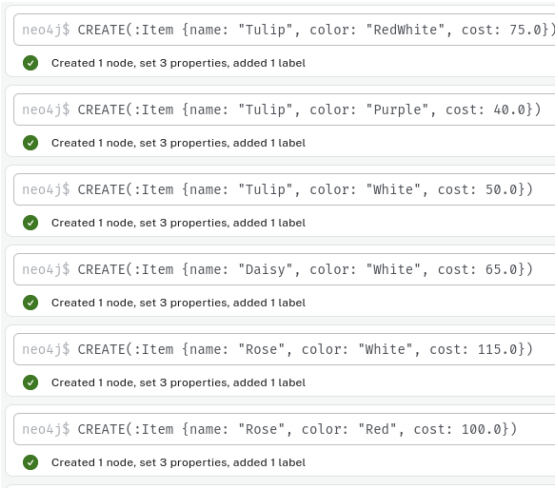
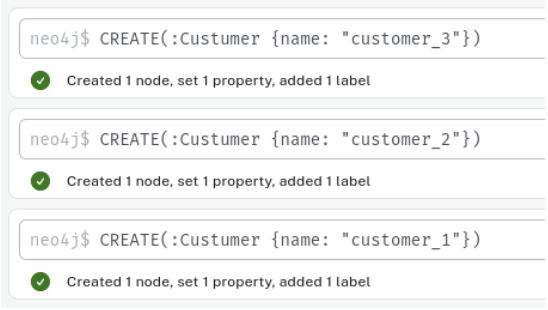
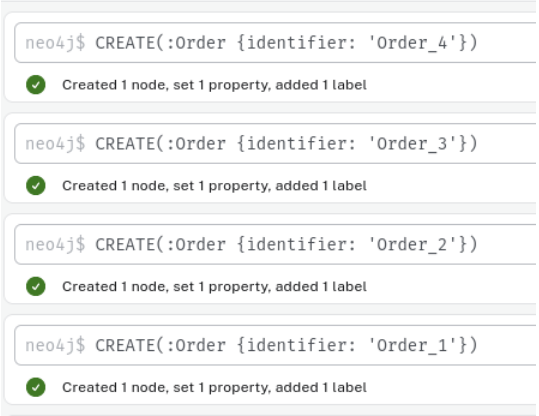
Завдання:

Змоделювати наступну предметну область:

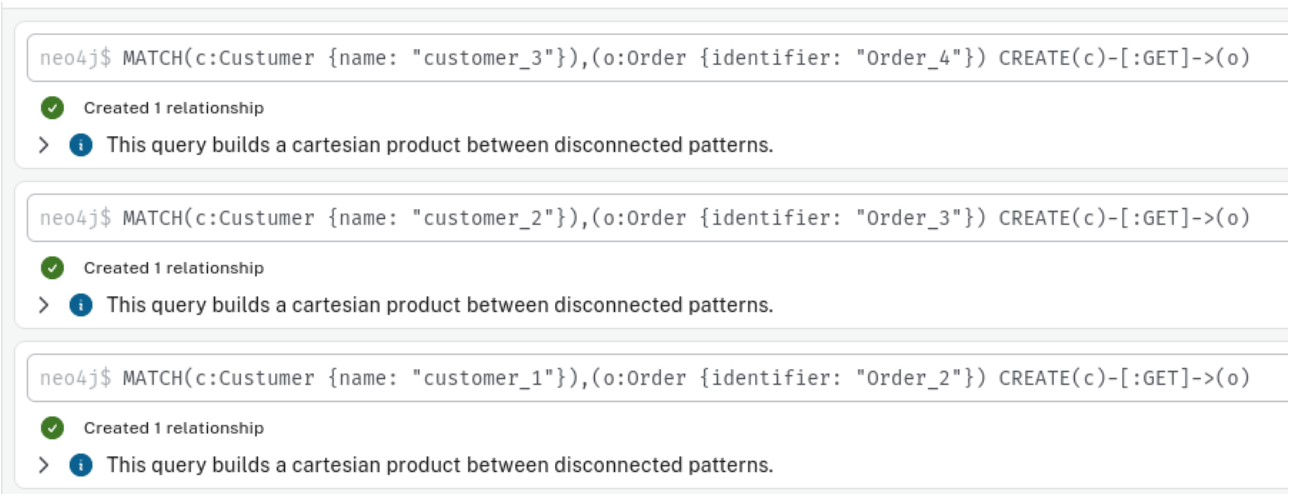
- Є: Items, Customers, Orders







- Customer може додати Item(s) до Order (тобто купити Товар)

