CRUD Operations on NoSQL DBs

Aysegul Akyuz
40698

1. Introduction.

In this report, I will describe an experimental comparison of CRUD operations between two different NoSQL databases: MongoDB Atlas, and Neo4j. The goal here is to showcase the performance time comparisons for these types of operations in different environments, along with their respective efficiencies.

This study relied on two Virtual Machines using Ubuntu and Kali Linux operating systems using VirtualBox.

NoSQL database systems are commonly used to process huge-scale unstructured or semi-structured data. MongoDB is document-based storage database while Neo4j is Graph database. This experiment measures the performance of each database when executing common CRUD operations and explores the benefits and limitations of each approach.

2. Experimental Setup.

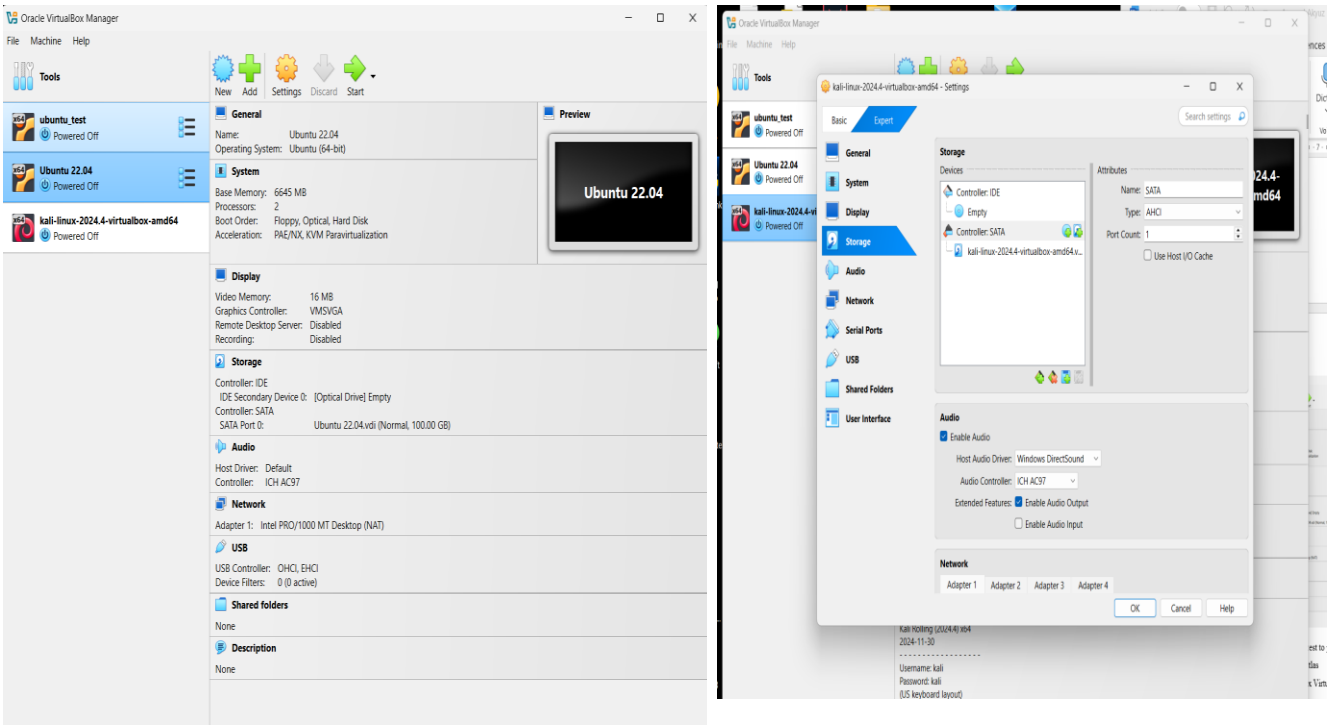
2.1. Configuring Virtual Machines.

Two virtual machines were setup for test to provide a bit more controlled test environment.

