

2024-2025 学年 知识工程（双语）实验报告

任课教师：吴天星

院 系 _____人工智能学院_____

专 业 _____人工智能_____

姓 名 _____陆文韬_____

任 务 _____知识查询_____

1 实验五

1.1 实验任务

针对下列自然语言，使用 SPARQL 查询语句返回结果：

1. List the creators (including paintings) of Guernica and Sunflowers, respectively
2. List all the artists (including living places) who live in Spain or other places
3. List all paintings, their names, and the corresponding techniques

1.2 基本概念

1.2.1 RDF4J

RDF4J 是一个用于处理 RDF（资源描述框架，Resource Description Framework）数据的开源 Java 库，提供了丰富的工具，帮助开发者在 Java 环境中存储、查询、推理和管理 RDF 数据。它是一个轻量级的框架，旨在提升开发者处理语义网和链接数据（Linked Data）的效率。

RDF（资源描述框架）概述

RDF 是一种用于描述资源（例如网页、实体或数据）的框架，特别适用于表示“三元组”形式的语义数据。每个 RDF 三元组由三个部分组成：

- **主题 (Subject)**：被描述的资源。
- **谓词 (Predicate)**：资源的属性或关系。
- **宾语 (Object)**：资源的值或与主题相关联的另一个资源。

例如，RDF 三元组可以描述如下：

- 主题：Alice
- 谓词：knows
- 宾语：Bob

RDF4J 核心功能

RDF4J 提供了多个组件来支持 RDF 数据的存储、查询、推理和管理，主要包括以下功能：

- **RDF 数据存储和管理**：RDF4J 支持将 RDF 数据存储在内​​存、文件、数据库等不同的后端存储中，并提供 SPARQL 端点进行查询。

- **SPARQL 查询支持**: RDF4J 完全支持 SPARQL 1.0 和 1.1, 允许执行 SELECT、CONSTRUCT、ASK 和 DESCRIBE 查询。
- **推理支持**: 提供推理引擎, 支持基于 RDFS 和 OWL 推理。
- **数据导入与导出**: 支持多种 RDF 数据格式, 如 RDF/XML、Turtle、N-Triples 和 JSON-LD。
- **事务管理**: 支持对 RDF 存储库的事务操作, 确保数据一致性。

RDF4J 主要组件

RDF4J 包含以下核心组件:

- **RDF4J API**: 提供操作 RDF 数据的基础 API。
- **RDF4J 存储库 (Repository)**: 用于存储和管理 RDF 数据的核心组件。
- **SPARQL 服务端**: 用于执行 SPARQL 查询的服务端。
- **推理引擎**: 基于 RDF 图中的规则和数据进行推理。

RDF4J 的使用场景

RDF4J 在以下领域中得到广泛应用:

- **语义网和链接数据**: 用于表示和查询语义数据, 应用于知识图谱、社交网络分析等。
- **本体管理**: 推理和查询本体数据, 处理大规模的本体。
- **开放数据集**: 处理开放数据, 通常采用 RDF 格式进行表示。
- **数据集成**: 利用 RDF 数据模型将异构数据源进行集成和统一表示。

与其他工具的对比

与其他 RDF 处理框架 (如 Apache Jena) 相比, RDF4J 具有以下特点:

- **轻量级**: 提供相对简单的 API, 适合轻量级和嵌入式应用。
- **高性能**: 提供较好的性能和查询优化, 特别适合大规模数据集。
- **灵活性**: 支持多种 RDF 存储和查询引擎, 适用于多种应用场景。

总结

RDF4J 是一个功能强大且灵活的工具库, 适用于任何需要处理 RDF 数据、执行 SPARQL 查询、进行推理的 Java 应用。它的设计考虑到性能、可扩展性以及 RDF 格式的全面支持, 广泛应用于语义网、大数据集成等领域。如果您需要处理复杂的 RDF 数据或开发语义网应用, RDF4J 提供了丰富的功能和工具, 是一个重要的选择。

1.2.2 SPARQL

SPARQL (SPARQL Protocol and RDF Query Language) 是用于查询和操作 RDF (资源描述框架) 数据的标准查询语言。SPARQL 提供了强大的查询功能, 可以从 RDF 数据集中提取、更新、删除和插入数据。作为语义网技术的重要组成部分, SPARQL 被广泛应用于处理和查询大规模语义数据。以下是 SPARQL 的详细介绍。