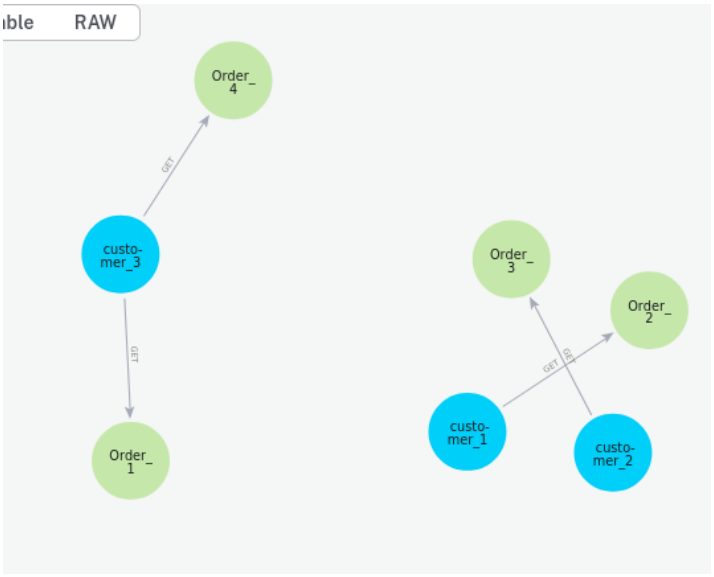


- У Customer може бути багато Orders

```
neo4j$ MATCH(c:Custumer {name: "customer_3"}),(o:Order {identifier: "Order_1"}) CREATE(c)-[:GET]->(o)
```

Created 1 relationship

This query builds a cartesian product between disconnected patterns.



- Item може входити в багато Orders, і у Item є вартість

```
neo4j$ MATCH(o:Order {identifier: "Order_3"}),(i:Item {cost: 75.0, color: "RedWhite", name: "Tulip"}) CREATE (o)-[:CONTAINS]-
```

Created 1 relationship — Completed after 58 ms

This query builds a cartesian product between disconnected patterns.

```
neo4j$ MATCH(o:Order {identifier: "Order_3"}),(i:Item {cost: 40.0, color: "Purple", name: "Tulip"}) CREATE (o)-[:CONTAINS]->(
```

Created 1 relationship — Completed after 18 ms

This query builds a cartesian product between disconnected patterns.

```
neo4j$ MATCH(o:Order {identifier: "Order_3"}),(i:Item {cost: 50.0, color: "White", name: "Tulip"}) CREATE (o)-[:CONTAINS]->(i
```

Created 1 relationship — Completed after 18 ms

This query builds a cartesian product between disconnected patterns.

```
neo4j$ MATCH(o:Order {identifier: "Order_2"}),(i:Item {cost: 115.0, color: "White", name: "Rose"}) CREATE (o)-[:CONTAINS]->(i
```

Created 1 relationship — Completed after 18 ms

This query builds a cartesian product between disconnected patterns.

```
neo4j$ MATCH(o:Order {identifier: "Order_1"}),(i:Item {cost: 65.0, color: "White", name: "Daisy"}) CREATE (o)-[:CONTAINS]->(i
```

