

Neo4j vs. GraphDB

Unraveling the Key Differences in Graph Databases using RDF Data

Group 10

Aaron Yang, Aravinda Vijayaram Kumar,
Luhuan Wang

INTRODUCTION

□ Comparing fundamental characteristics of GraphDB and with Neo4j, particularly focusing on processing RDF data.

□ Why Comparison?

This research can guide decision-makers in making better choices when selecting a graph database for their specific needs.

INTRODUCTION

Ontotext GraphDB is a graph database management system that focuses on semantic data storage, supports semantic query and reasoning operations, and is high-performance, scalable and open.

The screenshot displays the Ontotext GraphDB web interface. On the left is a sidebar with navigation links: Import, Explore, SPARQL, Monitor, Setup, Lab, and Help. The main content area is titled 'View resource' and shows the 'FinalProject6102' repository. It includes a search bar for RDF resources, a table view toggle, and a 'Visual' view option. Below the search bar, the 'Active repository' section shows details for 'FinalProject6102', including the number of statements (1,771,586) and expansion ratio (1.42). To the right, the 'Saved SPARQL queries' section contains buttons for 'Add statements', 'Clear graph', 'Remove statements', and a 'SPARQL Select template'. At the bottom, the 'License' section shows the 'GraphDB Free Edition' license with details on licensing, validity, cores, maintenance, and capabilities.

GraphDB

Import Explore SPARQL Monitor Setup Lab Help

View resource

Search RDF resources... [Table](#) [Visual](#)

Hint: "abc" matches "abc", "ab c" and "a b c"

Active repository

Local

FinalProject6102

total statements
1,771,586

1,248,700 explicit
522,886 inferred
1.42 expansion ratio

[Import RDF data](#)
[Export RDF data](#)

Saved SPARQL queries

Add statements
PREFIX dc: <http://purl.org/dc/elements/1.1/> INSERT DATA { GRAPH <http://example> { <http://exam...

Clear graph
CLEAR GRAPH <http://example>

Remove statements
PREFIX dc: <http://purl.org/dc/elements/1.1/> DELETE DATA { GRAPH <http://example> { <http://exam...

SPARQL Select template
SELECT ?s ?p ?o WHERE { ?s ?p ?o . } LIMIT 100

License

GraphDB Free Edition

Licensed to	Valid until	Number of cores	Maintenance date	Capabilities
Freeware	Perpetual	1	Perpetual	Lucene connector

THE SOFTWARE IS PROVIDED 'AS IS', WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

[License settings](#)

What is RDF?

RDF stands for **Resource Description Framework**.

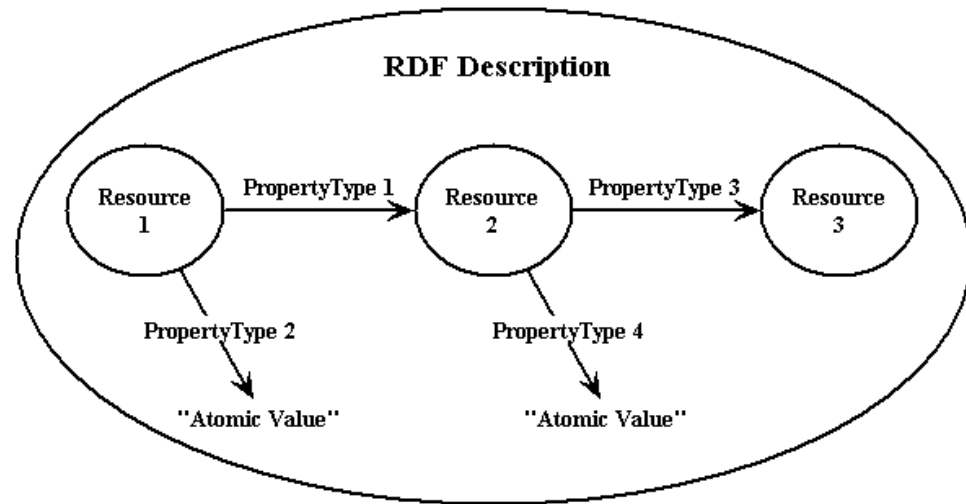
Key components:

Triples: RDF data is represented in triples, which consist of subject-predicate-object statements.