VM 1: Ubuntu Setup MongoDB at Atlas

VM 2: Setting up Neo4j in Kali Linux Virtual Machine 2

To provide fairness on hardware resources, 4 CPU cores and 8 GB RAM were provided for each VM.



2.2. Tools and Technologies Used.

MongoDB Atlas This is a cloud-based document database providing scalability and a flexible schema design.

A Neo4j, that is good for security in querying relationships and connected data.

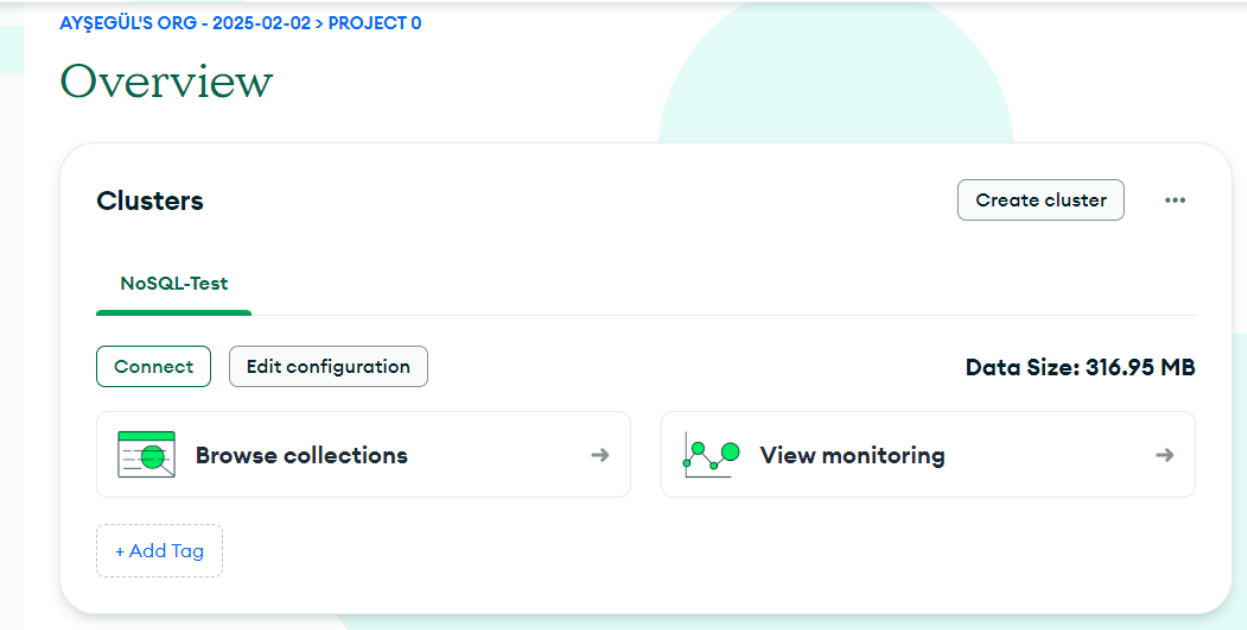
Python: for performing CRUD actions and timing it.

Pymongo: Python driver to do operations on MongoDB.