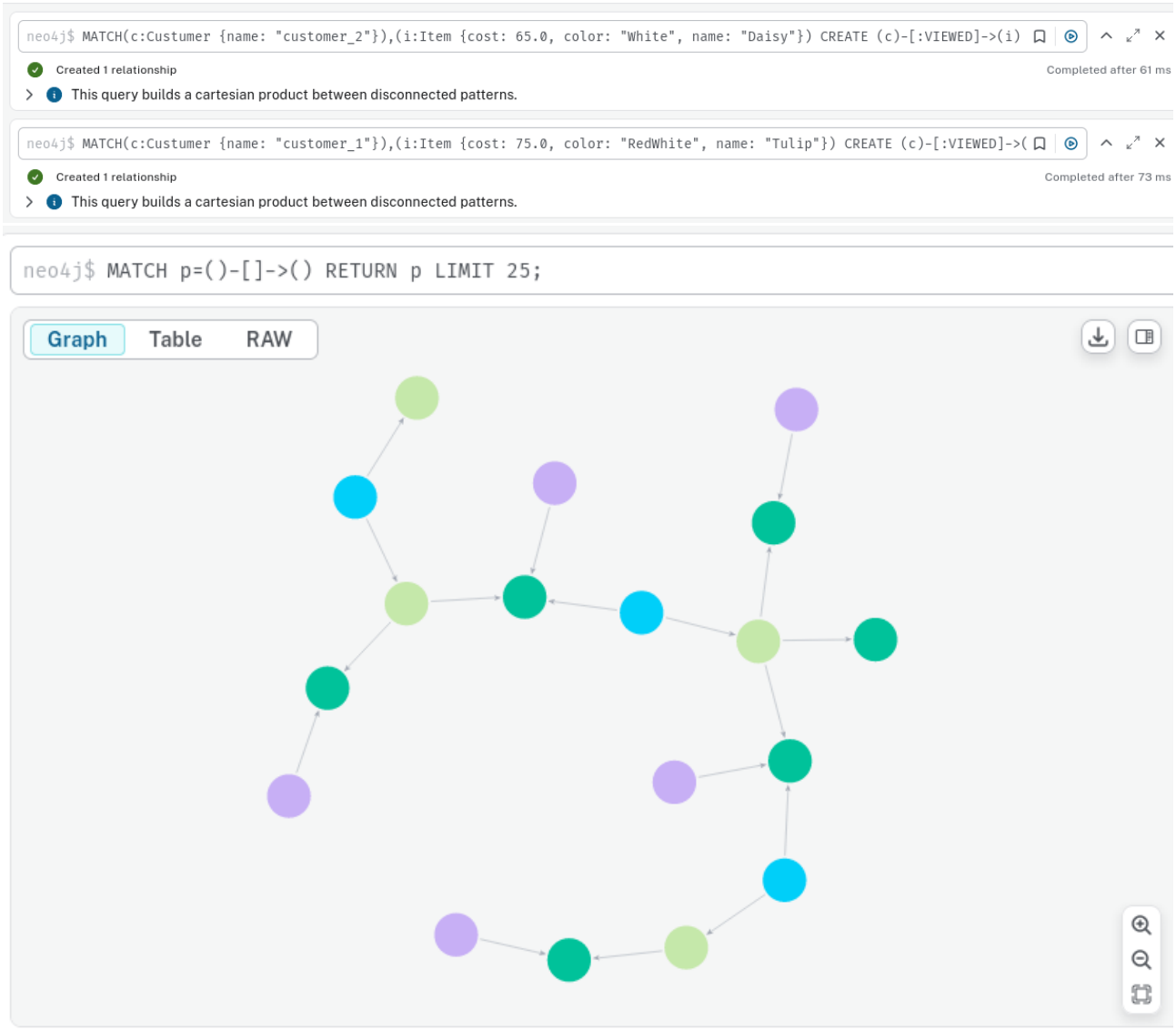
Created 1 relationship — Completed after 15 ms

This query builds a cartesian product between disconnected patterns.

```
neo4j$ MATCH(o:Order {identifier: "Order_1"}),(i:Item {cost: 100.0, color: "Red", name: "Rose"}) CREATE (o)-[:CONTAINS]->(i)
```

