

# Проектування високонавантажених систем

## Лабораторна робота №3

### Робота з базовими функціями граф-орієнтованої БД на прикладі Neo4j

Мартиненко Денис ФБ-42мп

#### Встановлення Neo4j за допомогою docker-compose.yml

```
docker-compose.yml X
docker-compose.yml
1  services:
2    neo4j:
3      image: neo4j
4      ports:
5        - "7474:7474"
6        - "7687:7687"
7      volumes:
8        - ./neo4j/data:/data
9        - ./neo4j/logs:/logs
10     environment:
11       - NEO4J_AUTH=neo4j/superadmin
12       - NEO4J_ACCEPT_LICENSE_AGREEMENT=yes
13       - NEO4JLABS_PLUGINS=["graph-data-science", "apoc"]
14   volumes:
15     neo4j:
16       driver: local

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SERIAL MONITOR

PS C:\Users\denis\OneDrive\Рабочий стол\dataintensive_lab3> docker-compose up
[+] Running 1/1
✓ Container dataintensive_lab3-neo4j-1 Recreated
Attaching to neo4j-1
neo4j-1 | NEO4JLABS_PLUGINS has been renamed to NEO4J_PLUGINS since Neo4j 5.0.0.
neo4j-1 | The old name will still work, but is likely to be deprecated in future releases.
neo4j-1 | Fetching versions.json for Plugin 'graph-data-science' from https://graphdata
neo4j-1 | Installing Plugin 'graph-data-science' from https://graphdatascience.ninja/ne
[]
```

← → ↻ 🌐 localhost:7474/browser/

📧 Gmail 📺 YouTube 🎵 SoundCloud 📍 Карты 🗨️ Перевести 🇺🇦 ukr.net 🛒 EKatolog 🟢 Rozetka 🇵🇹 Komai

\$

Database access not available. Please use `:server connect` to establish connection. There's a graph waiting for you.

\$ :server connect

### Connect to Neo4j

Database access might require an authenticated connection

Connect URL

neo4j://

Pick neo4j:// for a routed connection (Aura, Cluster), bolt:// for a direct connection to a single instance.

Database - leave empty for default

Authentication type

Username / Password

Username

Password

## Завдання:

Змодельувати наступну предметну область:

- Є: Items, Customers, Orders

```
CREATE (customer1:Customer {name: 'Denys'})
CREATE (customer2:Customer {name: 'Maksym'})
CREATE (customer3:Customer {name: 'Oleh'})
CREATE (item1:Item {name: 'Item1', price: 88})
CREATE (item2:Item {name: 'Item2', price: 121})
CREATE (item3:Item {name: 'Item3', price: 55})
CREATE (item4:Item {name: 'Item4', price: 23})
CREATE (order1:Order {number: 'Order1'})
CREATE (order2:Order {number: 'Order2'})
CREATE (order3:Order {number: 'Order3'})
CREATE (order4:Order {number: 'Order4'})
```