Resource Identification: RDF uses URIs (Uniform Resource Identifiers) to uniquely identify resources.

Graph-Based Structure: RDF data forms a graph structure where nodes represent resources, and edges (or arcs) between nodes represent the relationships or properties.

Extensibility: It allows for extensibility and flexibility in modeling relationships and describing data.



What is Semantic Web graphs?

A semantic web graph is the interconnected data structures that exist in the semantic web(a concept proposed by Tim Berners-Lee).

Standardized formats and protocols



Achieved by using technologies such as RDF (Resource Description Framework), OWL (Web Ontology Language), and SPARQL (SPARQL Protocol and RDF Query Language)

GraphDB Vs. Neo4j

DBs	GraphDB	Neo4j
Query Language	SPARQL	Cypher
Data Models	RDF	a property graph model
Licensing and Cost	Free and commercial versions	Community edition for free and commercial Enterprise editions with advanced features
Community	largely focused on semantic web and linked data	a large, active community with extensive support and resources

Dataset

- 2 datasets- ddw/ontology and The Medical Subject Headings (MeSH) RDF dataset
- The ddw/ontology dataset contains the core set of entities that data.world matches against and everything in this dataset is managed by data.world and is constantly being expanded and improved.
- The Medical Subject Headings (MeSH) RDF is a linked data representation of the MeSH biomedical vocabulary produced by the National Library of Medicine. It has a total of 15,439,946.


METHODOLOGY USED


- ❑ Connected GraphDB and Neo4j databases in Python through VS Code using the Sparqlwrapper and neo4j drivers.
- ❑ Executing our queries and a Windows Operating system with 16GB RAM
- ❑ Executed Queries and measured their execution times using the time library in python.


Neo4j Setup (5.11.0)

Example Project

+ Add ▾

 Movie DBMS 5.12.0

 Final Project 5.11.0


 Final Project2 5.11.0


Start


↗ Open ▾

⋮

This list of databases is cached, start the DBMS to refresh the list.

 system

 neo4j (default)

 FinalProject3 5.11.0

Details

Plugins

Upgrade

▸ APOC ✓ 5.11.0

▸ Graph Data Science Library ✓ 2.5.5

▸ Neo4j Streams

▸ Neosemantics (n10s) ✓ 5.11.0.0

THE GEORGE
WASHINGTON
UNIVERSITY
WASHINGTON, DC

```

##### 1. Loading data into the GraphDB
from SPARQLWrapper import SPARQLWrapper, POST,JSON
import time
import os

#Place the dataset folders in the same directory as the python file

path = os.getcwd()
path=path.replace("\\","/")
path=path[:0] + path[0].upper() + path[0 + 1:]

file_path1 = path+"/ddw-ontology-v0/original/entities/CanadianAdministrativeDistricts.ttl"
file_path2= path+"/ddw-ontology-v0/original/entities/UsAdministrativeDivisions.ttl"
file_path3= path+"/ddw-ontology-v0/original/entities/Currencies.ttl"
file_path4 = path+"/ddw-ontology-v0/original/entities/Countries.ttl"
file_path5 = path+"/ddw-ontology-v0/original/entities/Counties-Population.ttl"
file_path6 = path+"/ddw-ontology-v0/original/entities/NAICSCodes.ttl"
file_path7 = path+"/ddw-ontology-v0/original/entities/ZipCodes.ttl"
file_path8 = path+"/ddw-ontology-v0/original/entities/ICD10_PCS.ttl"
file_path9 = path+"/nlm-medical-subject-headings-mesh/original/mesh.nt"

sparql = SPARQLWrapper("http://DESKTOP-1KIJIK4:7200/repositories/mydb2/statements")
sparql.setMethod(POST)

```

Run Cell | Run Above | Debug Cell

Data Loading into Neo4j

```

from neo4j import GraphDatabase
import time

```

#neosemantics plugin has to be installed

#Please give absolute path of the files in the queries, when parameterized, the loading of

```

driver = GraphDatabase.driver("bolt://localhost:7687", auth=("neo4j", "12345678"))