Created 1 relationship — Completed after 16 ms

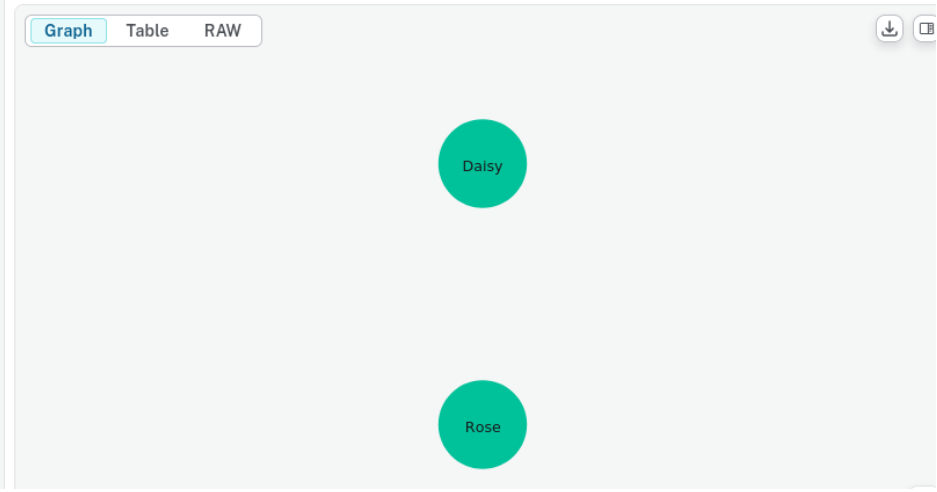
- Customer може переглядати (view), але при цьому не купувати Items



1. Написати наступні види запитів:

- Знайти Items які входять в конкретний Order

```
MATCH(o:Order {identifier: 'Order_1'})-[:CONTAINS]->(i:Item)
RETURN i
```



- Підрахувати вартість конкретного Order

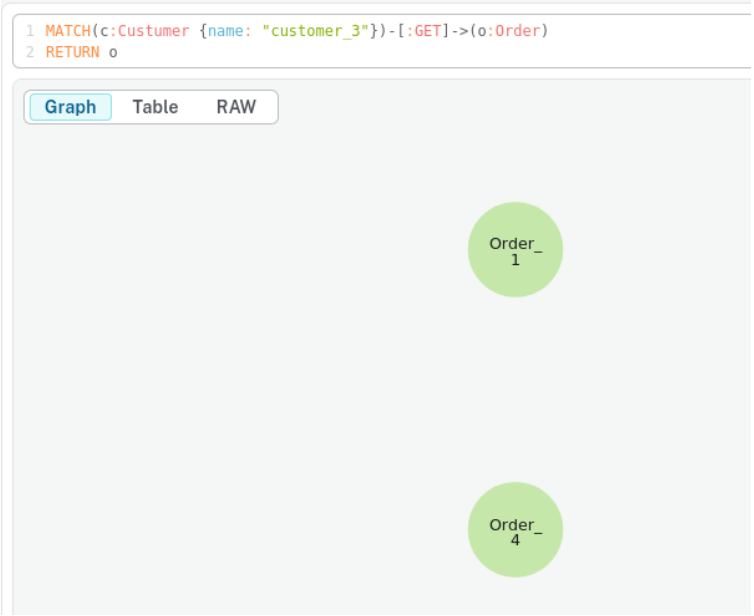
```
neo4j$ MATCH(o:Order{identifier: "Order_1"})-[:CONTAINS]->(i:Item) RETURN SUM(i.cost) AS total_cost
```