neo4j\$ MATCH (n:Customer) RETURN n LIMIT 25

Graph

Table

Text

Code

n

{:Customer (name: "Denys")}

{:Customer (name: "Maksym")}

{:Customer (name: "Oleh")}

neo4j\$ MATCH (n:Item) RETURN n LIMIT 25

Graph

Table

Text

Code

n

{:Item (price: 88,name: "Item1")}

{:Item (price: 121,name: "Item2")}

{:Item (price: 55,name: "Item3")}

{:Item (price: 23,name: "Item4")}

neo4j\$ MATCH (n:Order) RETURN n LIMIT 25

Graph

Table

Text

Code

n

{:Order (number: "Order1")}

{:Order (number: "Order2")}

{:Order (number: "Order3")}

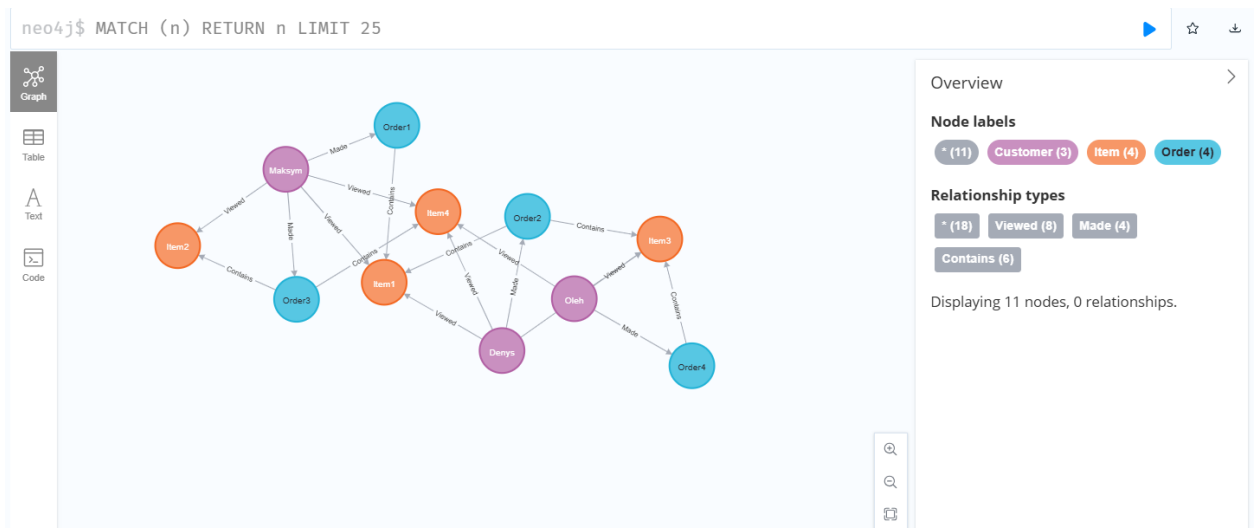
{:Order (number: "Order4")}

- Customer може додати Item(s) до Order (тобто купити Товар)
- У Customer може бути багато Orders
- Item може входити в багато Orders, і у Item є вартість
- Customer може переглядати (view), але при цьому не купувати Items

```
MATCH (customer1:Customer {name: 'Denys'}), (item1:Item {name: 'Item1'}), (item3:Item {name: 'Item3'}), (item4:Item {name: 'Item4'}), (order2:Order {number: 'Order2'})
CREATE (customer1)-[:Viewed]->(item1), (customer1)-[:Viewed]->(item3), (customer1)-[:Viewed]->(item4), (customer1)-[:Made]->(order2), (order2)-[:Contains]->(item1), (order2)-[:Contains]->(item3)
```

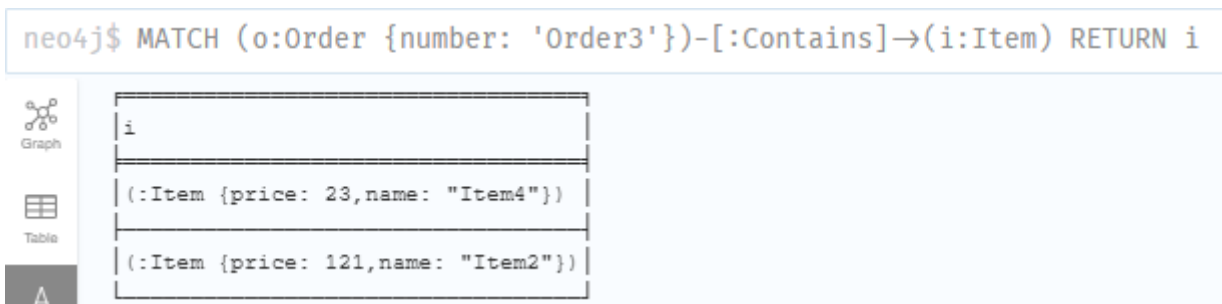
```
MATCH (customer2:Customer {name: 'Maksym'}), (item1:Item {name: 'Item1'}), (item2:Item {name: 'Item2'}), (item4:Item {name: 'Item4'}), (order1:Order {number: 'Order1'}), (order3:Order {number: 'Order3'})
CREATE (customer2)-[:Viewed]->(item1), (customer2)-[:Viewed]->(item2), (customer2)-[:Viewed]->(item4), (customer2)-[:Made]->(order1), (order1)-[:Contains]->(item1), (customer2)-[:Made]->(order3), (order3)-[:Contains]->(item2), (order3)-[:Contains]->(item4)
```

```
MATCH (customer3:Customer {name: 'Oleh'}), (item3:Item {name: 'Item3'}), (item4:Item {name: 'Item4'}), (order4:Order {number: 'Order4'})
CREATE (customer3)-[:Viewed]->(item3), (customer3)-[:Viewed]->(item4), (customer3)-[:Made]->(order4), (order4)-[:Contains]->(item3)
```