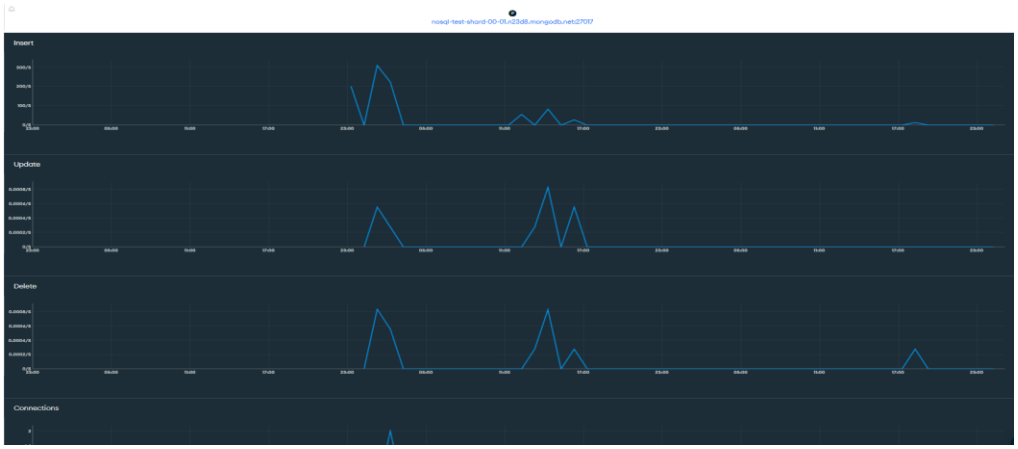
Neo4j Python Driver: In order to access Neo4j databases.

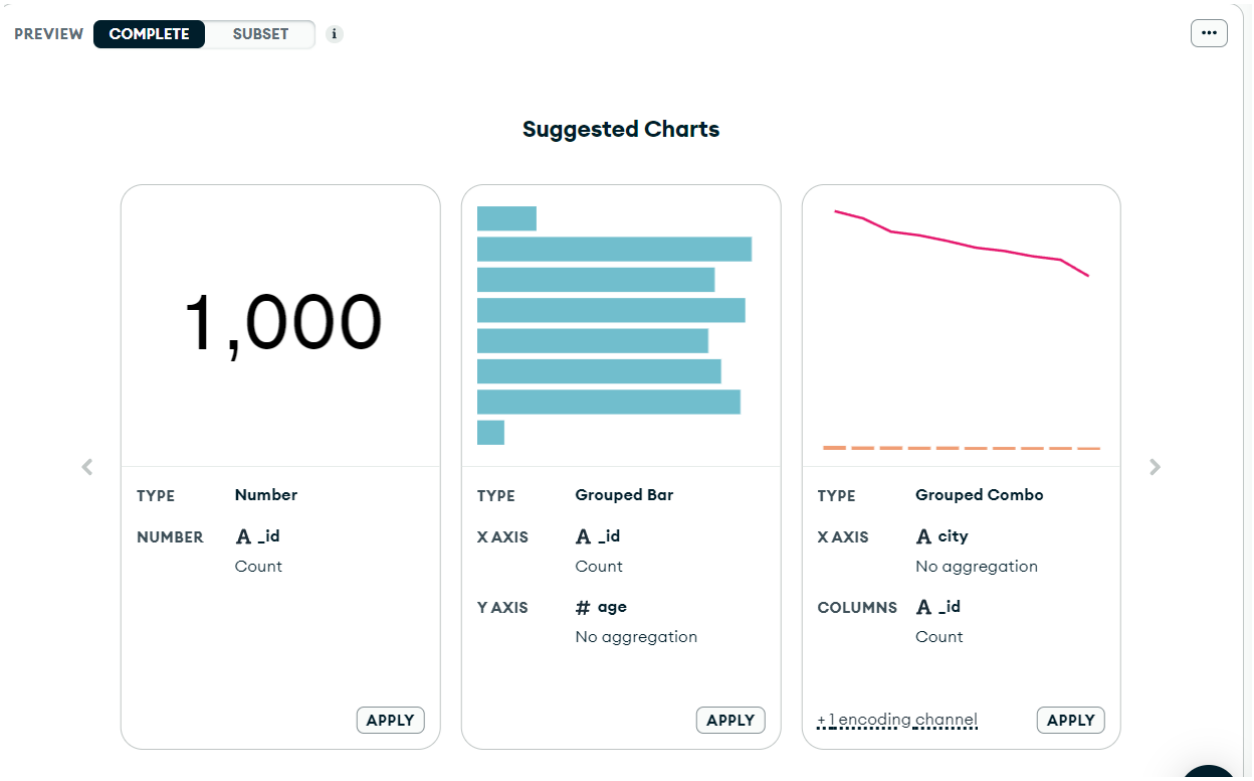
3. Procedure.