Table RAW

total_cost
165.0

- Знайти всі Orders конкретного Customer

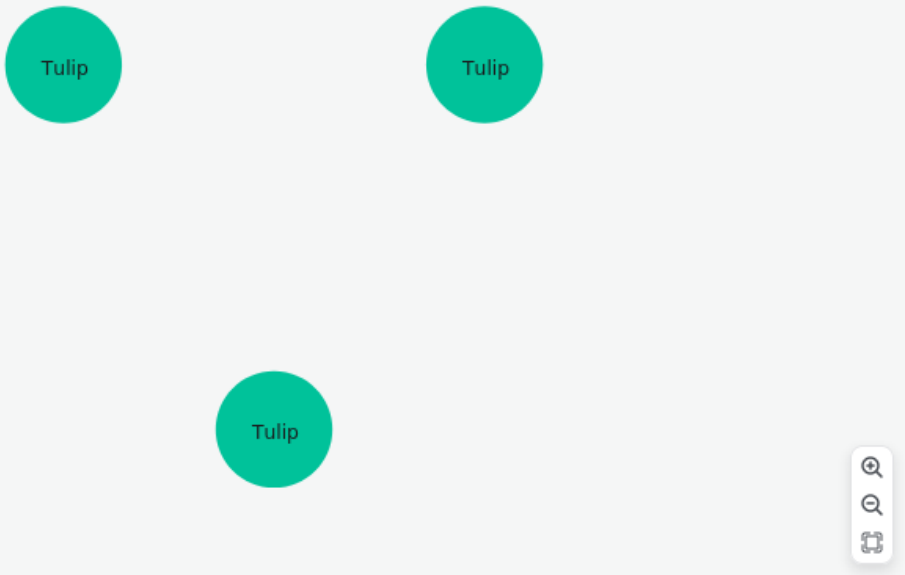
```
MATCH(c:Custumer {name: "customer_3"})-[:GET]->(o:Order)
RETURN o
```



- Знайти всі Items куплені конкретним Customer (через Order)

```
MATCH(c:Custumer {name: "customer_2"})-[:GET]->(o:Order)-[:CONTAINS]->(i:Item)
RETURN i
```

Graph Table RAW



- Знайти кількість Items куплені конкретним Customer (через Order)

```
neo4j$ MATCH(c:Custumer {name: "customer_2"})-[:GET]->(o:Order)-[:CONTAINS]->(i:Item) RETURN COUNT(i) AS total_items_amount
```

Table RAW

total_items_amount
3

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

```
MATCH(c:Custumer {name: "customer_2"})-[:GET]->(o:Order)-[:CONTAINS]->(i:Item)
RETURN SUM(i.cost) AS total_cost
```

Table RAW

total_cost
165.0

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

```
MATCH(i:Item)<-[:CONTAINS]-(o:Order)
RETURN i, COUNT(o) AS items_got ORDER BY items_got DESCENDING
```

Graph | Table | RAW

i	items_got
(:Item {cost: 100.0, color: "Red", name: "Rose"})	1
(:Item {cost: 65.0, color: "White", name: "Daisy"})	1
(:Item {cost: 115.0, color: "White", name: "Rose"})	1
(:Item {cost: 50.0, color: "White", name: "Tulip"})	1
(:Item {cost: 40.0, color: "Purple", name: "Tulip"})	1
(:Item {cost: 75.0, color: "RedWhite", name: "Tulip"})	1

```
MATCH(o:Order {identifier: "Order_2"}),(i:Item {cost: 65.0, color: "White", name: "Daisy"})
CREATE (o)-[:CONTAINS]->(i)
```

Created 1 relationship

This query builds a cartesian product between disconnected patterns.