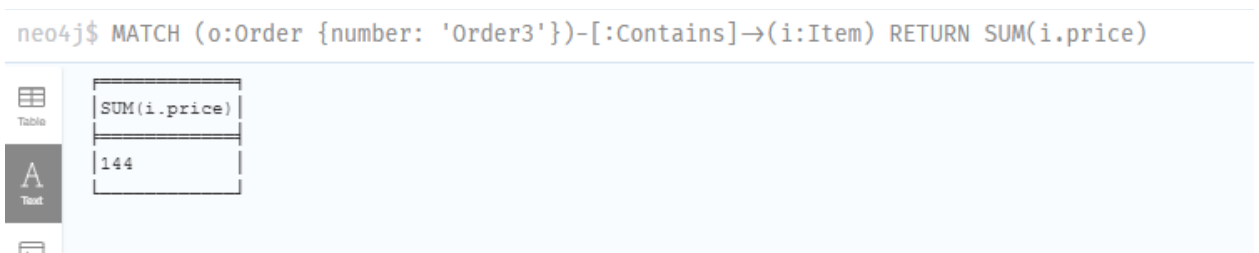


## 1. Написати наступні види запитів:

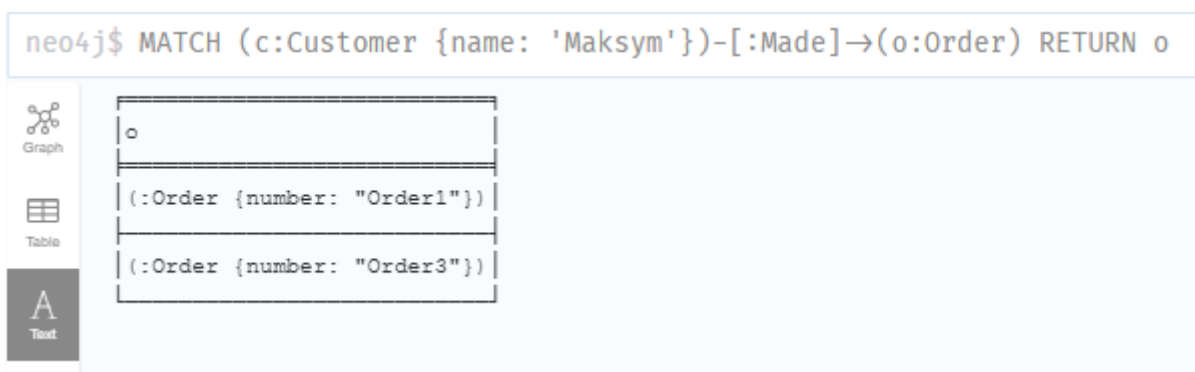
- Знайти Items які входять в конкретний Order



- Підрахувати вартість конкретного Order



- Знайти всі Orders конкретного Customer



- Знайти всі Items куплені конкретним Customer (через Order)

```
neo4j$ MATCH (c:Customer {name: 'Maksym'})-[:Made]→(o:Order)-[:Contains]→(i:Item) RETURN i
```

i
(:Item {price: 88,name: "Item1"})
(:Item {price: 23,name: "Item4"})
(:Item {price: 121,name: "Item2"})

- Знайти кількість Items куплені конкретним Customer (через Order)

```
neo4j$ MATCH (c:Customer{name: 'Maksym'})-[:Made]→(o:Order)-[:Contains]→(i:Item) RETURN COUNT(i)
```

COUNT(i)
3

- Знайти для Customer на яку суму він придбав товарів (через Order)

```
neo4j$ MATCH (c:Customer{name: 'Maksym'})-[:Made]→(o:Order)-[:Contains]→(i:Item) RETURN SUM(i.price)
```

SUM(i.price)
232

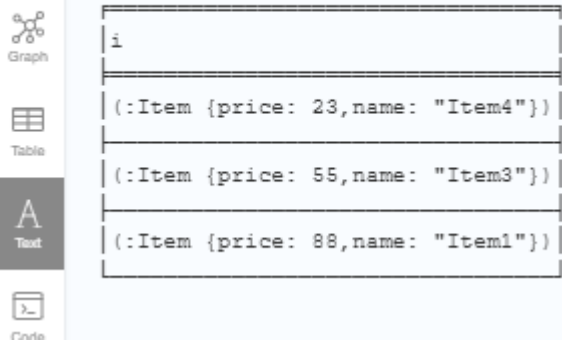
- Знайти скільки разів кожен товар був придбаний, відсортувати за цим значенням

```
neo4j$ MATCH (o:Order)-[:Contains]→(i:Item) RETURN i, COUNT(i) as n ORDER BY n DESC
```

i	n
(:Item {price: 88,name: "Item1"})	2
(:Item {price: 55,name: "Item3"})	2
(:Item {price: 121,name: "Item2"})	1
(:Item {price: 23,name: "Item4"})	1

- Знайти всі Items переглянуті (view) конкретним Customer

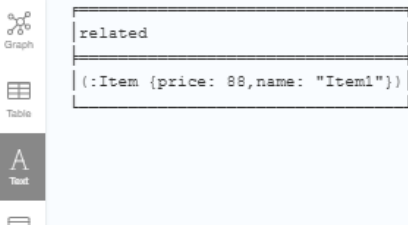
```
neo4j$ MATCH (c:Customer{name: 'Denys'})-[:Viewed]→(i:Item) RETURN i
```



i
(:Item {price: 23, name: "Item4"})
(:Item {price: 55, name: "Item3"})
(:Item {price: 88, name: "Item1"})