```

Types of Queries Executed

☐ Tested the performances of both DBs with different operations-

The types of scenarios considered were

1. Loading the Data into the Databases
2. Basic querying the data using Select and Match
3. Traversals such as Depth First Search and Breadth First search, Best Path, Full Text Search, and Pattern Matching.
4. Deletion of the Data from the Databases.
5. Executing Traversal with ID vs URI property in Neo4J

Result1: Loading data into the Databases

GraphDB	Neo4J
LOAD <file:/// {file_path1} >	<ol style="list-style-type: none">1. CALL n10s.graphconfig.init (Neo4j graph configuration for handling RDF)2. CREATE CONSTRAINT n10s_unique_uri FOR (r:Resource) REQUIRE r.uri IS UNIQUE (creates a unique constraint in the Neo4j database)3. CALL n10s.rdf.import.fetch("file://--path--","Turtle");

Result1: Loading data into the Databases

```
### 1. Loading data into the GraphDB
from SPARQLWrapper import SPARQLWrapper, POST, JSON
import time
import os

#Place the dataset folders in the same directory as the python file

path = os.getcwd()
path=path.replace("\\", "/")
path=path[:0] + path[0].upper() + path[0 + 1:]

file_path1 = path+"/ddw-ontology-v0/original/entities/CanadianAdministrativeDistricts.ttl"
file_path2= path+"/ddw-ontology-v0/original/entities/UsAdministrativeDivisions.ttl"
file_path3= path+"/ddw-ontology-v0/original/entities/Currencies.ttl"
file_path4 = path+"/ddw-ontology-v0/original/entities/Countries.ttl"
file_path5 = path+"/ddw-ontology-v0/original/entities/Counties-Population.ttl"
file_path6 = path+"/ddw-ontology-v0/original/entities/NAICS Codes.ttl"
file_path7 = path+"/ddw-ontology-v0/original/entities/ZipCodes.ttl"
file_path8 = path+"/ddw-ontology-v0/original/entities/ICD10_PCS.ttl"
file_path9 = path+"/nlm-medical-subject-headings-mesh/original/mesh.nt"

sparql = SPARQLWrapper("http://DESKTOP-1KIJIK4:7200/repositories/mydb2/statements")
sparql.setMethod(POST)
```