SPARQL 的基本概念

SPARQL 查询语言允许用户基于 RDF 三元组的结构, 通过模式匹配来执行查询。SPARQL 查询的基本单元是三元组查询, 它与 RDF 的数据表示方式紧密相关。每个 SPARQL 查询通常由以下几个部分组成:

- **查询模式:** 通过变量和常量的结合, 定义数据的结构模式。
- **查询语句:** 指定查询操作, 包括 SELECT、ASK、CONSTRUCT 和 DESCRIBE 等。
- **数据限制:** 可以对查询结果进行过滤、排序和分组等操作。

SPARQL 查询类型

SPARQL 支持四种主要的查询类型:

- **SELECT 查询:** 用于选择满足条件的 RDF 数据。常用于获取查询结果的某些变量或数据。
- **ASK 查询:** 返回一个布尔值, 指示是否存在满足查询模式的数据。
- **CONSTRUCT 查询:** 构造新的 RDF 图, 通常用于根据查询结果创建新的三元组。
- **DESCRIBE 查询:** 返回一个描述指定资源的 RDF 图, 通常用于获取有关资源的详细信息。

SPARQL 查询的基本结构

一个标准的 SPARQL 查询通常包括以下几个部分:

- **PREFIX:** 用于声明前缀, 以便简化 URIs 的书写。
- **SELECT:** 指定查询返回的变量。
- **WHERE:** 定义查询的条件, 即三元组模式。
- **FILTER:** 用于对查询结果进行过滤, 例如基于某些条件限制结果。
- **ORDER BY:** 用于对查询结果进行排序。