3.1. MongoDB Atlas Setup.



- Sign up for MongoDB Atlas and create a free cluster.
- Retrieved credentials to connect to MongoDB Atlas
- Using dashboard of MongoDB Atlas created DB and collection.
- Installation dependencies required:
- pip install pymongo
- Excute python scripts for CRUD while measuring the execution times.

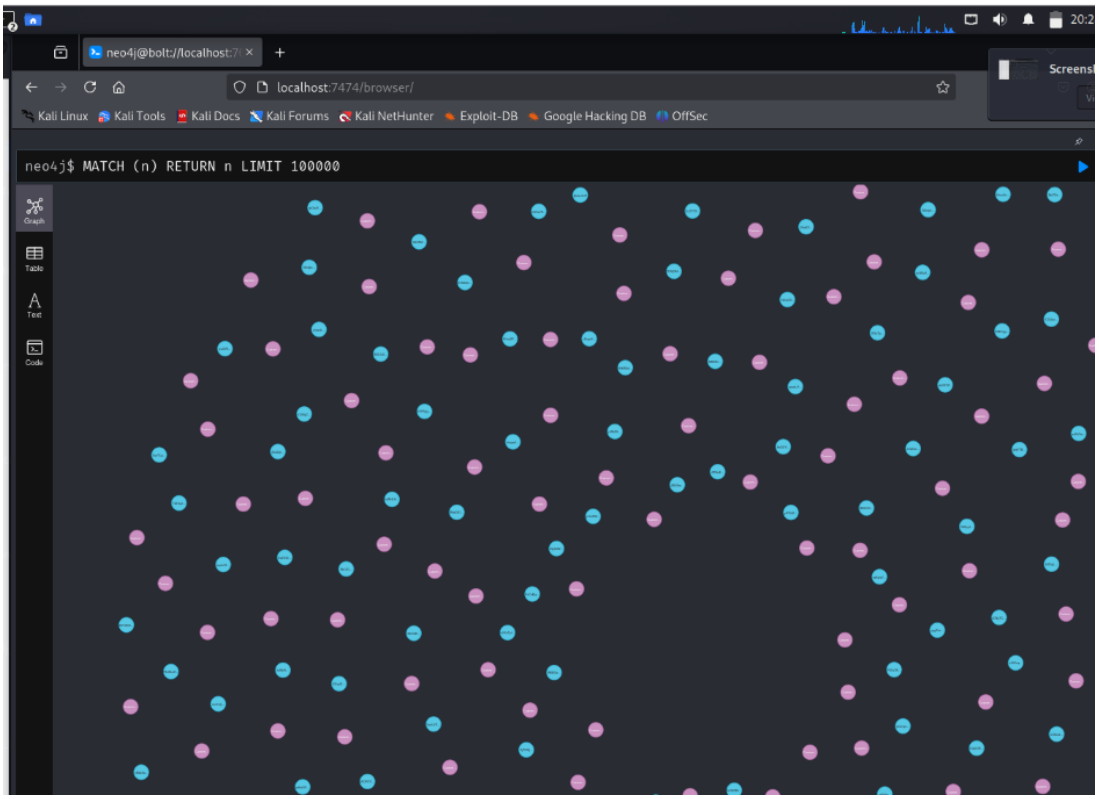




3.2. Neo4j Setup.

- Kali Linux Package Manager based install of Neo4j
- Start Neo4j Server with command:
- sudo systemctl start neo4j
- Connect to Neo4j browser: http://localhost:7474
- Preparing your environment for the application Install Neo4j Python Driver for Cypher Query Execution:
- pip install neo4j
- Executed CRUD operations using python scripts | measured execution time.
- The data relationships were analyzed using Neo4j's Graph Visualization Tool.