Result1: Loading data into the Databases

```
# initializing the graph ----run only first time (if any error thrown pls comment or uncomment as required)
loadQuery = """
CALL n10s.graphconfig.init
"""
LoadResult = session.run(loadQuery)
print("Graph is initialized")

# Setting the unique uri constraint----run only first time (if any error thrown pls comment or uncomment as required)
loadQuery = """
CREATE CONSTRAINT n10s_unique_uri FOR (r:Resource) REQUIRE r.uri IS UNIQUE
"""
LoadResult = session.run(loadQuery)
print("uri constraint created")

# Loading the triplets

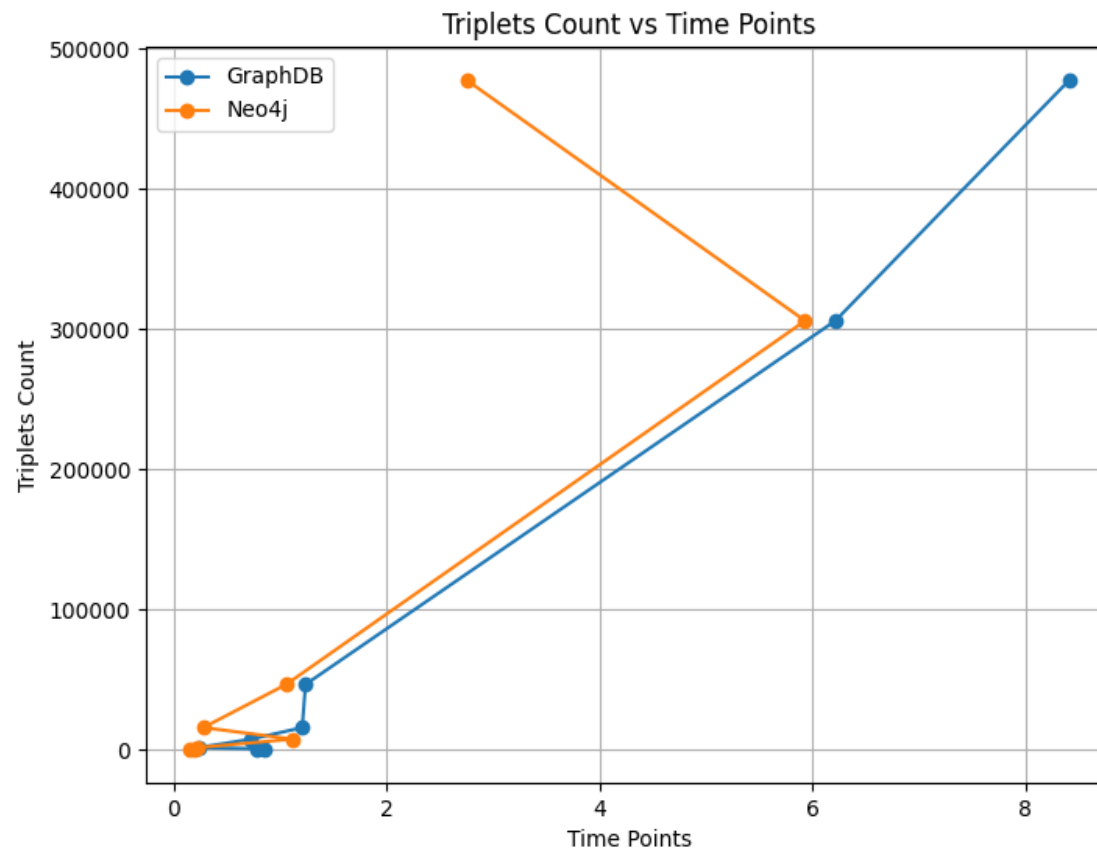
# 157 Triples
loadQuery =f"""
CALL n10s.rdf.import.fetch("file:///E:/GWU_1st_semester/Data_Warehousing/proj/ddw-ontology-v0/original/entities/CanadianAdministrativeDistricts.ttl",
"""
loadStartTime = time.time()
LoadResult = session.run(loadQuery).consume()
LoadEndTime = time.time()
n1=LoadEndTime - loadStartTime
```

Execution Performance-

Triplet Count	GraphDB	Neo4J
157	0.84797549247741 7 seconds	0.1910383701324 463 seconds
579	0.78243446350097 66 seconds	0.1515660285949 707 seconds
1433	0.23038530349731 445 seconds	0.2263948917388 916 seconds
7481	0.72723555564880 37 seconds	1.1153607368469 238 seconds

15705	1.2057037353515 625 seconds	0.27726054191589 355 seconds
46479	1.2367780208587 646 seconds	1.05511951446533 2 seconds
305779	6.2115886211395 26 seconds	5.92739105224609 4 seconds
477366	8.4084925651550 3	2.75075554847717 3 seconds
15439946	310.03265309333 8 seconds	250.409108161926 27 seconds

Performance Plot-



Result2: Basic Select/Match Operations

GraphDB	Neo4J
<pre>select * where { ?s ?p ?o . } limit 1000</pre>	<pre>MATCH (n) RETURN n LIMIT 1000</pre>

Result2: Basic Select/Match Operations

Run Cell | Run Above | Debug Cell

```
# %% Basic queries-
```

```
import time
```

```
sparql = SPARQLWrapper("http://DESKTOP-1KIJIK4:7200/repositories/mydb2")
```

```
sparql.setMethod(POST)
```

```
# 100
```

```
sparql.setQuery("""
```

```
    select * where {
```

```
    ?s ?p ?o .
```

```
} limit 100
```

```
""")
```

```
load_start_time = time.time()
```

```
sparql.setReturnFormat(JSON)
```

```
results = sparql.query().convert()
```

```
load_end_time = time.time()
```

```
t1=load_end_time - load_start_time
```

```
print("\n\nThe execution time to run a select query limit 100 in GraphDB is:", t1, "seconds\n")
```

