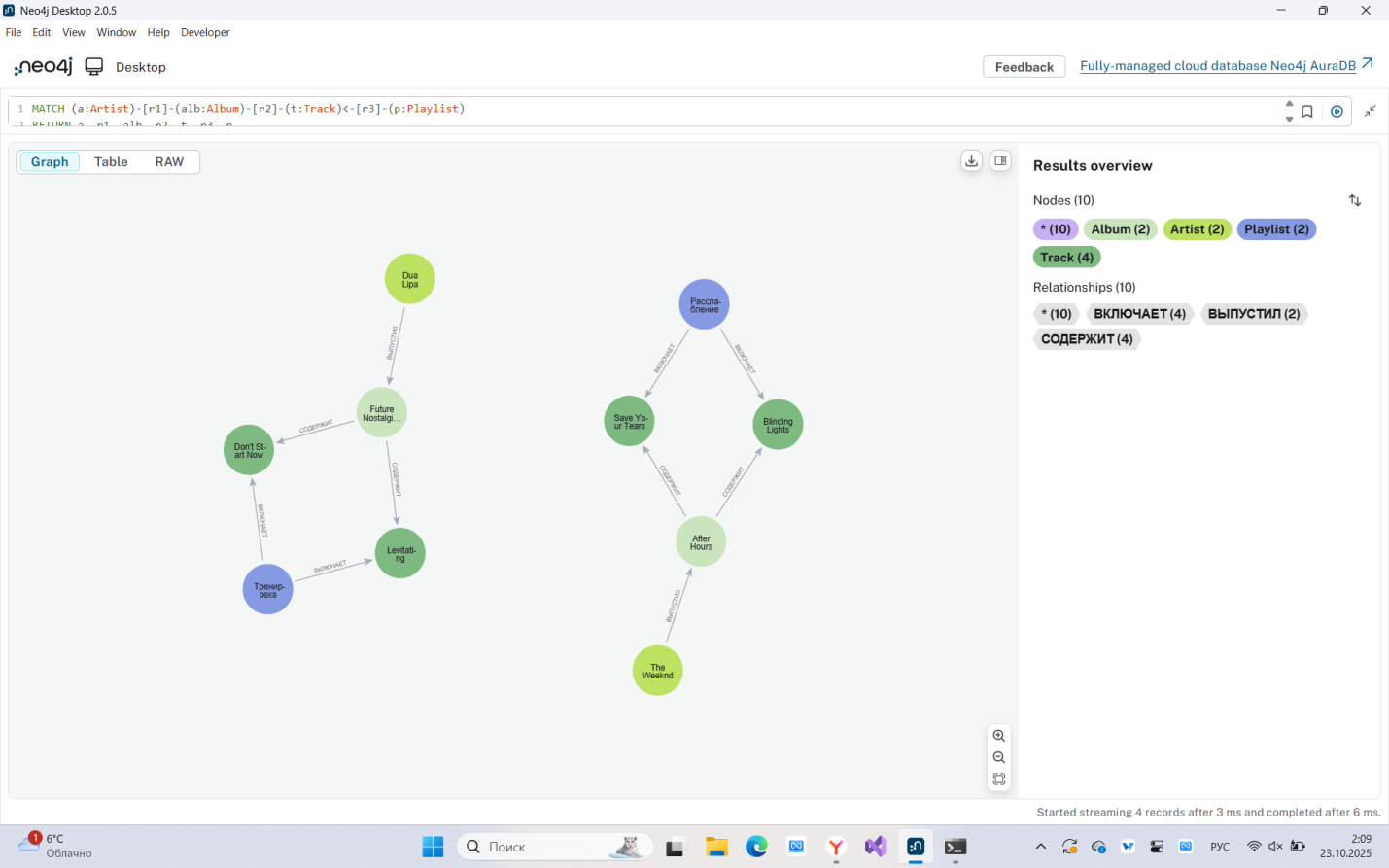


Рисунок 4 – Демонстрация связей

**5. Вывод**

В работе создана онтологическая модель музыкального сервиса, объединяющая **исполнителя, альбомы, треки и плейлисты**.

**SWRL-правила** и **Cypher-запросы** позволяют автоматически выводить обратные и косвенные связи.

Модель наглядно реализована в **Protégé** и **Neo4j**, что подтверждается визуализацией и скриншотами.

Практическая ценность: отслеживание взаимодействий между исполнителем, альбомами, треками и плейлистами, а также возможность расширения модели и анализа данных для музыкального сервиса системы.