例如, 下面是一个简单的 SPARQL 查询示例:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE {
    ?person foaf:name ?name .
    ?person foaf:knows <http://example.org/alice> .
}
```

该查询将返回所有与 Alice 认识的人及其名称。

SPARQL 的功能和特点

SPARQL 提供了许多强大的功能，特别适用于从 RDF 数据中提取信息：

- **多模式查询：**SPARQL 支持通过多重条件查询多个资源，查询结果可以包括多个变量。
- **文本搜索：**SPARQL 支持对数据进行文本搜索，可以查找包含指定关键字的资源。
- **路径查询：**SPARQL 允许查询资源之间的关系路径，尤其适用于复杂的数据模型。
- **数据合并：**SPARQL 查询可以跨多个数据集执行操作，通过联合查询获取来自不同来源的数据。

SPARQL 与 RDF 数据的关系

SPARQL 查询语言是专门为 RDF 数据设计的。RDF 数据以三元组形式表示，SPARQL 通过模式匹配来查询这些三元组。在 SPARQL 查询中，用户定义查询模式（通常是一个或多个三元组），然后 SPARQL 引擎查找所有符合该模式的 RDF 数据。SPARQL 的强大之处在于它能够处理大规模的 RDF 数据集，并通过高效的查询语法提取所需信息。

SPARQL 端点

SPARQL 端点是一种通过 HTTP 提供 SPARQL 查询服务的网络接口。用户可以通过 SPARQL 端点对存储在远程服务器上的 RDF 数据进行查询。许多开放数据集提供了 SPARQL 端点，允许用户对公共数据进行查询分析。例如，DBpedia 和 Wikidata 都提供了 SPARQL 端点供开发者和数据分析师使用。

SPARQL 的应用场景

SPARQL 作为语义网的核心技术之一，具有广泛的应用场景：

- **知识图谱：**SPARQL 在知识图谱中用于查询和提取各种实体和它们之间的关系。
- **数据集成：**SPARQL 用于跨不同数据源整合信息，尤其是在多个 RDF 数据集之间。
- **开放数据集访问：**许多政府和企业提供开放数据集，并通过 SPARQL 端点供公众查询。
- **语义搜索：**SPARQL 用于增强搜索引擎的语义能力，使得查询结果更符合用户需求。

总结

SPARQL 是一种强大且灵活的查询语言，专门用于查询和操作 RDF 数据。它支持丰富的查询功能，如数据选择、布尔检查、数据构建和资源描述。通过 SPARQL，用户可以从 RDF 数据集中高效地提取和分析信息，广泛应用于知识图谱、语义搜索、数据集成等领域。SPARQL 的设计使其成为语义网和链接数据的核心技术之一，是大规模 RDF 数据查询的标准工具。

1.2.3 Maven

Maven 是一个开源的构建自动化工具，主要用于 Java 项目的构建、管理和依赖管理。它由 Apache 软件基金会开发，并遵循项目对象模型（POM, Project Object Model）的结构。Maven 提供了统一的构建系统，通过描述项目的结构、依赖、插件等信息，实现自动化构建、测试和部署等操作。以下是 Maven 的详细介绍。

Maven 的核心概念

Maven 的核心概念包括以下几个方面：

- **POM (Project Object Model)**：每个 Maven 项目都由一个名为 `pom.xml` 的 POM 文件来描述，它包含了项目的基本信息、依赖、插件、构建配置等。
- **依赖管理**：Maven 允许自动下载和管理项目所依赖的库及其版本。通过在 POM 文件中声明依赖，Maven 会自动从中央仓库或指定的仓库下载所需的库。
- **生命周期**：Maven 提供了一套标准的构建生命周期，包括清理、编译、测试、打包、部署等过程，每个生命周期由多个阶段（phase）组成。
- **插件 (Plugin)**：Maven 通过插件来执行具体的构建任务，如编译源代码、运行测试、生成报告等。插件在 POM 文件中进行配置。
- **仓库 (Repository)**：Maven 使用仓库来存储项目的构建结果和依赖库。主要包括本地仓库、中央仓库和远程仓库。

Maven 的功能和优势

Maven 通过统一的配置管理和自动化构建，提供了许多强大的功能，具体包括：

- **构建自动化**：Maven 通过 POM 文件描述项目构建的过程，自动化管理项目的构建过程，减少了手动配置的繁琐。
- **依赖管理**：Maven 支持自动管理项目所依赖的库，用户只需在 POM 文件中声明依赖，Maven 会自动下载和配置所需的依赖。
- **项目标准化**：Maven 强制约定项目结构，所有 Maven 项目都有一个相似的目录结构，有利于团队协作和项目管理。

- **插件化扩展**：Maven 支持插件机制，可以根据需要扩展功能，提供对编译、测试、打包、部署等各个阶段的定制化支持。
- **统一的构建过程**：通过 Maven，所有项目都遵循相同的构建过程，提升了项目构建的一致性和可维护性。
- **支持多模块项目**：Maven 支持多模块项目的构建和管理，通过父子 POM 文件的继承与聚合，简化了多模块项目的管理。

Maven 的基本使用

Maven 的使用主要依赖于 POM 文件，该文件定义了项目的结构、依赖、构建信息等。一个典型的 Maven 项目的目录结构如下：

```
project-root
  pom.xml          % 项目配置文件
  src
    main
      java         % Java 源代码
      resources    % 资源文件
    test
      java         % 测试代码
      resources    % 测试资源
  target           % 构建输出目录
```

在 POM 文件中，开发者需要声明项目的依赖、插件和其他构建相关的信息。例如，下面是一个简单的 POM 文件示例：

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>myproject</artifactId>
  <version>1.0-SNAPSHOT</version>

  <dependencies>
    <dependency>
      <groupId>org.apache.commons</groupId>
      <artifactId>commons-lang3</artifactId>
      <version>3.12.0</version>
    </dependency>
```

```
</dependencies>  
</project>
```

在上述示例中，项目的依赖是 `commons-lang3` 库，Maven 会自动从中央仓库下载该依赖。

Maven 生命周期

Maven 的构建生命周期是由多个阶段组成的，每个阶段完成一个特定的任务。常见的生命周期包括：

- **clean**：清理之前的构建输出，通常是删除 `target` 目录。
- **default**：包含了项目的主要构建过程，通常包括编译、测试、打包、安装等。
- **site**：生成项目的网站文档。

每个生命周期由多个阶段（phase）组成，例如，`default` 生命周期包含的阶段有：

- **compile**：编译源代码。
- **test**：运行单元测试。
- **package**：将项目打包成 JAR 或 WAR 文件。
- **install**：将打包的文件安装到本地 Maven 仓库。
- **deploy**：将打包的文件部署到远程仓库。