Result2: Basic Select/Match Operations

```
## Basic Queries Neo4J

from neo4j import GraphDatabase
import time

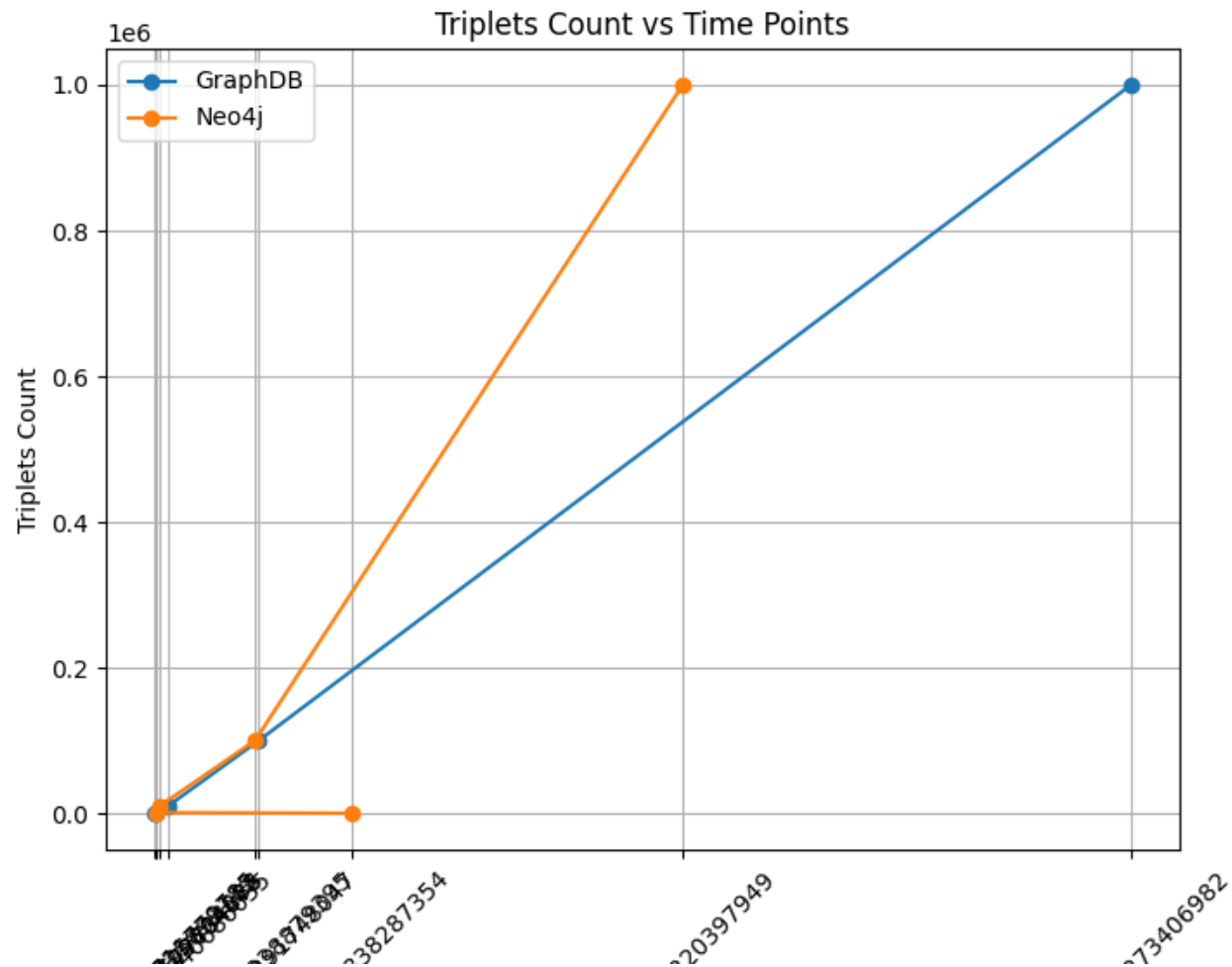
driver = GraphDatabase.driver("bolt://localhost:7687", auth=("neo4j", "12345678"))

with driver.session(database='neo4j') as session:

    #100 records
    loadQuery = """
        MATCH (n) RETURN n LIMIT 100
    """

    load_start_time = time.time()
    loadResult = session.run(loadQuery)
    load_end_time = time.time()
    n1=load_end_time - load_start_time
    print("\n\nThe execution time to run a match query limit 100 in Neo4J is:", n1, "seconds\n")
```

