Training Day 15
Report

3 July 2024

Key Takeways:

SPARQL Query Basics

SPARQL statements are constructed using the following basic query types, clauses, and other solution modifiers.

Standard Query Types

- **SELECT:** Run SELECT queries when you want to find and return all of the data that matches certain patterns.
- **CONSTRUCT:** Run CONSTRUCT queries when you want to create or transform data based on the existing data.
- **ASK:** Run ASK queries when you want to know whether a certain pattern exists in the data. ASK queries return only "true" or "false" to indicate whether a solution exists.
- **DESCRIBE:** Run DESCRIBE queries when you want to view the RDF graph that describes a particular resource.

Query Clauses

All queries may also include one or more of the following clauses:

- **PREFIX Clause:** The optional PREFIX clause declares the abbreviations that you want to use to reference URIs in the query.
- **FROM Clause:** The optional FROM clause defines the data sets or graphs to query. By default, if you do not specify the FROM clause, the scope of a query is limited to the default graph.
- **WHERE Clause:** The WHERE clause specifies the query pattern for data to match in the graphs or data sets specified in the query.

Solution Modifiers

The following optional query modifiers enable you to restrict, group, sort, or further refine query results:

- **ORDER BY**: This modifier sorts the result set in a particular order. It sorts query solution results based on the value of one or more variables.
- **OFFSET:** Using this modifier in conjunction with LIMIT and ORDER BY returns a slice of a sorted solution set, for example, for paging.
- **LIMIT:** This modifier puts an upper bound on the number of results returned by a query.
- **GROUP BY:** This modifier is used with aggregate functions and specifies the key variables to use to partition the solutions into groups.

For information about the AnzoGraph DB GROUP BY clause extensions, see Advanced Grouping Sets.

- **HAVING:** This modifier is used with aggregate functions and further filters the results after applying the aggregates.

EXAMPLE:

1.Retrive All Tuples:

sparql

```
SELECT ?subject ?predicate ?object
WHERE {
  ?subject ?predicate ?object.
}
```

Explanation: This query selects all triples from the dataset. Each triple has a subject, predicate, and object. It essentially retrieves all the data in the RDF graph.

2. Retrieve all individuals with their names and branches:

sparql

```
PREFIX ns1: <http://example.org/>
```

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?id ?name ?branch
WHERE {
  ?id rdf:type rdf:Description .
  ?id ns1:name ?name .
  ?id ns1:branch ?branch .
}
```

3. Retrieve all individuals with a specific branch (e.g., Branch_1):

```
sparql
PREFIX ns1: <http://example.org/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?id ?name
WHERE {
  ?id rdf:type rdf:Description .
  ?id ns1:name ?name .
  ?id ns1:branch "Branch_1" .
}
```

4. Count the number of individuals in each branch:

```
sparql
```

```
PREFIX ns1: <http://example.org/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?branch (COUNT(?id) AS ?count)
WHERE {
  ?id rdf:type rdf:Description .
  ?id ns1:branch ?branch .
}
GROUP BY ?branch
```

5. Retrieve individuals with a specific name:

```
sparql
PREFIX ns1: <http://example.org/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?id ?branch
WHERE {
  ?id rdf:type rdf:Description .
  ?id ns1:name "Name_15" .
  ?id ns1:branch ?branch .
}
```

6. Retrieve all names and their corresponding branches sorted by name:

sparql

```
PREFIX ns1: <http://example.org/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>


SELECT ?name ?branch
WHERE {
  ?id rdf:type rdf:Description .
  ?id ns1:name ?name .
  ?id ns1:branch ?branch .
}
ORDER BY ?name
```