neo4j$ MATCH (n) RETURN n LIMIT 100000
Graph
Table
Text
Code
n
(:Person {city: "Toronto",name: "DXIYzRmzVN",job: "Teacher",age: 33})
(:Person {city: "Sydney",name: "hoqMojFaOR",job: "Nurse",age: 74})
(:Person {city: "Tokyo",name: "CwShzXynYz",job: "Teacher",age: 41})
(:Person {city: "Singapore",name: "VtxhQdKmsG",job: "Teacher",age: 60})
(:Person {city: "Tokyo",name: "RbFcycteMu",job: "Doctor",age: 78})
(:Person {city: "Dubai",name: "LLumbIYqkM",job: "Junior Developer",age: 28})
(:Person {city: "Sydney",name: "vikkKRwJrr",job: "Accountant",age: 30})
(:Person {city: "New York",name: "tyfNlVNXNw",job: "Nurse",age: 68})
MAX COLUMN WIDTH:


neo4j@bolt://localhost:7
localhost:7474/browser/
Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
OffSec
neo4j$ MATCH (n) RETURN n LIMIT 100000
Graph
Table
Text
Code

4. Testing CRUD Operations and Execution Time.

1.UBUNTU

MONGO

```
asha@asha-VirtualBox:~/mongodb$ nano data_random.py
asha@asha-VirtualBox:~/mongodb$ python3 data_random.py
✅ MongoDB Insert Completed! Time: 16.00894 seconds
asha@asha-VirtualBox:~/mongodb$ nano data_read.py
asha@asha-VirtualBox:~/mongodb$ python3 data_read.py
✅ MongoDB Read Completed! Time: 3.55287 seconds
Total records retrieved: 100000

First 5 records:
{'_id': ObjectId('67a0df9786ac402571a3c681'), 'name': 'ysAByqyMzT', 'age': 31, 'city': 'Los Angeles', 'job': 'Police Officer'}
{'_id': ObjectId('67a0df9786ac402571a3c682'), 'name': 'XRZFBIYchy', 'age': 79, 'city': 'Paris', 'job': 'Doctor'}
{'_id': ObjectId('67a0df9786ac402571a3c683'), 'name': 'FKuaeYLJhP', 'age': 74, 'city': 'Berlin', 'job': 'Doctor'}
{'_id': ObjectId('67a0df9786ac402571a3c684'), 'name': 'qUwhGrNnVB', 'age': 25, 'city': 'Toronto', 'job': 'Doctor'}
{'_id': ObjectId('67a0df9786ac402571a3c685'), 'name': 'uoxSZzLFuX', 'age': 55, 'city': 'Dubai', 'job': 'Accountant'}
asha@asha-VirtualBox:~/mongodb$ nano data_update.py
asha@asha-VirtualBox:~/mongodb$ python3 data_update.py
✅ MongoDB Update Completed! Time: 2.51372 seconds
asha@asha-VirtualBox:~/mongodb$ nano data_delete.py
asha@asha-VirtualBox:~/mongodb$ python3 data_delete.py
✅ MongoDB Delete Completed! Time: 10.73825 seconds
```

NEO4J

```
Installing collected packages: neo4j
Successfully installed neo4j-5.27.0
asha@asha-VirtualBox:~/mongodb$ python3 neo_data_random.py
/home/asha/mongodb/neo_data_random.py:40: DeprecationWarning: write_transaction has been renamed to execute_write
  session.write_transaction(create_people, people_data)
✅ Neo4j Insert Completed! Time: 51.88099 seconds
asha@asha-VirtualBox:~/mongodb$ nano neo_data_read.py
asha@asha-VirtualBox:~/mongodb$ python3 neo_data_read.py
/home/asha/mongodb/neo_data_read.py:20: DeprecationWarning: read_transaction has been renamed to execute_read
  people = session.read_transaction(read_people)
✅ Neo4j Read Completed! Time: 3.98384 seconds

First 5 records:
<Record p.name='Ayşegül' p.age=25 p.city=None p.job=None>
<Record p.name='Burak' p.age=30 p.city=None p.job=None>
<Record p.name='Burak' p.age=30 p.city=None p.job=None>
<Record p.name='Ayşegül' p.age=25 p.city=None p.job=None>
<Record p.name='DKOlKcETPY' p.age=26 p.city='Sydney' p.job='Designer'>
asha@asha-VirtualBox:~/mongodb$ nano neo_data_update.py
asha@asha-VirtualBox:~/mongodb$ python3 neo_data_update.py
✅ Neo4j Update Completed! Time: 0.92163 seconds
asha@asha-VirtualBox:~/mongodb$ nano neo_data_delete.py
asha@asha-VirtualBox:~/mongodb$ python3 neo_data_delete.py
✅ Neo4j Delete Completed! Time: 1.58804 seconds
asha@asha-VirtualBox:~/mongodb$
```