Basic Select	GraphDB	Neo4J
LIMIT 100	0.01995110511779785 seconds	2.0970842838287354 seconds
LIMIT 1000	0.03729748725891113 seconds	0.034908294677734375 seconds
LIMIT 10000	0.16156792640686035 seconds	0.07380294799804688 seconds
LIMIT 10000	1.1192455291748047 seconds	1.0791707038879395 seconds
LIMIT 1000000:	10.309787273406982 seconds	5.585696220397949 seconds



Result3: Data Deletion

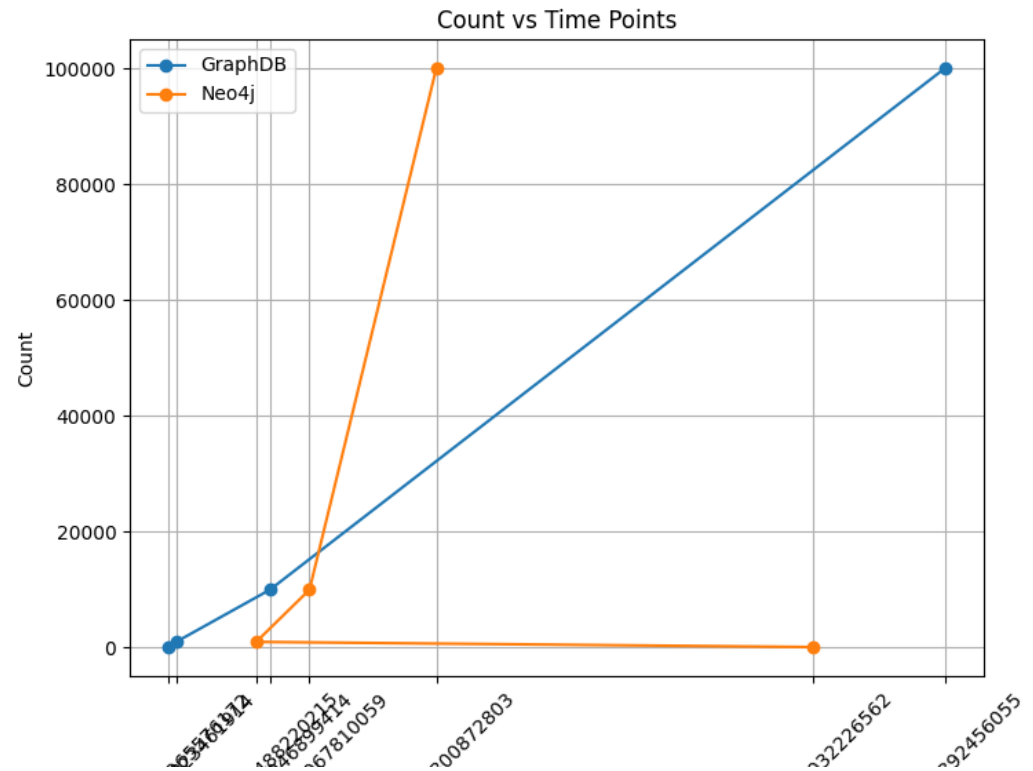
GraphDB	Neo4J
<pre>DELETE { ?subject ?predicate ?object . } WHERE { { SELECT * WHERE { ?subject ?predicate ?object . } LIMIT 100 } }</pre>	<pre>MATCH (n) WITH n LIMIT 100 DETACH DELETE n;</pre>

Basic Select	GraphDB	Neo4J
100 Triplets	0.1386 seconds	2.1626 seconds
1000 Triplets	0.1626 seconds	0.4136 seconds
10000 Triplets	0.4568 seconds	0.5808 seconds
1000000 Triplets	2.5747 seconds	0.9787 second

Results3: Data Deletion

Neo4j is relatively more stable as data grows,

GraphDB performs better than Neo4j on various query types and is also faster when dealing with small databases.



Result4: Basic Data traversal/Search

a. Breadth-First Traversal and Depth-First Traversal

PREFIX dwo: <https://ontology.data.world/v0#>

PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>

PREFIX skos: <http://www.w3.org/2004/02/skos/core#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?stateLabel ?geonameID

WHERE {

 ?state a dwo:UsState ;

 rdfs:label ?stateLabel ;

 geo:lat ?lat ;

 geo:long ?long ;

 dwo:geonameID ?geonameID .