Maven 与其他构建工具的对比

与其他构建工具（如 Ant 和 Gradle）相比，Maven 有以下优势：

- **依赖管理**：Maven 提供了强大的依赖管理机制，可以自动下载、更新和管理项目的依赖。
- **标准化的构建过程**：Maven 强制使用标准的项目结构和构建生命周期，使得不同的 Java 项目具有一致性。
- **广泛的插件支持**：Maven 拥有大量的插件，可以用于测试、构建、部署等各个方面。

总结

Maven 是一个功能强大的构建自动化工具，它通过标准化的 POM 文件、生命周期和插件化的构建过程，使得 Java 项目的构建、测试、部署等变得更加高效和可管理。Maven 通过自动化依赖管理、构建过程管理和多模块支持，帮助开发者减少了繁琐的手动操作，提升了项目的构建一致性和可维护性。它已经成为 Java 开发中不可或缺的工具之一。

1.3 实验过程

1.3.1 问题一

问题一要求列出 Guernica 和 Sunflowers 的作者和画作，以下是 SPARQL 查询语句：

```
PREFIX ex: <http://example.org/>
SELECT ?s ?p WHERE {
  ?s ex:creatorOf ?p .
  ?p rdfs:label "Guernica" .
}
```

如果要查询 Sunflowers 对应的内容，则只需要将代码中的“Guernica”替换为“Sunflowers”即可。

但在将 SPARQL 查询语句转换为 Java 语句的时候遇到了一些问题。在一开始，我模仿例子中的代码结构，书写了如下语句：

```
String queryString = "PREFIX ex: <http://example.org/> \n";
queryString += "PREFIX foaf: <" + FOAF.NAMESPACE + "> \n";
queryString += "SELECT ?s ?p\n";
queryString += "WHERE { \n";
queryString += "    ?s ex:creatorOf ?p ; \n";
queryString += "    ?p rdfs:label \"Guernica\" .";
queryString += "}";
```

图 1: 报错的代码

在查询了诸多资料以及数次更改之后，我将代码更改为如下形式，程序可以正常运行：

```
String queryString = "PREFIX ex: <http://example.org/>\n" +
    "SELECT ?s ?p WHERE {\n" +
    "    ?s ex:creatorOf ?p .\n" +
    "    ?p rdfs:label \"Sunflowers\" .\n" +
    "}";
```

图 2: 正确的代码

通过比较两个代码形式的不同，我发现错误在于我在原本应该使用句点的时候误用了分号。同时考虑到第一种代码字符串的拼接逻辑书写较为复杂，容易产生错误，但是在后者则不存在这个问题，为了提高效率，减少错误的发生，我在接下来的代码中均采用了第二种写法。

问题一查询得到的结果如下所示：

List the creators(including paintings) of Guernica:

?s = <http://example.org/Picasso>

?p = <http://example.org/guernica>

List the creators(including paintings) of Sunflowers:

?s = <http://example.org/VanGogh>

?p = <http://example.org/sunflowers>

1.3.2 问题二

问题二要求列出所有住在西班牙或者其他地方的艺术家以及他们对应的居住地。这个问题中特殊点在于，ex:country 这个属性并非可以直接获取，而是储存在一个空白节点当中，空白节点可以用来表示一个复杂的结构化数据，也可以用来表示一个匿名的资源，在此处，_node:1 这个空白节点表示家庭住址这一复杂的结构化数据（家庭住址中包含了街道、城市和国家三个不同的属性），因此，在获取艺术家所在的国家的过程中，我们不能直接获取这一属性的值，而是需要借助一个中间变量。

```
ex:Picasso a ex:Artist ;
    foaf:firstName "Pablo" ;
    foaf:surname "Picasso";
    ex:creatorOf ex:guernica ;
    ex:homeAddress _:node1 .

_:node1 ex:street "31 Art Gallery" ;
        ex:city "Madrid" ;
        ex:country "Spain" .
```

图 3: 数据结构

问题二所需要的 SPARQL 查询语句如下：

```
PREFIX ex: <http://example.org/>
SELECT ?s ?c WHERE {
    ?s a ex:Artist .
    OPTIONAL {
        ?s ex:homeAddress ?h .
        ?h ex:country ?c .
    }
}
```

```
}
```