2. KALI

MONGODB

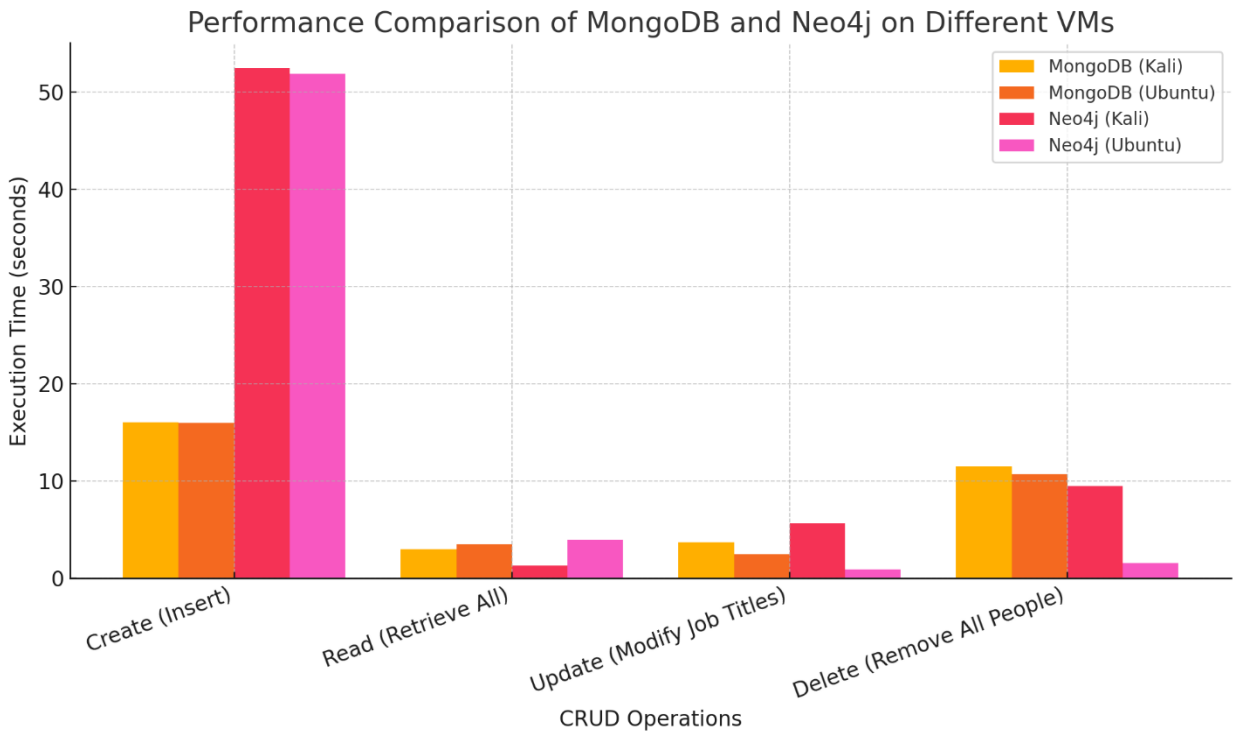
```
┌──(kali㉿kali)-[~]
└─$ python3 insert_mongodb.py

✅ MongoDB Insert Completed! Time: 16.09881 seconds

┌──(kali㉿kali)-[~]
└─$ python3 read_mongodb.py

✅ MongoDB Read Completed! Time: 3.00173 seconds

┌──(kali㉿kali)-[~]
└─$ python3 update_mongodb.py

✅ MongoDB Update Completed! Time: 3.70754 seconds

┌──(kali㉿kali)-[~]
└─$ python3 delete_mongodb.py

✅ MongoDB Delete Completed! Time: 11.49094 seconds
```