}

DFT-

MATCH path = (startNode)-[*]-
>(endNode) WHERE endNode.uri
'https://entities.data.world/usstate_alabama'
' RETURN path;

BFT-

MATCH path = (startNode)-[*1..]->(endNode)
WHERE endNode.uri =
'https://entities.data.world/usstate_alabama'
RETURN path;

b. Best Path

```
PREFIX dwo: <https://ontology.data.world/v0#>
PREFIX                                     geo:
<http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>

SELECT ?state1Label ?state2Label

WHERE {
    ?state1 a dwo:UsState ;
            rdfs:label ?state1Label .

    ?state2 a dwo:UsState ;
            rdfs:label ?state2Label .

    FILTER(?state1 != ?state2)
}
```

```
MATCH path = shortestPath((startNode)-[*]-
(endNode))

WHERE startNode.uri =
'https://entities.data.world/usstate_alabama'

AND endNode.uri =
'https://en.wikipedia.org/wiki/Maine'

RETURN path;
```

c. Pattern Matching

PREFIX dwo: <https://ontology.data.world/v0#>

PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>

PREFIX skos: <http://www.w3.org/2004/02/skos/core#>

CONSTRUCT {

 ?state a dwo:UsState ;

 rdfs:label ?stateLabel ;

 geo:lat ?lat ;

 geo:long ?long ;

 dwo:geonameId ?geonameId .

} WHERE {

 ?state a dwo:UsState ;

 rdfs:label ?stateLabel ; geo:lat ?lat ;

 geo:long ?long ; dwo:geonameId ?geonameId .

}

MATCH (subject)-[*]->(object)

WHERE subject.uri =

'https://entities.data.world/usstate_colorado'

RETURN subject, object;

d. Full Text Search

PREFIX dwo:
<https://ontology.data.world/v0#>

PREFIX skos:
<http://www.w3.org/2004/02/skos/core#>

SELECT ?stateLabel

WHERE {

 ?state a dwo:UsState ;

 rdfs:label ?stateLabel .

 FILTER(CONTAINS(LCASE(?stateLabel),
"new"))

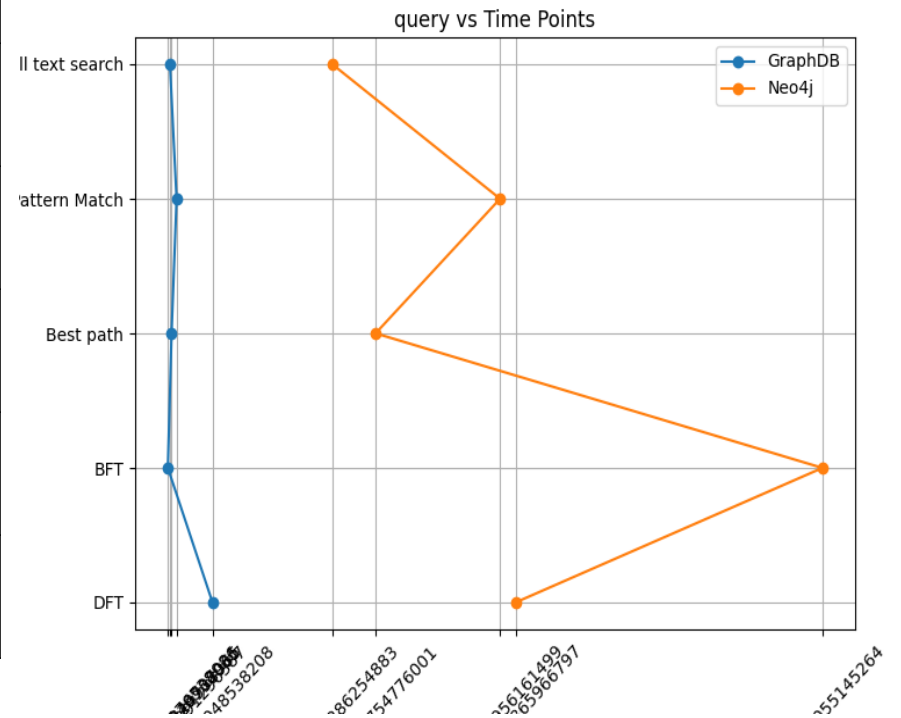
}

MATCH (node)