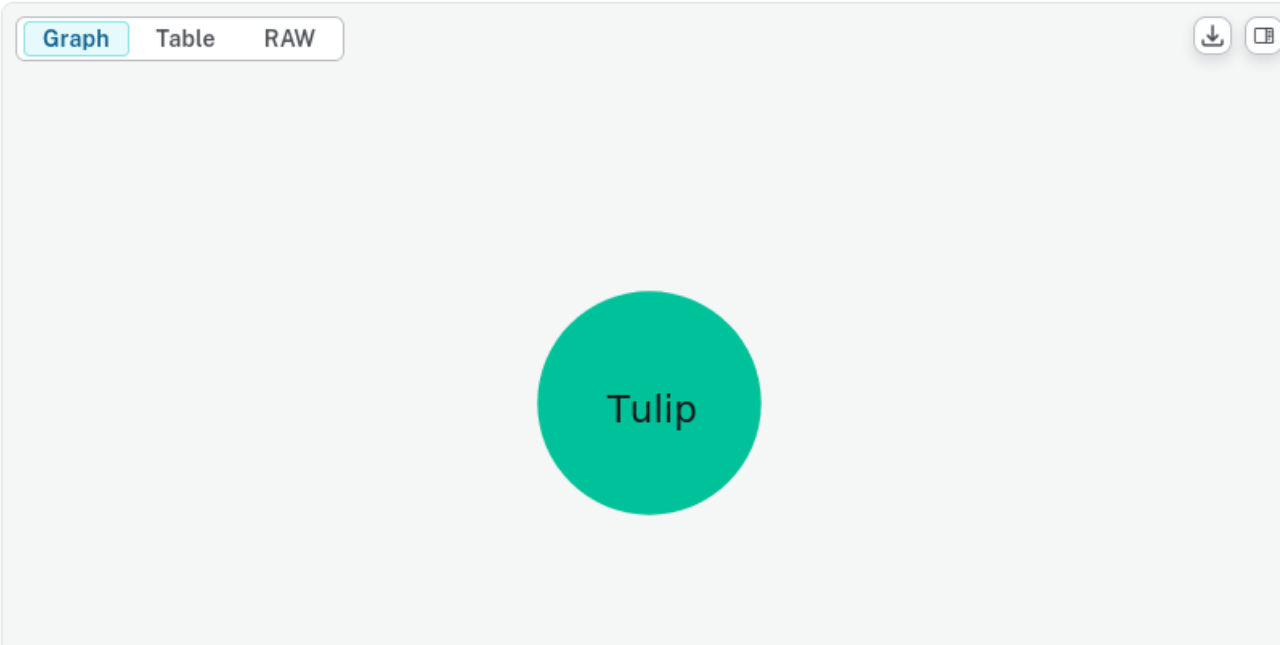
```
MATCH(i:Item)<-[:CONTAINS]-(o:Order)
RETURN i, COUNT(o) AS items_got ORDER BY items_got DESCENDING
```

Graph | Table | RAW

i	items_got
(:Item {cost: 65.0, color: "White", name: "Daisy"})	2
(:Item {cost: 100.0, color: "Red", name: "Rose"})	1
(:Item {cost: 115.0, color: "White", name: "Rose"})	1
(:Item {cost: 50.0, color: "White", name: "Tulip"})	1
(:Item {cost: 40.0, color: "Purple", name: "Tulip"})	1
(:Item {cost: 75.0, color: "RedWhite", name: "Tulip"})	1