对应的 Java 代码如下:

```
String queryString = "PREFIX ex: <http://example.org/>\n" +
    "SELECT ?s ?c WHERE {\n" +
    "    ?s a ex:Artist .\n" +
    "    OPTIONAL {\n" +
    "        ?s ex:homeAddress ?h .\n" +
    "        ?h ex:country ?c .\n" +
    "    }\n" +
    "}";
```

问题二查询得到的结果如下:

```
List all the artists(including living spaces) who live in Spain or other places:
?s = http://example.org/Picasso
?c = "Spain"
?s = http://example.org/VanGogh
?c = null
```

1.3.3 问题三

问题三要求列出所有画作、对应的名称和对应的绘画技法, 观察画作的数据结构可以看到, 这三个属性都是可以直接获取的属性, SPARQL 语句如下:

```
PREFIX ex: <http://example.org/>
SELECT ?p ?n ?t WHERE {
    ?p a ex:Painting .
    ?p rdfs:label ?n .
    ?p ex:technique ?t .
}
```

对应的 Java 代码如下:

```
String queryString = "PREFIX ex: <http://example.org/>\n" +
    "SELECT ?p ?n ?t WHERE {\n" +
    "    ?p a ex:Painting .\n" +
    "    ?p rdfs:label ?n .\n" +
    "    ?p ex:technique ?t .\n" +
    "}";
```

问题三查询得到的结果如下:

```
List all paintings, their names, and the corresponding techniques:
?p = http://example.org/guernica
?n = "Guernica"
?t = "oil on canvas"
?p = http://example.org/starryNight
?n = "Starry Night"
?t = "oil on canvas"
?p = http://example.org/sunflowers
?n = "Sunflowers"
?t = "oil on canvas"
?p = http://example.org/potatoEaters
?n = "The Potato Eaters"
?t = "oil on canvas"
```

2 实验六

2.1 实验任务

1. 导入 contact-tracing-43.dump 文件到数据库 neo4j 中
2. 查询名叫 Madison Odonnell 的人物节点，并记录下该节点的 healthstatus、name、confirmed-times 属性和属性值
3. 将该人物节点及与其相连的关系删除，并检查是否删除成功
4. 重新创建该节点以及第 2 步记录下来的节点属性
5. 重新创建关系：Madison Odonnell 的人物节点与名为 ‘Place nr 40’ 的 Place 节点间的关系，不考虑关系属性
6. Madison Odonnell 不幸被确诊为新冠（healthstatus= ‘sick’），对图谱进行更新

2.2 基本概念

2.2.1 Neo4j

Neo4j 是一种基于图数据库的开源 NoSQL 数据库系统，专为存储和管理高度连接的数据而设计。与传统的关系型数据库不同，Neo4j 使用图形数据结构（节点、边和属性）来表示和存储数据，并通过图遍历和高效的查询机制实现快速查询。Neo4j 的主要应用领域包括知识图谱、社交网络分析、推荐系统、网络安全和供应链管理等。

图数据库的基本概念

图数据库以图（Graph）的形式组织和存储数据，其核心概念包括：

- **节点 (Node)**: 图中的基本实体, 用于表示对象 (如用户、产品、地点等)。每个节点可以拥有属性 (key-value 对)。
- **边 (Relationship)**: 连接两个节点的有向或无向关系, 用于表示实体之间的关联。每个边也可以包含属性。
- **属性 (Property)**: 节点或边的键值对, 描述实体或关系的特征。
- **标签 (Label)**: 为节点定义类别或分类 (如 User、Product)。
- **图模式 (Graph Schema)**: 虽然图数据库通常是模式自由的, 但可以定义特定的模式以指导数据建模。

Neo4j 的特点与优势

Neo4j 作为图数据库的领先实现, 具备以下特点和优势:

- **高效的图遍历**: Neo4j 针对图结构优化查询操作, 使得复杂的图遍历 (如寻找最短路径、邻居节点等) 非常高效。
- **灵活的数据建模**: 支持模式自由的建模, 可以根据实际需求动态添加节点、边和属性。
- **查询语言 Cypher**: Neo4j 使用声明性图查询语言 Cypher, 语法直观, 易于表达复杂的图操作。
- **事务支持**: Neo4j 是 ACID 合规的数据库, 支持事务管理, 保证数据一致性和可靠性。
- **强大的可视化工具**: 内置交互式图数据可视化功能, 方便用户探索和分析数据。
- **水平扩展能力**: 支持分布式部署和水平扩展, 可以处理大规模的图数据集。

Cypher 查询语言

Cypher 是 Neo4j 提供的一种声明式查询语言, 用于高效地操作和查询图数据。Cypher 的语法类似于 SQL, 但专门针对图结构。以下是一个简单的 Cypher 查询示例:

```
MATCH (a:Person)-[:KNOWS]->(b:Person)
WHERE a.name = "Alice"
RETURN b.name
```

该查询语句的含义是: 查找所有与 Alice 存在 KNOWS 关系的 Person, 并返回这些人的名字。

Neo4j 的常见应用场景

Neo4j 的图结构使其在以下领域具有广泛的应用:

- **知识图谱**: 用于存储和查询实体及其关系, 常见于语义网、搜索引擎等。
- **社交网络分析**: 分析社交网络中的用户关系、群体行为、影响力等。

- **推荐系统**：通过用户行为和关系，生成个性化的产品或服务推荐。
- **网络安全**：检测和分析网络中的安全威胁，例如识别恶意攻击路径。
- **供应链管理**：优化供应链中的物流路径、库存管理及合作伙伴关系。
- **医疗与基因研究**：建模基因组关系、疾病与药物的相关性等。

Neo4j 与传统数据库的对比

与传统关系型数据库（如 MySQL 和 PostgreSQL）相比，Neo4j 的优势在于高效处理连接性强的数据。关系型数据库需要通过复杂的连接（JOIN）操作来处理关系，而 Neo4j 通过图结构直接存储关系，从而显著提升查询性能。具体对比如下：

- **连接性查询**：Neo4j 针对高度连接的数据（如社交网络）优化性能，而关系型数据库在处理复杂连接时性能较低。
- **灵活性**：Neo4j 支持模式自由的建模，方便动态扩展；而关系型数据库需要预定义模式。
- **存储结构**：Neo4j 使用图模型存储数据，而关系型数据库基于表结构。

总结

Neo4j 是一种专注于图数据存储和查询的数据库系统，以其高效的图遍历、灵活的数据建模和强大的查询语言 Cypher 在知识图谱、社交网络分析、推荐系统等领域得到了广泛应用。作为图数据库的领导者，Neo4j 是处理复杂关系和高度连接数据的理想选择，为解决数据分析中的图问题提供了强有力的支持。

2.2.2 Cypher

Cypher 是 Neo4j 图数据库使用的一种声明式查询语言，用于高效地查询和操作图数据。类似于 SQL 的语法风格，Cypher 提供了直观且易于理解的方式来表达复杂的图查询操作，例如匹配模式、插入数据、更新数据以及删除数据等。Cypher 是图数据领域中广泛使用的语言之一，专为处理节点（Node）、关系（Relationship）和属性（Property）的查询而设计。

Cypher 的核心概念

Cypher 语言的核心概念基于图模型，包括以下基本元素：

- **节点 (Node)**：表示图中的实体，用圆括号 () 表示。例如，(a:Person) 表示一个标签为 Person 的节点。
- **关系 (Relationship)**：表示节点之间的边，用方括号 [] 表示。例如，[r:KNOWS] 表示一个关系类型为 KNOWS 的关系。
- **属性 (Property)**：表示节点或关系的属性，存储为键值对。例如，a.name = "Alice" 表示节点 a 的属性 name 值为 "Alice"。

- **模式匹配 (Pattern Matching)**: 通过匹配图中的模式提取所需数据。例如, (a)-[r]->(b) 表示从节点 a 到节点 b 的有向关系 r。

Cypher 的主要查询功能

Cypher 提供了多种操作图数据的功能, 常见功能包括:

- **MATCH 查询**: 用于匹配图中的模式。是 Cypher 中最基本的查询子句。
- **RETURN 结果返回**: 用于返回查询结果。
- **WHERE 条件过滤**: 用于对查询结果应用条件限制。
- **CREATE 数据插入**: 用于在图中创建节点、关系或属性。
- **SET 数据更新**: 用于更新节点或关系的属性。
- **DELETE 数据删除**: 用于删除节点或关系。
- **OPTIONAL MATCH**: 执行可选模式匹配, 不影响查询整体结果。

Cypher 查询语法示例

以下是一些 Cypher 查询语句的示例:

-- 匹配一个图模式

```
MATCH (a:Person)-[r:KNOWS]->(b:Person)
WHERE a.name = "Alice"
RETURN b.name
```

-- 插入一个节点和关系

```
CREATE (p:Person {name: "Bob"})
CREATE (a:Person {name: "Alice"})-[:KNOWS]->(p)
```

-- 更新节点属性

```
MATCH (p:Person {name: "Bob"})
SET p.age = 30
```

-- 删除节点及其关系

```
MATCH (p:Person {name: "Bob"})
DETACH DELETE p
```

Cypher 的特点

Cypher 作为一种面向图的查询语言, 具有以下特点:

- **声明式语法**: 与 SQL 类似, 用户只需描述“要查询什么”, 无需关心具体的实现细节。
- **图模式直观**: 通过图形化的语法 (如 `()` 和 `[]`), 使得查询逻辑更加贴近数据的图模型。
- **高效处理关系**: 在查询节点之间的关系时, Cypher 的性能优于传统关系型数据库的复杂 JOIN 操作。
- **可读性强**: 语法简洁直观, 便于开发者快速上手。

Cypher 的应用场景

Cypher 的查询能力使其在许多应用场景中得到了广泛应用:

- **社交网络分析**: 查询用户之间的关系网络, 例如好友推荐。
- **知识图谱查询**: 在知识图谱中查找实体及其关联关系。
- **路径分析**: 查询节点之间的最短路径。
- **推荐系统**: 根据用户和内容的关系生成个性化推荐。

Cypher 与 SQL 的对比

虽然 Cypher 和 SQL 都是声明式查询语言, 但它们有显著的差异:

- **数据模型**: SQL 针对表格数据, 而 Cypher 针对图数据 (节点和关系)。
- **查询机制**: SQL 使用 JOIN 操作处理表之间的关系, 而 Cypher 直接通过关系边处理。
- **表达方式**: Cypher 使用图形化的模式匹配, 语法更加贴近图数据的直观表达。

总结

Cypher 是一种高效、灵活的图查询语言, 为处理和操作图数据提供了强大的功能。通过声明式语法和直观的图模式表示, Cypher 广泛应用于社交网络分析、知识图谱、推荐系统等领域, 是图数据库 (特别是 Neo4j) 用户的核心工具。

2.3 实验过程

2.3.1 问题一

首先需要导入数据库, 将数据文件中的 `contact-tracing-43.dump` 文件移动到 `neo4j/import` 文件夹下, 然后输入以下指令来导入数据:

```
C:\Users\PC\Desktop\knowledge_querying - 2024\Neo4j\Neo4j实验安装包与数据\neo4j-community-4.4.1-windows>neo4j-admin load --from=import/contact-tracing-43.dump --database=neo4j --force
Selecting JVM - Version:11.0.13+10-LTS-370, Name:Java HotSpot(TM) 64-Bit Server VM, Vendor:Oracle Corporation
Done: 87 files, 7.07GB processed.
```

导入成功之后, 我们进入 `data/databases` 文件下, 进入 `neo4j` 文件夹, 可以看到如下内容:


```
MATCH (n:Person {name: "Madison Odonnell"})
RETURN n.healthstatus, n.name, n.confirmedtime
```

查询得到的结果如下：



n.healthstatus	n.name	n.confirmedtime
"Healthy"	"Madison Odonnell"	"2020-04-25T23:09:38Z"

图 5: Madison Odonnell 相关属性

2.3.3 问题三

问题三要求将该人物节点及与其相连的关系删除，并检查是否删除成功。

可以使用以下语句删除节点及与其相连的关系：

```
MATCH (n:Person {name: "Madison Odonnell"})
DETACH DELETE n
```

可以使用以下语句检查是否删除成功：

```
MATCH (n:Person {name: "Madison Odonnell"})
RETURN n
```

删除结果如下：



Table
Deleted 1 node, deleted 8 relationships, completed after 26 ms.