WHERE node.rdfs__label CONTAINS 'Colorado'

RETURN node

Query	GraphDB	Neo4J
DFT	0.282304048538208 seconds	2.1563243865966797 seconds
BFT	0.00589 seconds	4.054851055145264 seconds
BestPath	0.02892279624938965 seconds	1.290935754776001 seconds
Pattern Matching	0.05983996391296387 seconds	2.060992956161499 seconds
Full Text Search	0.01889944076538086 seconds	1.0228090286254883 seconds



Result5: Executing Traversal with ID vs URI property in Neo4J

```
MATCH path = (startNode)-[*]->(endNode)
```

```
WHERE ID(startNode) = 159
```

```
AND ID(endNode) = 13
```

```
RETURN path;
```

```
MATCH path = (startNode)-[*]->(endNode)
```

```
WHERE startNode.uri =  
'https://entities.data.world/county_bath_county_vir  
ginia'
```

```
AND endNode.uri =  
'https://entities.data.world/census_division_south_a  
tlantic'
```

```
RETURN path;
```

neo4j\$ MATCH path = (startNode)-[*]→(endNode) WHERE ID(startNode) = 159 AND ID(endNode) = 159

Graph

Table

Text

Warn

Code

```

path
{
  "start": {
    "identity": 159,
    "labels": [
      "Resource",
      "ns0__County"
    ],
    "properties": {
      "ns0__population2016": 4476,
      "ns0__population2015": 4502,
      "ns0__population2014": 4560,
      "skos__prefLabel": "Bath County, Virginia",
      "ns0__population2013": 4608,
      "ns0__geonameId": "4745776",
      "ns0__countyShortName": "Bath",
      "ns0__fips6_4": "51017",
    }
  }
}

```

Started streaming 1 records after 17 ms and completed after 34 ms.

neo4j\$

Graph

Table

Text

Warn

Code

```

1 MATCH path = (startNode)-[*]→(endNode)
2 WHERE startNode.uri = 'https://entities.data.world/county_bath_county_virginia'
3 AND endNode.uri = 'https://entities.data.world/census_division_south_atlantic'
4 RETURN path;

```

```

path
{
  "properties": {
    "ns0__population2016": 4476,
    "ns0__population2015": 4502,
    "ns0__population2014": 4560,
    "skos__prefLabel": "Bath County, Virginia",
    "ns0__population2013": 4608,
    "ns0__geonameId": "4745776",
    "ns0__countyShortName": "Bath",
    "ns0__fips6_4": "51017",
    "ns0__population2012": 4644,
    "uri": "https://entities.data.world/county_bath_county_virginia",
    "ns0__fips6_4County_Code": "017"
  },
  "elementId": "159"
},
{
  "end": {
    "identity": 159,
    "labels": [
      "Resource",
      "ns0__County"
    ],
    "properties": {
      "ns0__population2016": 4476,
      "ns0__population2015": 4502,
      "ns0__population2014": 4560,
      "skos__prefLabel": "Bath County, Virginia",
      "ns0__population2013": 4608,
      "ns0__geonameId": "4745776",
      "ns0__countyShortName": "Bath",
      "ns0__fips6_4": "51017",
    }
  }
}

```

Started streaming 1 records after 15 ms and completed after 785 ms.

the query is executed faster when ID is given rather than the URI

CONCLUSION

- Our thorough comparison of Ontotext GraphDB and Neo4j covering data loading, search queries, traversal algorithms, and data deletion has revealed subtle differences in the functionality and performance of these two well-known graph database systems.
- Even though it is a Property graph database, Neo4j showed great performance in handling RDF data, especially in loading, searching, and deletion. Neo4j performed better than GraphDB at handling higher data size.

The GraphDB excelled at traversals and searches, of RDF data outperforming the Neo4j.

- Neo4j is relatively more stable as data grows, while GraphDB performs better than Neo4j on various query types and is faster when dealing with small databases.

Thank you!

Group 10

Aaron Yang, Aravinda Vijayaram Kumar,
Luhuan Wang