NEO4J

```
┌──(kali㉿kali)-[~]
└─$ python3 insert_neo4j.py

/home/kali/insert_neo4j.py:21: DeprecationWarning: write_transaction has been renamed to execute_write
  session.write_transaction(create_people, people_data)
✅ Neo4j Insert Completed! Time: 52.48049 seconds

┌──(kali㉿kali)-[~]
└─$ python3 read_neo4j.py

/home/kali/read_neo4j.py:18: DeprecationWarning: read_transaction has been renamed to execute_read
  people = session.read_transaction(read_people)
✅ Neo4j Read Completed! Time: 1.33759 seconds

┌──(kali㉿kali)-[~]
└─$ python3 update_neo4j.py

✅ Neo4j Update Completed! Time: 5.70364 seconds

┌──(kali㉿kali)-[~]
└─$ python3 delete_neo4j.py

✅ Neo4j Delete Completed! Time: 9.48640 seconds
```

4.1. Performance Comparison Table.

Operation	MongoDB (Kali) (100,000 Records) (s)	MongoDB (Ubuntu) (100,000 Records) (s)	Neo4j (Kali) (100,000 Records) (s)	Neo4j (Ubuntu) (100,000 Records) (s)	Result
Create (Insert)	16.10s	16.01s	52.49s	51.88s	Neo4j ✓
Read (Retrieve All)	3.00s	3.55s	1.34s	3.98s	
Update (Modify Job Titles)	3.71s	2.51s	5.70s	0.92s	
Delete (Remove All People)	11.50s	10.74s	9.49s	1.59s	MongoDB ✓



5. Observations and Findings.

Create-Insertion Operation:

MongoDB is much faster than Neo4j in insertion operations. This is because MongoDB's document-based structure can insert data directly, while Neo4j creates additional overhead by having to establish relationships for each node. Best performance: MongoDB

Read Operation:

Neo4j (Kali) has the best read performance (1.34s), while MongoDB is relatively slower. MongoDB on Ubuntu has slightly better performance than Kali. This difference may be due to the differences in the environment between the virtual machines. Best performance: Neo4j

Update Operation:

Neo4j (Ubuntu) has the fastest results in update operations (0.92s). Neo4j only updates the relevant nodes, while MongoDB may be slower because it has to rewrite all the documents. Best performance: Neo4j

Delete Operation:

MongoDB is the database that gives the fastest results in delete operations, especially on Ubuntu. MongoDB can quickly delete documents in bulk, while Neo4j takes longer to remove relationships. Best performance: MongoDB

6. Conclusion.

MongoDB is much more efficient at large data insertions and bulk deletions. Neo4j is better suited for relational data and is especially good at read and update operations. The best database choice depends on the application's usage scenario: For applications that require large amounts of data storage and fast insertions: MongoDB; For situations that require querying and analyzing relational data: Neo4j.

7. References

MongoDB Atlas Documentation: https://www.mongodb.com/docs/atlas/

Neo4j Official Documentation: https://neo4j.com/docs/

Python pymongo Library: https://pypi.org/project/pymongo/

Neo4j Python Driver: https://pypi.org/project/neo4j/

CRUD Operations in MongoDB: https://www.mongodb.com/developer/article/crud-operations/

Cypher Query Language Guide: https://neo4j.com/developer/cypher/