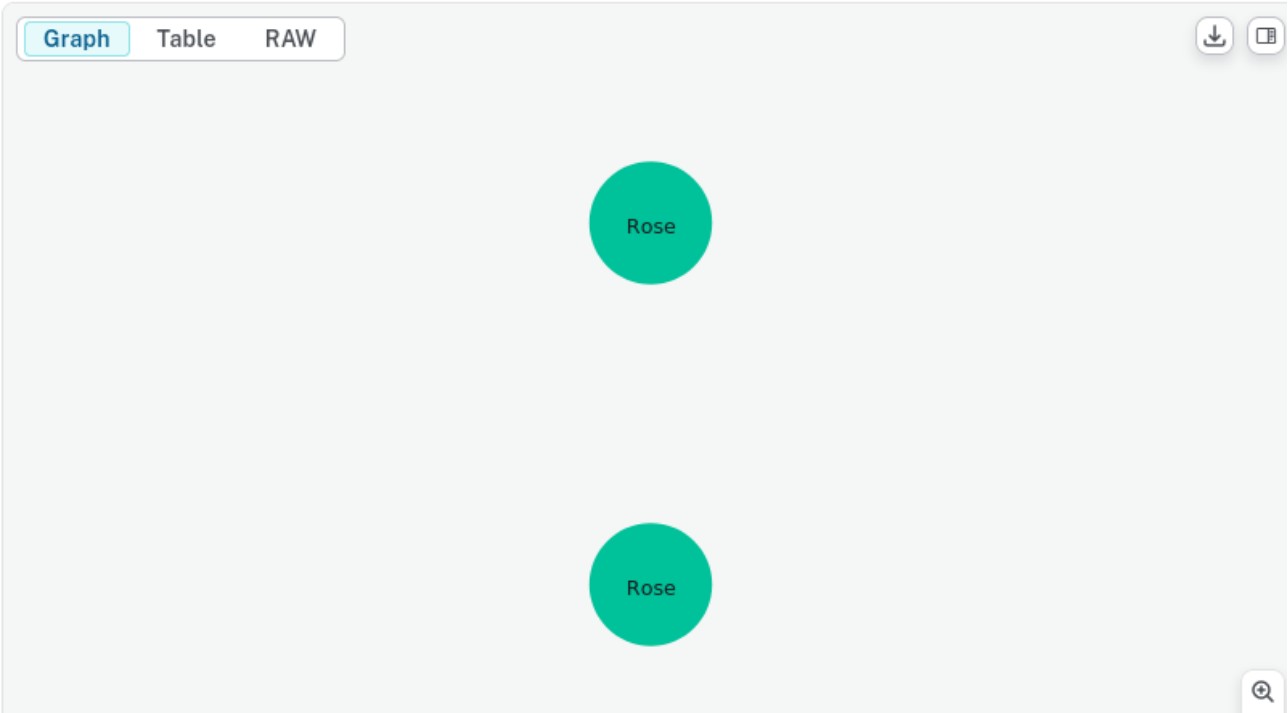
- Знайти всі Items переглянуті (view) конкретним Customer

```
MATCH(c:Custumer {name: "customer_1"})-[:VIEWED]->(i:Item)
RETURN i
```



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

```
MATCH(i:Item {cost: 65.0, color: "White", name: "Daisy"})<-[:CONTAINS]-(o:Order)-[:CONTAINS]->(i_:Item)
WHERE i<>i_
RETURN i_
```



- Знайти Customers які купили даний конкретний Item

```
MATCH(i:Item {cost: 65.0, color: "White", name: "Daisy"})<-[:CONTAINS]-(o:Order)<-[:GET]-(c:Custumer)
RETURN c
```

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

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

- З 10 окремих клієнтів одночасно запустити інкерементацію каунтеру лайків по 10_000 на кожного клієнта
- зробіть так щоб не було втрат та перевірте щоб фінальне значення було 100_000
- заміряйте час роботи

Instance01 • RUNNING

Overview | Snapshots | Import instance | Logs

ID: 5639217d

Connection URI: neo4j+s://5639217d.databases.neo4j.io

Query API URL: https://5639217d.databases.neo4j.io/db/{databaseName}/query/v2

Version: 5

Region: GCP / Belgium (europe-west1)

Nodes: 0 (0%)

Relationships: 0 (0%)

Type: Free

```
neo4j$ CREATE(:Item_plus {id: 1, name: 'Wisteria', price: 750, likes: 0});
```

Created 1 node, set 4 properties, added 1 label

```
neo4j$ UNWIND range(1, 10) AS id CREATE (:Customer_plus {id: id, name: 'Customer_' + id});
```

Created 10 nodes, set 20 properties, added 10 labels

Database information

Nodes (29)

* Customer_plus Custumer Item Item_plus Order

Relationships (18)

* CONTAINS GET VIEWED

Property keys

color cost data id identifier likes name nodes price relationships style visualisation

```
┌──(ivan㉿kali)-[~/pvns/3]
└─$ python lab3.py
Final amount of likes ── 100000
Lasted ── 416.34653878211975 seconds
```

```
neo4j$ MATCH (n:Item_plus) RETURN n LIMIT 25;
```

Graph | Table | RAW

n
(:Item_plus {price: 750, name: "Wisteria", id: 1, likes: 100000})