- Знайти інші Items що купувались разом з конкретним Item (тобто всі Items що входять до Order-s разом з даними Item)

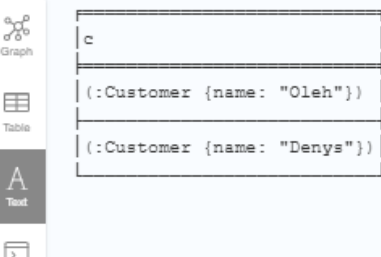
```
neo4j$ MATCH (i:Item {name: 'Item3'})←[:Contains]-(o:Order)-[:Contains]→(related:Item) RETURN related
```



related
(:Item {price: 88, name: "Item1"})

- Знайти Customers які купили даний конкретний Item

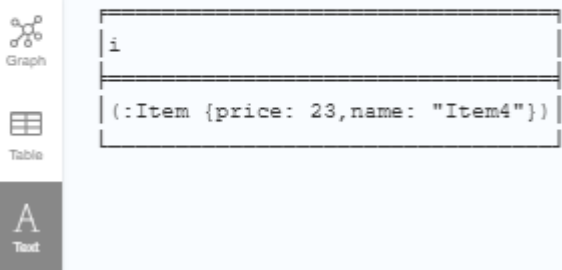
```
neo4j$ MATCH (i:Item {name: 'Item3'})←[:Contains]-(o:Order)←[:Made]-(c:Customer) RETURN c
```



c
(:Customer {name: "Oleh"})
(:Customer {name: "Denys"})

- Знайти для певного Customer(a) товари, які він переглядав, але не купив

```
1 MATCH (c:Customer{name: 'Oleh'})-[:Viewed]→(i:Item)
2 WHERE NOT (c)-[:Made]→(:Order)-[:Contains]→(i)
3 RETURN i
```



i
(:Item {price: 23, name: "Item4"})

## 2. Як і в попередніх завданнях, для якогось одного обраного Item додайте поле з кількістю його лайків.

Створюємо вузол для Item із лічильником лайків, інкрементуємо лічильник та створюємо конкуренцію:

The image shows the Neo4j interface. On the left, a Cypher query is entered: `neo4j$ MATCH (n:Item) RETURN n LIMIT 25`. Below the query, a table of results is shown with columns `n` and `likes`. The results are: `{:Item (price: 88, name: "Item1")}`, `{:Item (price: 121, name: "Item2")}`, `{:Item (price: 66, name: "Item3")}`, `{:Item (price: 23, name: "Item4")}`, and `{:Item (name: "Item1", likes: 0)}`. On the right, another Cypher query is entered: `1 MATCH (item:Item {name: 'Item1'})`, `2 SET item.likes = item.likes + 1`, `3 RETURN item.likes`, `4`. Below this query, a table view shows the results for `item.likes` with values `null` and `2`.

- З 10 окремих клієнтів одночасно запустити інкрементацію каунтеру лайків по 10\_000 на кожного клієнта

```
# Запустимо 10 клієнтів, кожен збільшує лічильник 10_000 разів
with ThreadPoolExecutor(max_workers=10) as executor:
    futures = [executor.submit(increment_likes, uri, user, password, 10000) for _ in range(10)]
```

- зробить так щоб не було втрат та перевірте щоб фінальне значення було 100\_000

The image shows the Neo4j interface. On the left, a Cypher query is entered: `1 MATCH (item:Item {name: 'Item1'})`, `2 RETURN item.likes`, `3`. Below the query, a table view shows the results for `item.likes` with values `null` and `100000`.

- заміряйте час роботи

```
Інкрементація завершена!
Час виконання: 646.7097592353821 секунд
PS C:\Users\denis\OneDrive\Рабочий стол\dataintensive_lab3>
```

## Код:

```
from neo4j import GraphDatabase
from concurrent.futures import ThreadPoolExecutor
import time

def increment_likes(uri, user, password, count):
    driver = GraphDatabase.driver(uri, auth=(user, password))

    def increment(tx):
        tx.run("MATCH (item:Item {name: 'Item1'}) SET item.likes = item.likes + 1")

    with driver.session() as session:
        for _ in range(count):
            session.write_transaction(increment)

    driver.close()

if __name__ == "__main__":
    uri = "bolt://localhost:7687"
    user = "neo4j"
    password = "superadmin"

    start_time = time.time()

    # Запустимо 10 клієнтів, кожен збільшує лічильник 10_000 разів
    with ThreadPoolExecutor(max_workers=10) as executor:
        futures = [executor.submit(increment_likes, uri, user, password, 10000)
                    for _ in range(10)]

    end_time = time.time()
    print("Інкрементація завершена!")
    print(f"Час виконання: {end_time - start_time} секунд")
```