图 6: 删除 Madison Odonnell 节点和相连关系

检查是否删除成功：

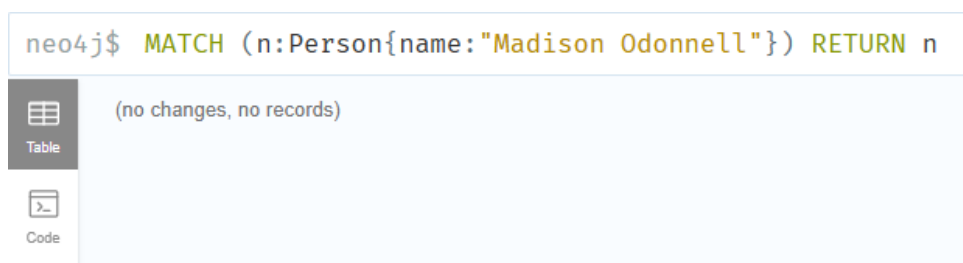


Table
(no changes, no records)

图 7: 检查是否删除成功的结果

2.3.4 问题四

问题四要求重新创建 Madison Odonnell 节点以及第二步记录下来的节点属性。

```
CREATE (n:Person {
  healthstatus: "Healthy",
  name: "Madison Odonnell",
  confirmedtime: "2020-04-25T23:09:38Z"
})
```

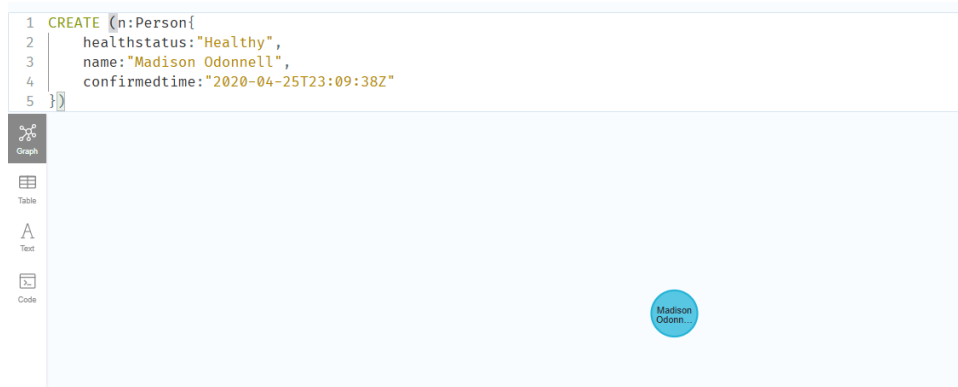


图 8: Madison Odonnell 节点重新创建的结果

2.3.5 问题五

问题五要求重新创建关系，在 Madison Odonnell 和一个名为 ‘Place nr 40’ 的 Place 节点间的关系。

先查看 Place 节点的属性：

Place		
<id>	559	
homelocation	point({srid:7203, x:51.20490471, y:4.39312909})	
id	40	
name	Place nr 40	
type	Park	

图 9: Place 节点的相关情况

使用以下语句建立关系：

```

MATCH (n:Person {name: "Madison Odonnell"}), (p:Place {name: "Place nr 40"})
CREATE (n)-[r:PERFORMS_VISIT]->(p)
RETURN n, r, p

```

在刚刚阅读这道题的时候, 我错误考虑了“不考虑关系属性”这一语句的含义, 误以为是在书写 Cypher 语句的时候可以不写明关系的名称, 结果发生了错误。然后, 我才意识到, 所谓的“不考虑关系属性”是指在书写 Cypher 语句的时候可以任意选取一个关系来书写, 也就是 PERFORMS_VISIT 和 VISIT 这两个关系任意选取一个。



图 10: 错误的情况

2.3.6 问题六

问题六要求对图谱中的信息进行更新, 将 Madison Odonnell 节点的 healthstatus 这一属性的值从 Healthy 更改为 sick。

```

MATCH (n:Person {name: "Madison Odonnell"})
SET n.healthstatus = 'sick'
RETURN n

```

更新数据之后得到的结果如下:

